



## **64-bit Imports Rebuilding and Unpacking**

Sebastien Doucet <[sdoucet@ncircle.com](mailto:sdoucet@ncircle.com)>

# Who am I?

---

- Security Research Engineer at nCircle in Toronto
- My accent is from Montreal
- Used to be involved in the online reverse-engineering community
  - Moderator at [reverse-engineering.net](http://reverse-engineering.net) forum
  - Co-founder of [video.reverse-engineering.net](http://video.reverse-engineering.net) (defunct)
  - Most evil moderator at [crackmes.de](http://crackmes.de)
  - Member of ARTeam

# Warning

---

- Used to be a 60-minutes presentation
- Had to take away all the jokes
- Only dry technical stuff left
- Focus is on the unpacking process instead of the tool
- For more information or the full deck of slides, come see me at the nCircle booth

# Looking at the Bigger Picture

- More 64-bit malware is appearing every day
- Next generation of Windows will probably be 64-bit only like Windows Server 2008 R2
- Programming languages limited to a specific architecture like C and C++ increase the difficulty in dealing with cross-architecture tasks like unpacking
- Platform-independent languages like Python simplify things but still require you to know what you are doing when dealing with the PE format and headers
- If I had to redo it today, I would do it in C#

# Overview

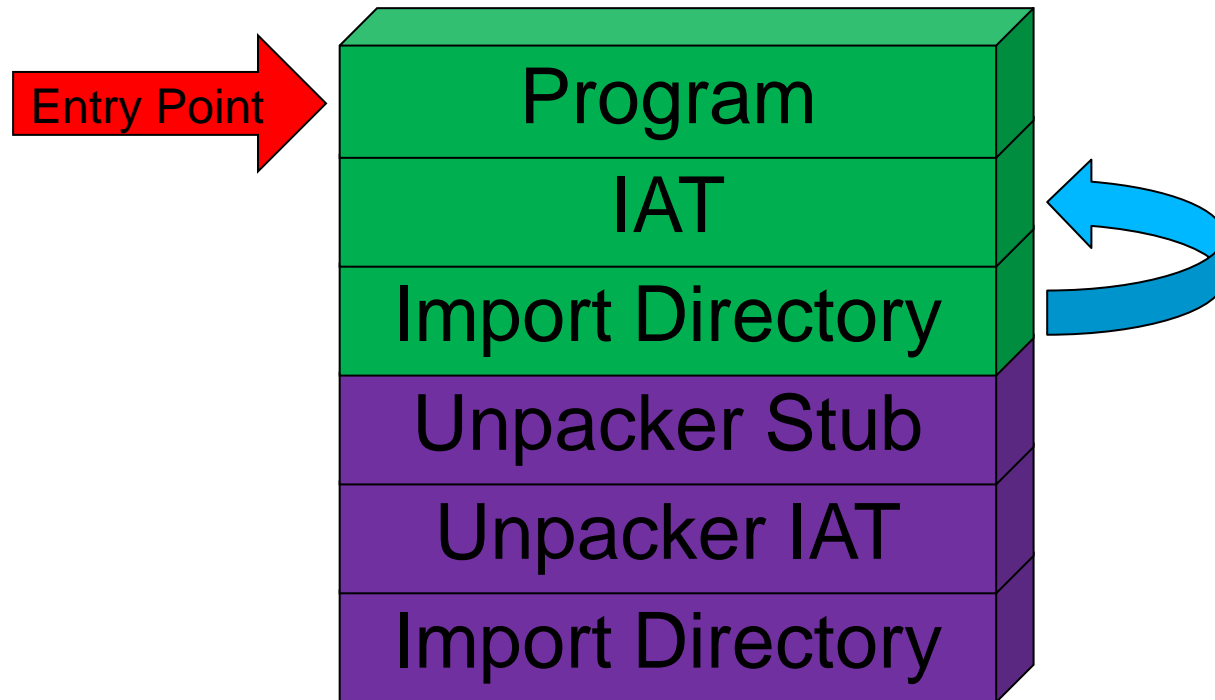
---

- Part 1: What is unpacking?
- Part 2: What is ImpREC? (original tool)
- Part 3: What is CHimpREC? (my tool)
- Part 4: Inner workings of an imports rebuilder
- Part 5: Live 64-bit unpacking session

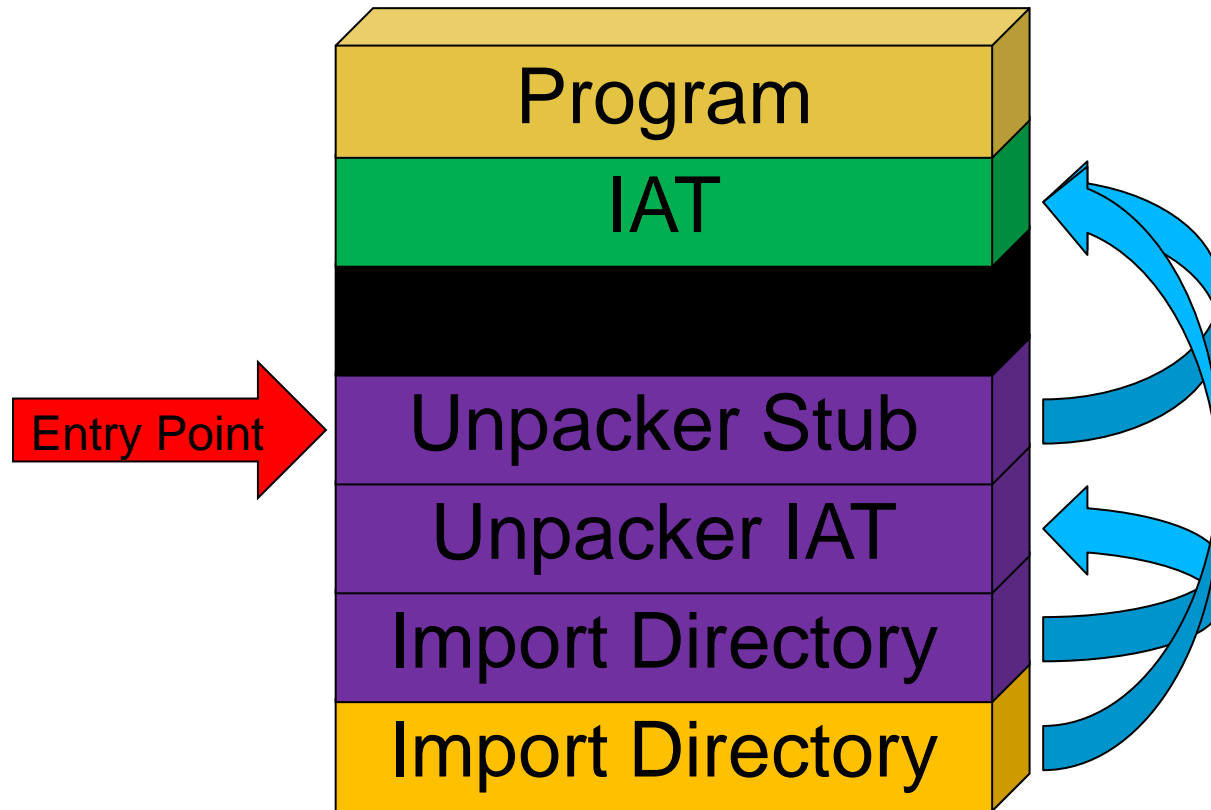
# Part 1: What is unpacking?

- Packers are designed to protect the content of an executable binary or library
  - Commercial software (game copy protection)
  - Malware
- Most packers use encryption or compression
- Original assembly code not accessible
- Make static analysis impossible or at least, very hard
- Called a “shell” in Chinese

# How simple packers work



# General unpacking theory

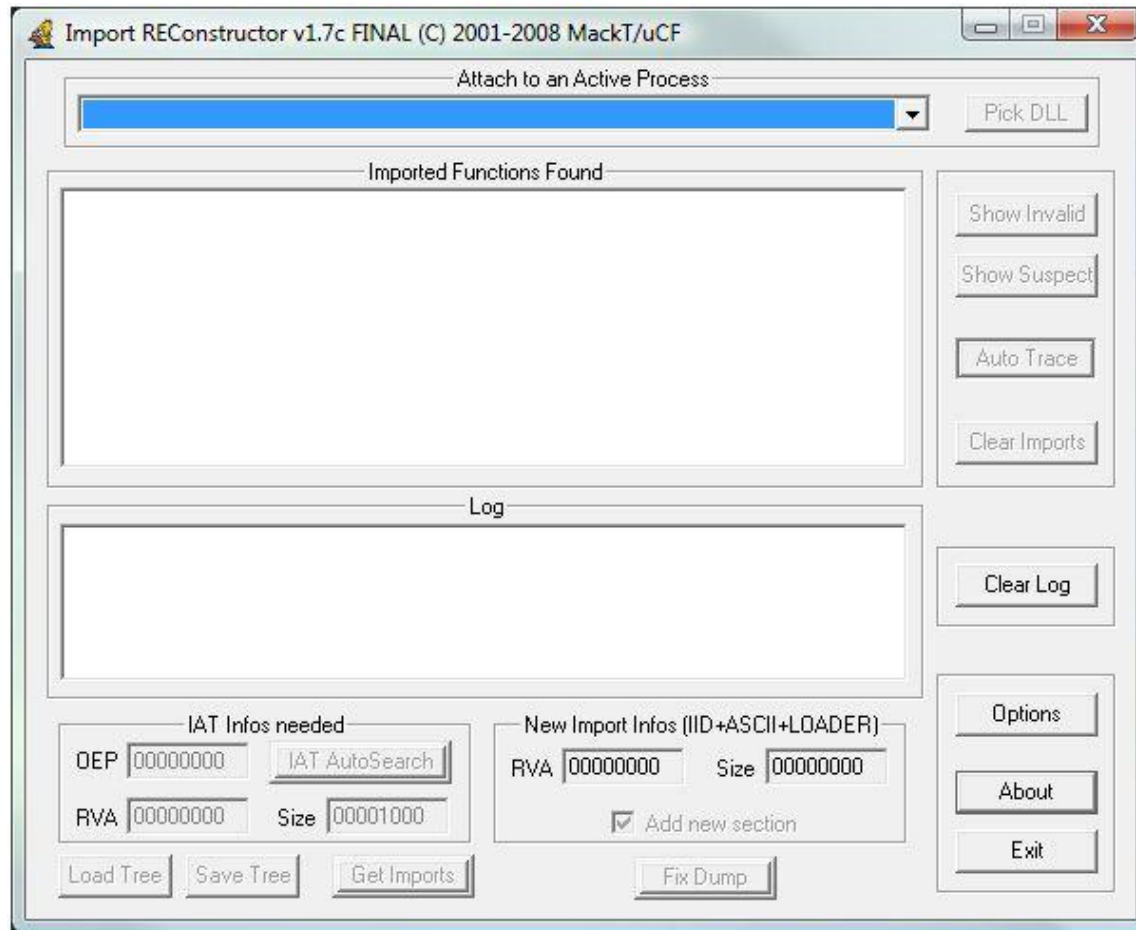




## Part 2: What is ImpREC?

- 32-bit only imports rebuilder
- Saves a lot of time over the manual rebuilding method
- Wasn't designed to deal with “features” of Vista
- No documentation or source code was available
- It did its job really well though

# Part 2: What is ImpREC?



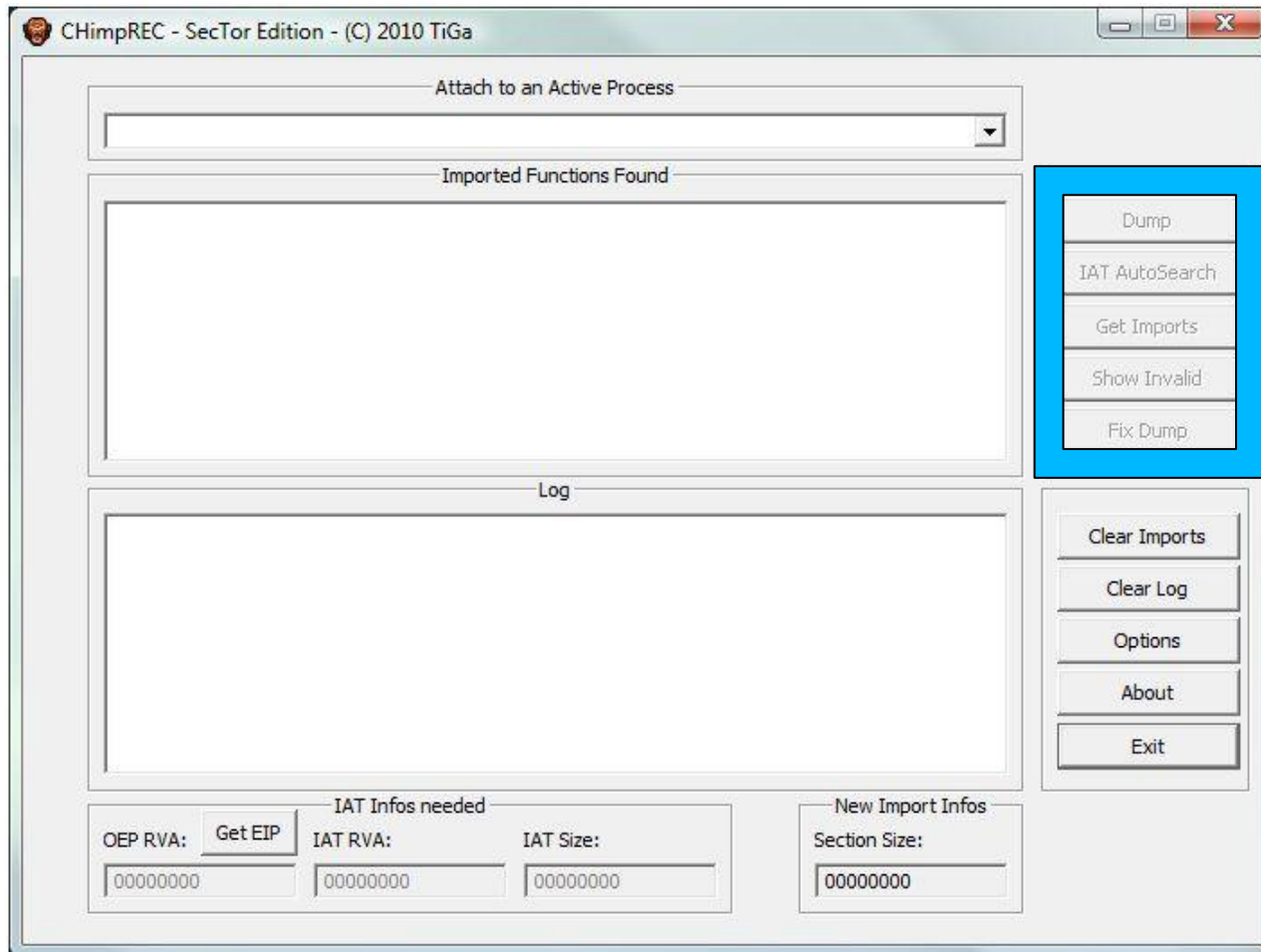
# Why do this project?

- ImpREC was getting older.
- There were no public 64-bit imports rebuilder freely available on the internet at the time.
- Some functionality was missing (process dumper)
- I was curious.  
I am a reverser.  
It's what I do.
- Got tired of waiting for somebody else to do it.  
So I made it by myself.

## Part 3: What is CHimpREC?

- 32 and 64-bit imports rebuilder
- Improved version of ImpREC
- Fixes many existing bugs
- Introduces new features
- Made especially for WoW64 compatibility
- Allows for an all-in-one version
- Chinese version available (unexpectedly)
- Done entirely through black box reverse-engineering
- Done by making up equivalent operations that yield exactly the same results in all possible scenarios

# Part 3: What is CHimpREC?

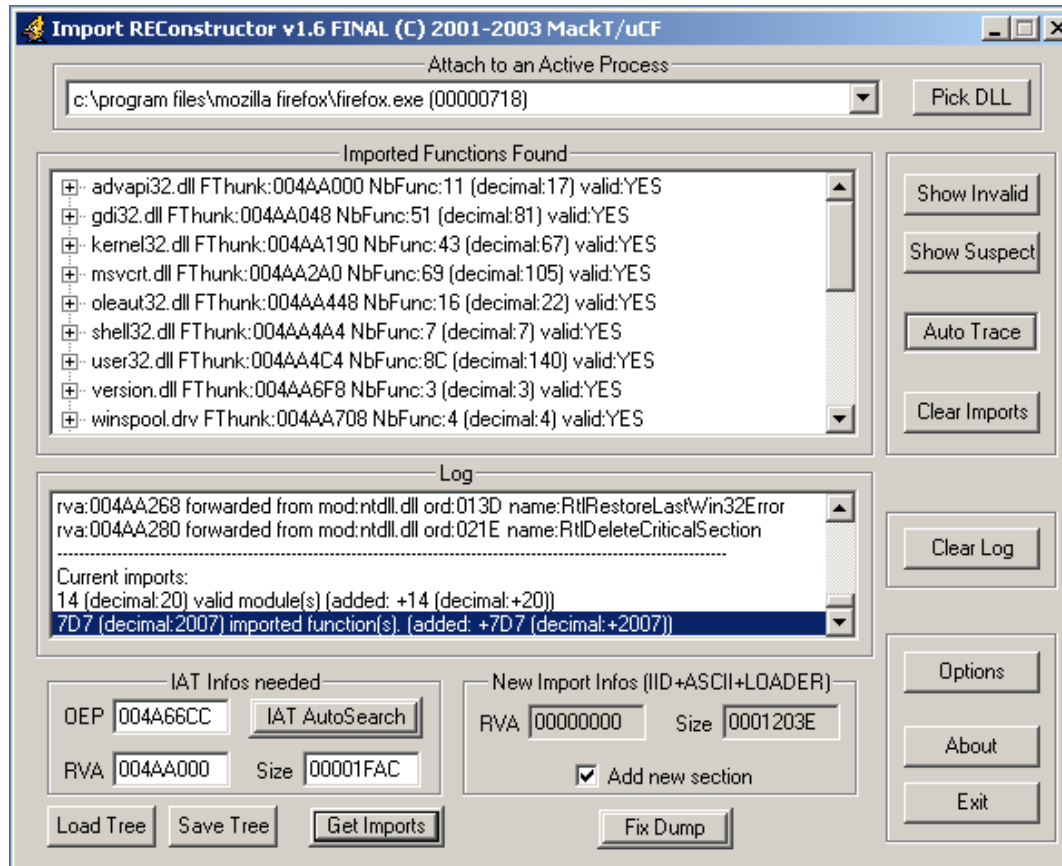


# What is CHimpREC? (Chinese)



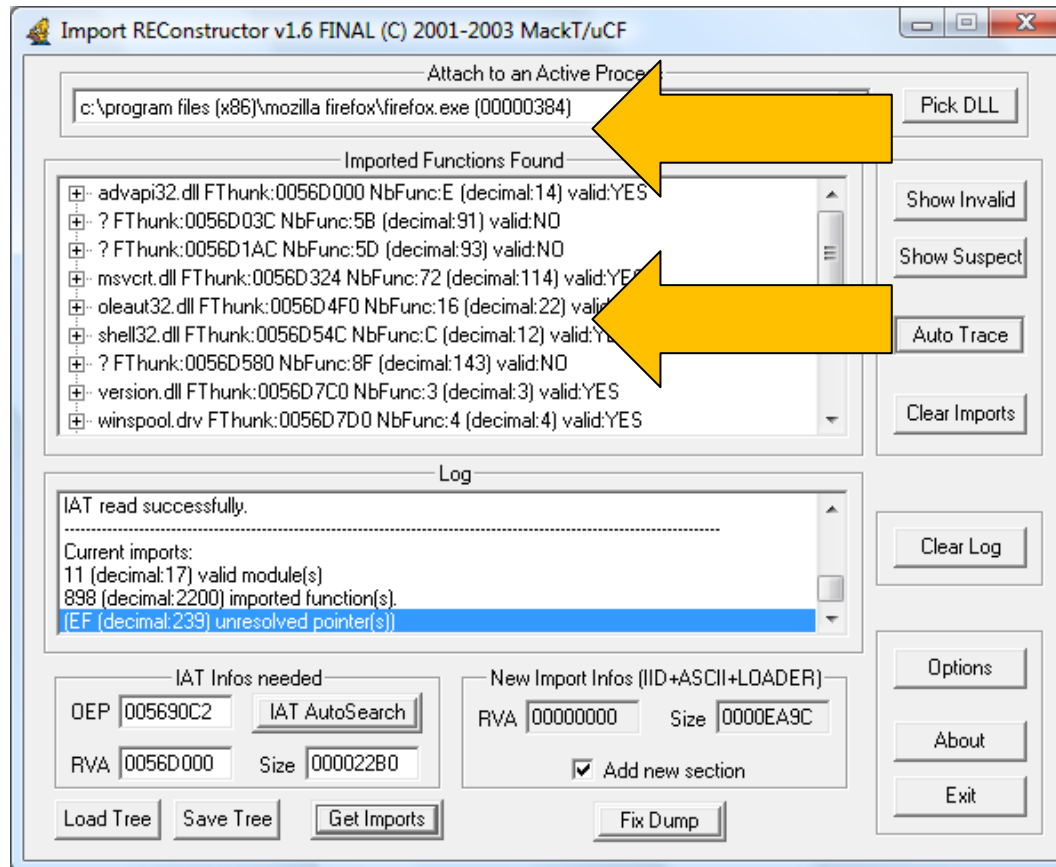
# Limitations of ImpREC

## XP or Vista w/o ASLR



# Limitations of ImpREC

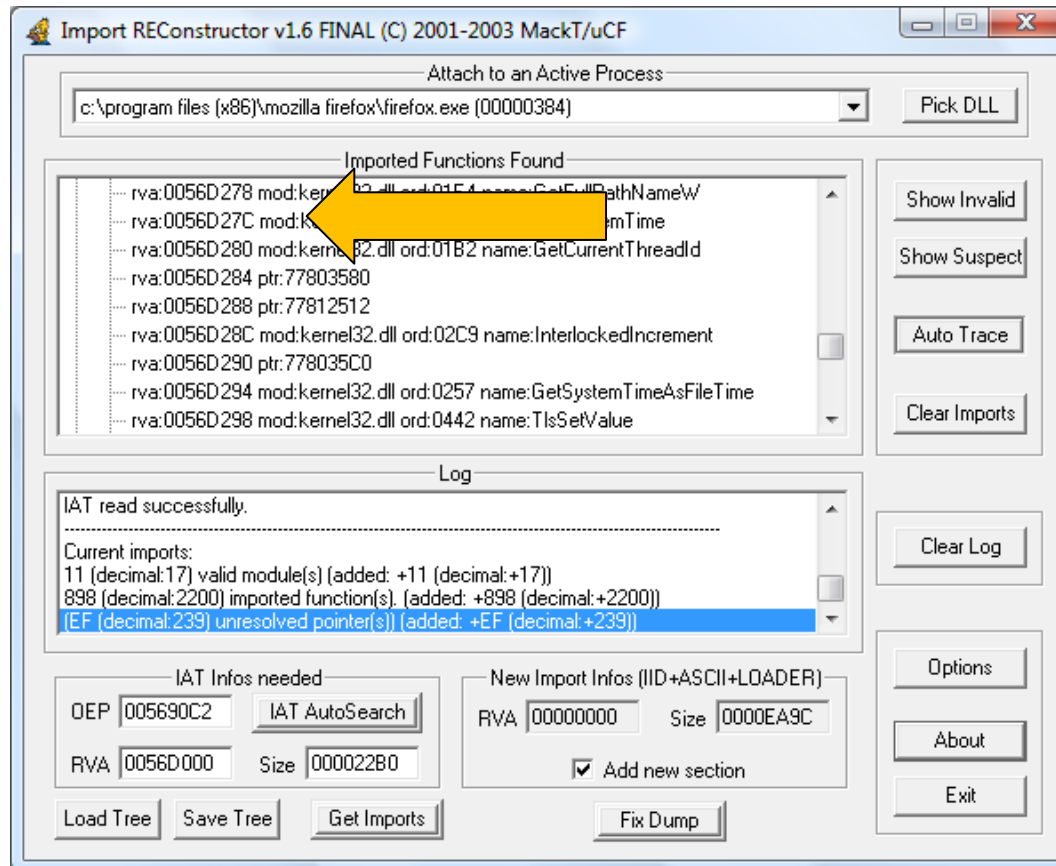
## Vista ASLR





# Limitations of ImpREC

## Vista ASLR







# Limitations of ImpREC






## Vista ASLR






### GDI32






### KERNEL32






### USER32






 gdi32.dll	77F10000	77F11000	R	.	.	D	.
 gdi32.dll	77F11000	77F53000	R	.	X	D	.
 gdi32.dll	77F53000	77F54000	R	W	.	D	.
 gdi32.dll	77F54000	77F57000	R	.	.	D	.

 kernel32.dll	7C800000	7C801000	R	.	.	D	.
 kernel32.dll	7C801000	7C883000	R	.	X	D	.
 kernel32.dll	7C883000	7C886000	R	W	.	D	.
 kernel32.dll	7C886000	7C888000	R	W	.	D	.
 kernel32.dll	7C888000	7C8F4000	R	.	.	D	.

 user32.dll	77D40000	77D41000	R	.	.	D	.
 user32.dll	77D41000	77DA0000	R	.	X	D	.
 user32.dll	77DA0000	77DA1000	R	W	.	D	.
 user32.dll	77DA1000	77DA2000	R	W	.	D	.
 user32.dll	77DA2000	77DD0000	R	.	.	D	.

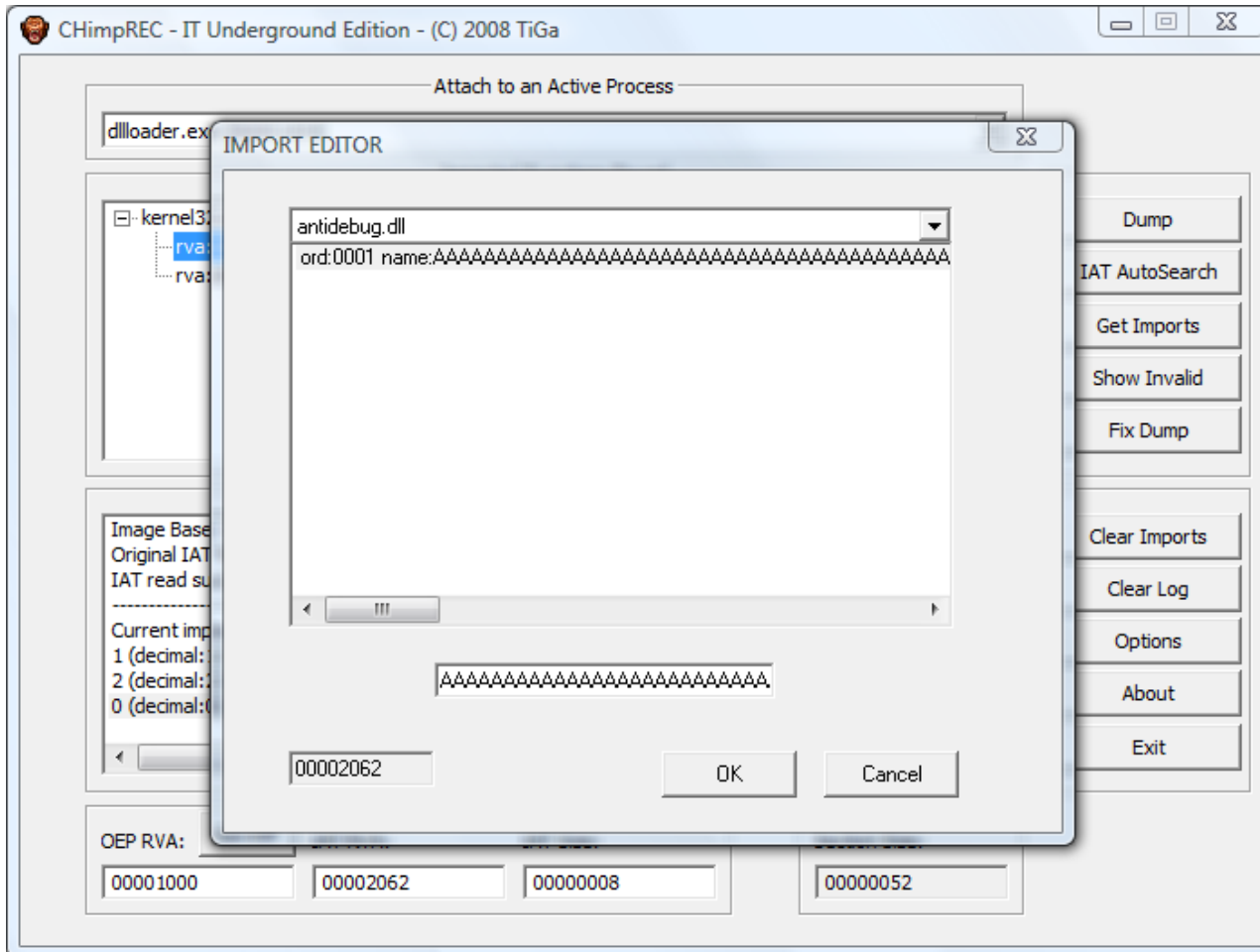
 gdi32.dll	77100000	77101000	R	.	.	D	.
 gdi32.dll	77110000	77157000	R	.	X	D	.
 gdi32.dll	77160000	77162000	R	W	.	D	.
 gdi32.dll	77170000	77171000	R	.	.	D	.
 gdi32.dll	77180000	77182000	R	.	.	D	.

 kernel32.dll	76590000	76591000	R	.	.	D	.
 kernel32.dll	765A0000	76665000	R	.	X	D	.
 kernel32.dll	76670000	76673000	R	W	.	D	.
 kernel32.dll	76680000	76681000	R	.	.	D	.
 kernel32.dll	76690000	7669A000	R	.	.	D	.

 user32.dll	76F80000	76F81000	R	.	.	D	.
 user32.dll	76F90000	76FFD000	R	.	X	D	.
 user32.dll	77000000	77002000	R	W	.	D	.
 user32.dll	77010000	7703E000	R	.	.	D	.
 user32.dll	77040000	77044000	R	.	.	D	.

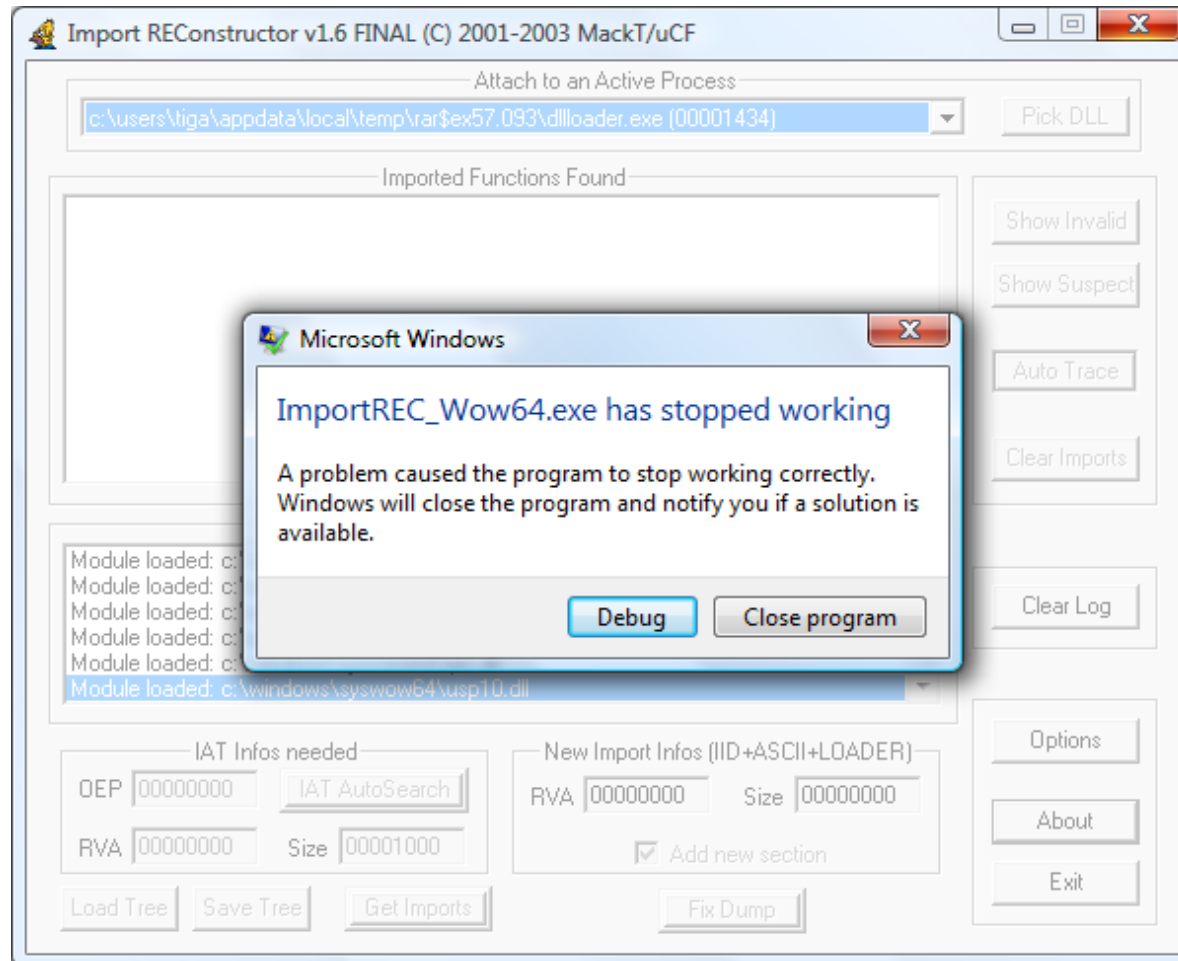
# Limitations of ImpREC

## Buffer Overflow Vulnerabilities



# Limitations of ImpREC

## Buffer Overflow Vulnerabilities



## Part 4: Inner workings of an import rebuilder

- API families: Toolhelp32 vs. PSAPI
- How planning efficiently can save time
- 5-steps of the process:
  - Dump
  - IAT AutoSearch
  - Get Imports (Unforwarding)
  - Show Invalid
  - Fix Dump

# Toolhelp32 vs. PSAPI

- Toolhelp32 APIs
  - CreateToolhelp32Snapshot
  - Process32First
  - Process32Next
  - Module32First
  - Module32Next
  - ToolHelp32ReadProcessMemory

# Toolhelp32 vs. PSAPI

- PSAPI APIs
  - EnumProcesses
  - EnumProcessModules
  - EnumProcessModulesEx
  - GetModuleInformation
  - GetModuleBaseName
  - GetModuleFileNameEx

# Toolhelp32 vs. PSAPI

## Windows Version Compatibility

	95	98	Me	NT4	2000	2003	XP	Vista
CreateToolhelp32Snapshot	X	X	X		X	X	X	X
EnumProcessModules				X	X	X	X	X
EnumProcessModulesEx								X



# How planning efficiently can save time

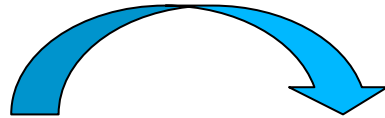
- 2 Single-Architecture versions (x86 OR x64)
  - To each his own
  - APIs: CreateToolhelp32Snapshot
  - Best OS compatibility range
  - Allows for common project source and headers
  - Coded in 32-bit then ported to 64-bit
- Cross-Architecture All-in-one version (x86 AND x64)
  - Made from a different x64 project
  - Requires 64-bit OS
  - EnumProcessModules & Ex
  - Runs on Vista x64 and Windows 7 only (not XP)

# Step 1: Dump

- Copying the memory area of a process to a file
- When the process has reached its Original Entry Point
- Each section is dumped individually
- Each section RawSize must be realigned from FileAlignment to SectionAlignment
- RawAddress matches VirtualAddress
- All sections are made writable by adding the flag:
  - IMAGE\_SCN\_MEM\_WRITE
- VirtualProtectEx to change the process memory to:
  - PAGE\_EXECUTE\_READWRITE

# Step 1: Dump

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
UPX0	00005000	00001000	00000000	00000400	00000000	00000000	0000	0000	E0000080
UPX1	00001000	00006000	00000600	00000400	00000000	00000000	0000	0000	E0000040
.rsrc	00001000	00007000	00000200	00000A00	00000000	00000000	0000	0000	C0000040



Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
UPX0	00005000	00001000	00005000	00001000	00000000	00000000	0000	0000	E0000080
UPX1	00001000	00006000	00001000	00006000	00000000	00000000	0000	0000	E0000040
.rsrc	00001000	00007000	00001000	00007000	00000000	00000000	0000	0000	C0000040



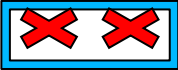
## Step 2: IAT AutoSearch

- Binary search looking for indirect call opcodes:
  - 8B0D MOV ECX,[ADDRESS]
  - 8B15 MOV EDX,[ADDRESS]
  - 8B1D MOV EBX,[ADDRESS]
  - 8B25 MOV ESP,[ADDRESS]
  - 8B2D MOV EBP,[ADDRESS]
  - 8B35 MOV ESI,[ADDRESS]
  - 8B3D MOV EDI,[ADDRESS]
  - A1 MOV EAX,[ADDRESS]

## Step 2: IAT AutoSearch

- Binary search looking for direct call opcodes:
  - FF15 CALL [ADDRESS]
  - FF25 JMP [ADDRESS]
  - FF35 PUSH [ADDRESS]
- Binary search ignores relative calls in 32-bit
- Starting from ImageBase or EntryPoint
- Found call must lead to a valid import
- Search up for the beginning of the IAT
- Search down for the end of the IAT
- Just like trying to identify a weird object in the dark

# Step 2: IAT AutoSearch



```
public start
start proc near
6A 00          push    0          ; lpModuleName
E8 81 00 00 00 call    GetModuleHandleA
A3 04 30 40 00 mov     hInstance, eax
6A 00          push    0          ; dwInitParam
68 29 10 40 00 push   offset DialogFunc ; lpDialogFunc
6A 00          push    0          ; hWndParent
6A 65          push    65h        ; lpTemplateName
FF 35 04 30 40 00 push   hInstance   ; hInstance
E8 54 00 00 00 call    DialogBoxParamA
6A 00          push    0          ; uExitCode
E8 59 00 00 00 call    ExitProcess
start endp
```

```
.text:00401088          ; ===== S U B R O U T I N E =====
.text:00401088          ; Attributes: thunk
.text:00401088          ; HMODULE __stdcall GetModuleHandleA(LPCSTR lpModuleName)
.text:00401088          GetModuleHandleA proc near          ; CODE XREF: start+2fp
.text:00401088          FF 25 00 20 40 00          jmp     ds: __imp_GetModuleHandleA
.text:00401088          GetModuleHandleA endp
```



## Step 2: IAT AutoSearch

```
.idata:00402000 ; Imports from kernel32.dll
.idata:00402000 ;
.idata:00402000 ; =====
.idata:00402000 ; Segment type: Externs
.idata:00402000 ; _idata
.idata:00402000 ; HMODULE __stdcall GetModuleHandleA(LPCSTR lpModuleName)
.idata:00402000         extrn __imp_GetModuleHandleA:dword
.idata:00402000                                 ; DATA XREF: GetModuleHandleA↑r
.idata:00402004 ; void __stdcall ExitProcess(UINT uExitCode)
.idata:00402004         extrn __imp_ExitProcess:dword ; DATA XREF: ExitProcess↑r
.idata:00402008 ;
.idata:0040200C ; Imports from user32.dll
.idata:0040200C ;
.idata:0040200C ; BOOL __stdcall EndDialog(HWND hDlg, INT_PTR nResult)
.idata:0040200C         extrn __imp_EndDialog:dword ; DATA XREF: EndDialog↑r
.idata:00402010 ; INT_PTR __stdcall DialogBoxParamA(HINSTANCE hInstance, LPCSTR lpTemplateName, HWND hWndParent,
.idata:00402010         extrn __imp_DialogBoxParamA:dword
.idata:00402010                                 ; DATA XREF: DialogBoxParamA↑r
.idata:00402014 ; =====
.rdata:00402014 ;
```

## Step 3: Get Imports

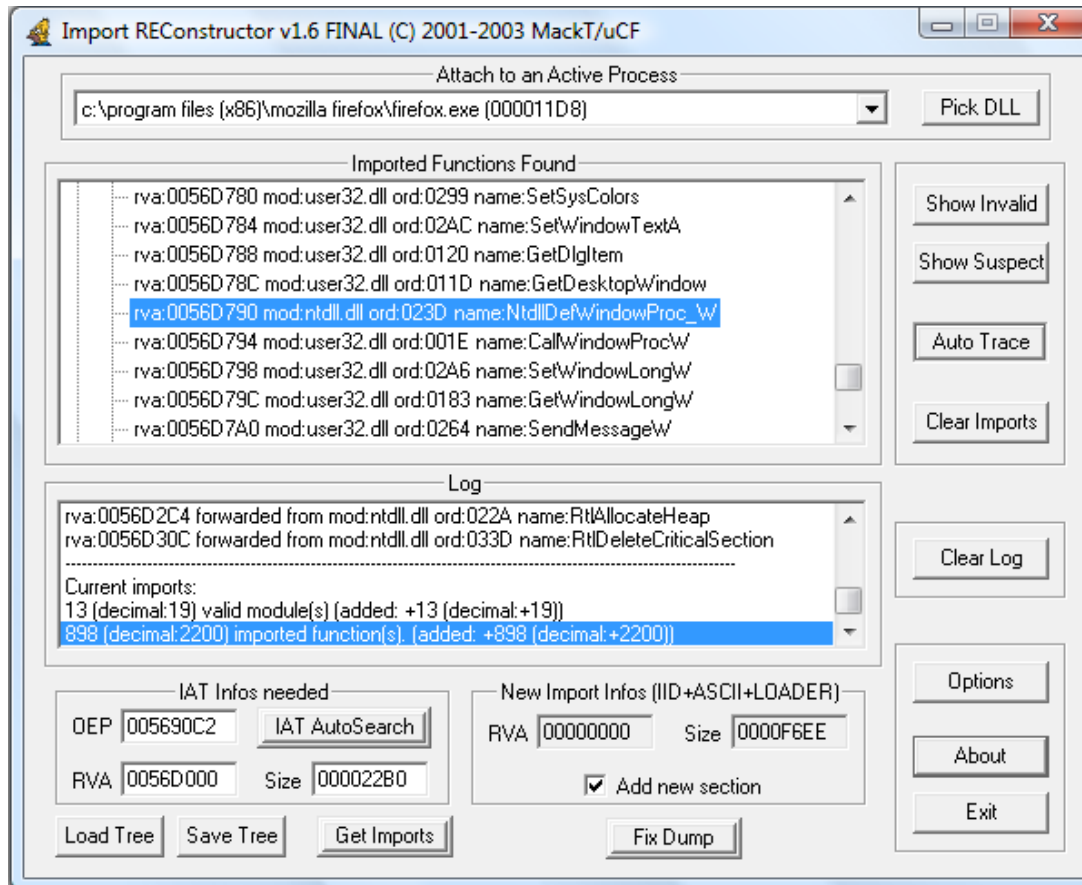
- Identify the elements of the IAT in the specified range
- Exactly the contrary of GetProcAddress
- Using custom-made reusable functions:
  - GetProcModuleName
  - GetProcName
  - GetProcOrdinal
  - GetProcNameAndOrdinal
  - GetProcInfo
  - Unforward



## Step 3: Get Imports (Unforwarding)

- The Entry Point of the function is not code but a string
- Imports are forwarded for compatibility between all the different versions of Windows
- If an import can be unforwarded, it doesn't mean that it really was forwarded
- There are many false-positives
- Must analyze the context with some fuzzy logic
- Could be called guessing too

# Step 3: Get Imports (Unforwarding)



# Step 3: Get Imports (Unforwarding)

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
N/A	000029DC	00004106	00003608	0000500E
(nFunctions)	Dword	Word	Dword	szAnsi
00000096	0001751E	0095	00014BF0	DefWindowProcA
00000097	00017539	0096	00014BFF	DefWindowProcW

```

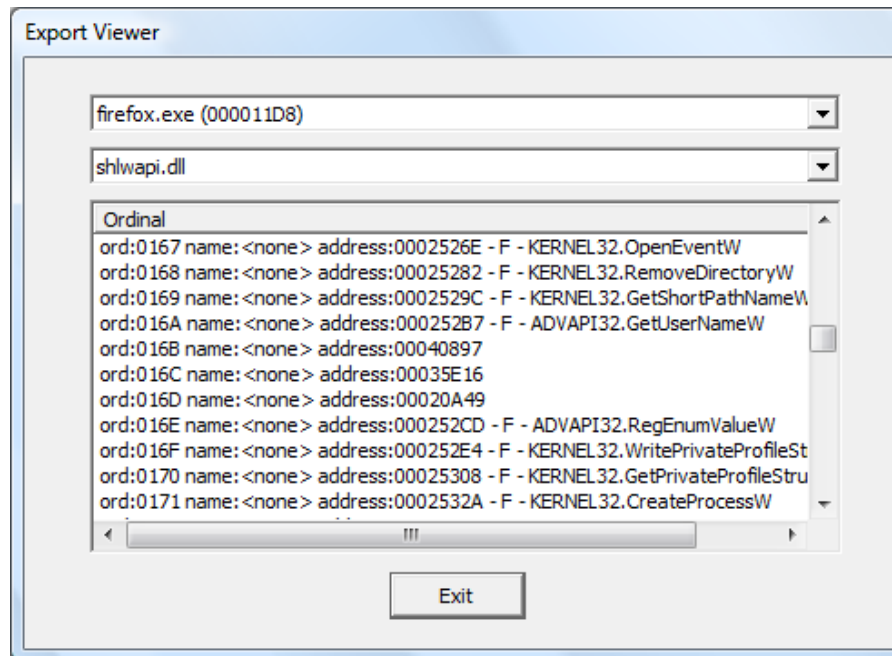
user32.dll:76460000 user32_dll segment byte public 'CONST' use32
user32.dll:76460000 assume cs:user32_dll
user32.dll:76460000 ;org 76460000h
user32.dll:764774E8 aNtdll_ntd1ld_1 db 'NTDLL.NtdllDialogWndProc_A',0
user32.dll:76477503 aNtdll_ntd1ldia db 'NTDLL.NtdllDialogWndProc_W',0
user32.dll:7647751E aNtdll_ntd1ldef db 'NTDLL.NtdllDefWindowProc_A',0
user32.dll:76477539 aNtdll_ntd1ld_0 db 'NTDLL.NtdllDefWindowProc_W',0
  
```

```

.idata:0096D790 ; LRESULT __stdcall DefWindowProcW(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)
.idata:0096D790 4D 3D 84 77 DefWindowProcW dd offset ntdll_NtdllDefWindowProc_W
  
```

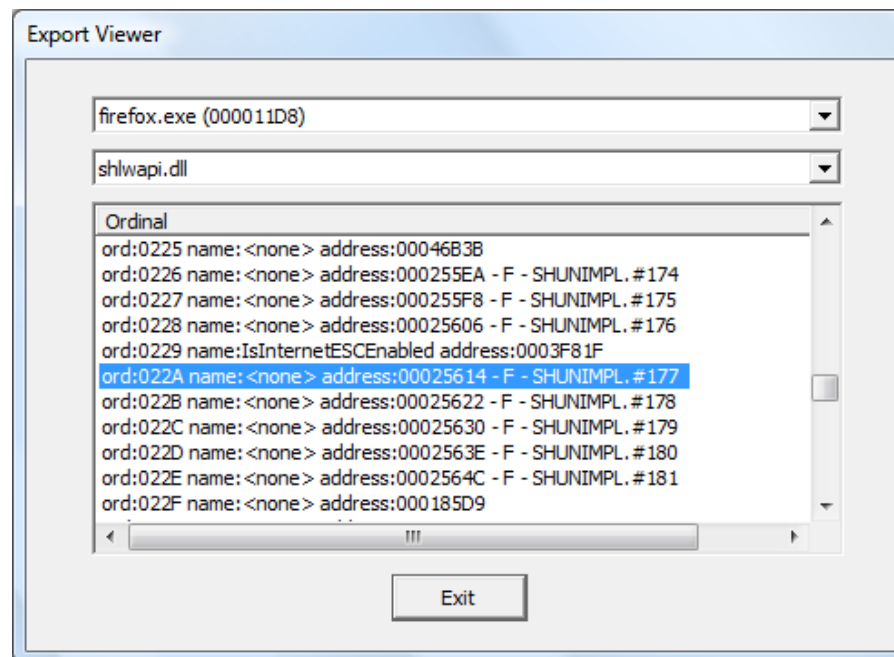
## Step 3: Get Imports (Unforwarding)

### False-positives



## Step 3: Get Imports (Unforwarding)

### Forwarding by ordinal



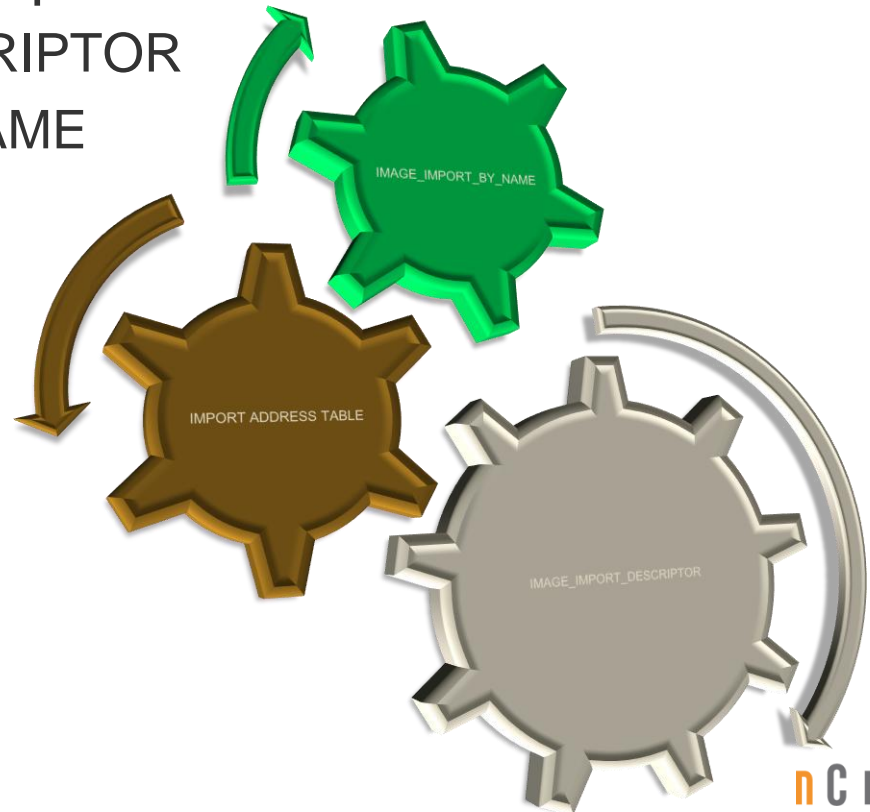
## Step 4: Show Invalid

---

- Display unidentified IAT entries
- Text search through the interface
- Check all imports one by one for validity
- Simplest step to implement

## Step 5: Fix dump

- Recreate the Import Directory to satisfy the loader
- Restore the original IAT
- Assemble structures that point to each other
  - IMAGE\_IMPORT\_DESCRIPTOR
  - IMAGE\_IMPORT\_BY\_NAME
- Like gears in a clock



# Changes from PE to PE32+ format

- All registers extended to QWORDS
  - EAX -> RAX
  - ESP -> RSP
- New registers
  - R8X-R15X
- All DLLs used must be 64-bit
- BaseOfData has disappeared
- New calling convention for APIs



# Changes in the imports rebuilding process

- IAT elements are QWORDS
- Pointer to Original First Thunk is a QWORD
- ImageBase is a QWORD
- Exception Handlers are now stored as structures in the new PE32+ Exception Directory

## Part 5: Live 64-bit unpacking session

---

- Tools used:
  - IDA Pro Advanced 64
  - CHimpREC-64
- Example: MPRESS 1.07
  - Simple UPX-like packer



Do you have any questions?

nCircle<sup>o</sup>



For more information or the full deck of slides, come see me at the nCircle booth.

nCircle<sup>o</sup>