# Lab 8: Tunnelling

In this lab we will investigate the usage of SSL/TLS and VPN tunnels.

## 1 Web cryptography assessment

The ssllabs tool (<https://ssllabs.com>) can be used to assess the security of the cryptography used on a Web site. You will be given a range of Web sites to scan in the lab, and you should pick three sites from the list. Now perform a test on them, and determine:

|  |  |  |  |
| --- | --- | --- | --- |
| **Site** | **Site 1:** | **Site 2:** | **Site 3:** |
| What grade does the site get? |  |  |  |
| The digital certificate key size and type? |  |  |  |
| Does the name of the site match the name on the server? |  |  |  |
| Who is the signer of the digital certificate? |  |  |  |
| The expiry date on the digital certificate? |  |  |  |
| What is the hashing method on the certificate? |  |  |  |
| If it uses RSA keys, what is the e value that is used in the encryption (Me mod N)? |  |  |  |
| Determine a weak cipher suite used and example why it might be weak? |  |  |  |
| What does TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384 identify? |  |  |  |
| Is SSL v2 supported? |  |  |  |
| If SSL v2 was supported, what problems might there be with the site (this will require some research)? |  |  |  |
| Outline the usage of TLS 1.0/1.1 and 1.2, and identify a problem if one of these TLS versions were not supported? |  |  |  |
| Is the site vulnerable to Heartbleed?  Is the site vulnerable to DROWN?  Is the site vulnerable to BEAST?  Is the site vulnerable to POODLE? |  |  |  |

Research questions:

If a site gets a ‘T’ grade, what is the problem?

If the site was susceptible to Poodle, what is the vulnerability?

## 2 Viewing details

|  |  |  |
| --- | --- | --- |
| **No** | **Description** | **Result** |
| **1** | Go to your Kali Linux instance. Run Wireshark and capture traffic from your main network connection. Start a Web browser, and go to **www.napier.ac.uk**.  Stop Wireshark and identify some of your connection details: | Your IP address and TCP port:  Napier’s Web server IP address and TCP port:  Right-click on the GET HTTP request from the client, and follow the stream:  What does the red and blue text identify?  Can you read the HTTP requests that go from the client to the server? [Yes][No] |
| **2** | Go to your Kali Linux instance. Run Wireshark and capture traffic from your main network connection. Start a Web browser, and go to **Google.com**.  Stop Wireshark and identify some of your connection details: | Your IP address and TCP port:  Google’s Web server IP address and TCP port:  Which SSL/TLS version is used:  By examining the Wireshark trace, which encryption method is used for the tunnel:  By examining the Wireshark trace, which hash method is used for the tunnel:  By examining the Wireshark trace, what is the length of the encryption key:  By examining the certificate from the browser which encryption method is used for the tunnel:  By examining the certificate from the browser, which hash method is used for the tunnel:  By examining the certificate from the browser is the length of the encryption key: |
| **3** | Run Wireshark and capture traffic from your main network connection. Start a Web browser, and go to **https://twitter.com**.  Stop Wireshark and identify some of your connection details: | Your IP address and TCP port:  Twitter’s Web server IP address and TCP port:  Which SSL/TLS version is used:  By examining the Wireshark trace, which encryption method is used for the tunnel:  By examining the Wireshark trace, which hash method is used for the tunnel:  By examining the Wireshark trace, what is the length of the encryption key:  By examining the certificate from the browser which encryption method is used for the tunnel:  By examining the certificate from the browser, which hash method is used for the tunnel:  By examining the certificate from the browser is the length of the encryption key: |

## 3 OpenSSL

|  |  |  |
| --- | --- | --- |
| **No** | **Description** | **Result** |
| **1** | Go to your Kali Linux instance, and make a connection to the **www.live.com** Web site:  openssl s\_client -connect www.live.com:443 | Which SSL/TLS method has been used:  Which method is used on the encryption key on the certificate, and what is the size of the public key?  Which is the handshaking method that has been used to create the encryption key?  Which TLS version is used for the tunnel?  Which encryption method is used for the tunnel:  Which hash method is used for the tunnel:  What is the length of the encryption key:  What is the serial number of the certificate:  Who has signed the certificate: |

## 4 Examining traces

|  |  |  |
| --- | --- | --- |
| **No** | **Description** | **Result** |
| **1** | Download the following file, and examine the trace with Wireshark:  http://asecuritysite.com/log/ssl.zip | Client IP address and TCP port:  Web server IP address and TCP port:  Which SSL/TLS method has been used:  Which encryption method is used for the tunnel:  Which hash method is used for the tunnel:  What is the length of the encryption key: |
| **2** | Download the following file, and examine the trace with Wireshark:  http://asecuritysite.com/log/https.zip | Client IP address and TCP port:  Web server IP address and TCP port:  Which SSL/TLS method has been used:  Which encryption method is used for the tunnel:  Which hash method is used for the tunnel:  What is the length of the encryption key: |
| **2** | Download the following file, and examine the trace with Wireshark:  http://asecuritysite.com/log/heart.zip | Client IP address and TCP port:  Web server IP address and TCP port:  Which SSL/TLS method has been used:  Which encryption method is used for the tunnel:  Which hash method is used for the tunnel:  What is the length of the encryption key:  Can you spot the packet which identifies the Heartbleed vulnerability? |
| **3** | Download the following file, and examine the trace with Wireshark:  http://asecuritysite.com/log/ipsec.zip | Which is the IP address of the client and of the server:  Which packet number identifies the start of the VPN connection (Hint: look for UDP Port 500):  Determine one of the encryption and hashing methods that the client wants to use:  Now determine the encryption and hashing methods that are agreed in the ISAKMP: |

# Part 2: Additional Lab

Link: <http://www.asecuritysite.com/csn09112/software01>

Video demo: <https://www.youtube.com/watch?v=raphJCH2SPE>

## 1 Number formats

Within cryptography we often have to present numbers in different formating, and typically have to convert from decimal into hexadecimal (based 16). Enter the following Python program:

import sys

val=10

if (len(sys.argv)>1):

val=int(sys.argv[1])

print "Hex: ",hex(val)

print "Decimal: ",val

print "Octal: ",oct(val)

print "Binary: ",bin(val)

Now use it to complete the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Decimal** | **Hex** | **Octal** | **Binary** |
| 10 |  |  |  |
|  | 0x23 |  |  |
|  |  | 032 |  |
|  |  |  | 1111000 |

## 2 Capturing packets

We will use Wireshark and WinPCap fairly extensively through the module. You can download the WinPCap Python script here. Put this into the default Python folder (such as c:\python27). Next create the following script:

## Based on code at <https://code.google.com/p/winpcapy>

import ctypes

from winpcapy import \*

import time

import sys

import string

# Packet capture function

PHAND=CFUNCTYPE(None,POINTER(c\_ubyte),POINTER(pcap\_pkthdr),POINTER(c\_ubyte))

## Callback function which is called for every new packet

def \_packet\_handler(param,header,pkt\_data):

local\_tv\_sec = header.contents.ts.tv\_sec

ltime=time.localtime(local\_tv\_sec);

timestr=time.strftime("%H:%M:%S", ltime)

print

print("%s,%.6d len:%d" % (timestr, header.contents.ts.tv\_usec, header.contents.len))

def get\_ad():

i=0

d=alldevs.contents

while d:

i=i+1

print("%d. %s" % (i, d.name))

print (" (%s)\n" % (d.description))

if d.next:

d=d.next.contents

else:

d=False

print ("Enter the interface number (1-%d):" % (i))

inum= raw\_input('--> ')

inum=int(inum)

d=alldevs

## Get Selected adaptor

for i in range(0,inum-1):

d=d.contents.next

return d.contents

## Define the Callback function name

packet\_handler=PHAND(\_packet\_handler)

alldevs=POINTER(pcap\_if\_t)()

errbuf= create\_string\_buffer(PCAP\_ERRBUF\_SIZE)

## Find all the devices

if (pcap\_findalldevs(byref(alldevs), errbuf) == -1):

print ("Error in pcap\_findalldevs: %s\n" % errbuf.value)

sys.exit(1)

## Get adapator

d=get\_ad()

adhandle = pcap\_open\_live(d.name,65536,1,1000,errbuf)

print("\nStarting to listen on %s...\n" % (d.description))

## Get 20 packets

pcap\_loop(adhandle, 20, packet\_handler, None)

pcap\_close(adhandle)

Run the script. What are the names of your interfaces?

For the first 20 packets, what is the minimum and maximum packet size?

## 3 Displaying IP addresses

Next we will parse the packets for the IP addresses. First add the following to define the parsing of the packets:

u\_short = c\_ushort

u\_char = c\_ubyte

u\_int = c\_int

class ip\_address(Structure):

\_fields\_ = [("byte1", u\_char),

("byte2", u\_char),

("byte3", u\_char),

("byte4", u\_char)]

class ip\_header(BigEndianStructure):

\_fields\_ = [("ver\_ihl", u\_char),

("tos", u\_char),

("tlen", u\_short),

("identification", u\_short),

("flags\_fo", u\_short),

("ttl", u\_char),

("proto", u\_char),

("crc", u\_short),

("saddr", ip\_address),

("daddr", ip\_address),

("op\_pad", u\_int)]

Next replace the call back function with:

## Callback function which is called for every new packet

def \_packet\_handler(param,header,pkt\_data):

# retrieve the position of the ip header

v\_pkt\_data = ctypes.cast(pkt\_data, ctypes.c\_void\_p)

v\_ip\_header = ctypes.c\_void\_p(v\_pkt\_data.value + 14)

pih = ctypes.cast(v\_ip\_header, ctypes.POINTER(ip\_header))

ih = pih.contents

print("{}.{}.{}.{} -> {}.{}.{}.{}".format(ih.saddr.byte1, ih.saddr.byte2, ih.saddr.byte3, ih.saddr.byte4, ih.daddr.byte1, ih.daddr.byte2, ih.daddr.byte3, ih.daddr.byte4))

Run the code and find the IP address connections for the first five connections?

## 4 Displaying connection details

Now we will read the TCP header, and which follows the IP address. In this case we will just display the TCP ports. First we add the format of the TCP packet (we have just used the first four fields):

class tcp\_header(BigEndianStructure):

\_fields\_ = [("source\_port", u\_short),

("destination\_port", u\_short),

("seq", u\_int),

("ack", u\_int)]

And we can replace the call back function:

def \_packet\_handler(param,header,pkt\_data):

# retrieve the position of the ip header

v\_pkt\_data = ctypes.cast(pkt\_data, ctypes.c\_void\_p)

v\_ip\_header = ctypes.c\_void\_p(v\_pkt\_data.value + 14)

pih = ctypes.cast(v\_ip\_header, ctypes.POINTER(ip\_header))

ih = pih.contents

ip\_len = (ih.ver\_ihl & 0xf) \* 4

th = ctypes.cast(ctypes.cast(pih, ctypes.c\_void\_p).value + ip\_len,

ctypes.POINTER(tcp\_header)).contents

print("{}.{}.{}.{}:{} -> {}.{}.{}.{}:{}".format(ih.saddr.byte1, ih.saddr.byte2, ih.saddr.byte3, ih.saddr.byte4, th.source\_port,ih.daddr.byte1, ih.daddr.byte2, ih.daddr.byte3, ih.daddr.byte4,th.destination\_port))

Run the code and find the IP address connections and TCP ports used for the first five packets?

## 5 Examining the Transport Layer protocol

The problem with the previous example is that there can be several transport layer protocols. So we must look at the Protocol field in the IP packet. Now modify your packet hander to add the IP Protocol field:

print("{}.{}.{}.{}:{} -> {}.{}.{}.{}:{} Protocol: {}".format(ih.saddr.byte1, ih.saddr.byte2, ih.saddr.byte3, ih.saddr.byte4, th.source\_port,ih.daddr.byte1, ih.daddr.byte2, ih.daddr.byte3, ih.daddr.byte4,th.destination\_port,ih.proto))

Now run the Python program, and generate some traffic (such as loading a Web page. You will now see other protocols, such as 6- TCP and 17 - UDP. List the protocols that you see:

Run the code and find the IP address connections and TCP ports used for the first five packets?

## 6 Filtering for TCP

Now we can filter for just TCP traffic by examining the IP Protocol field. For this just replace your packet handler with:

def \_packet\_handler(param,header,pkt\_data):

# retrieve the position of the ip header

v\_pkt\_data = ctypes.cast(pkt\_data, ctypes.c\_void\_p)

v\_ip\_header = ctypes.c\_void\_p(v\_pkt\_data.value + 14)

pih = ctypes.cast(v\_ip\_header, ctypes.POINTER(ip\_header))

ih = pih.contents

ip\_len = (ih.ver\_ihl & 0xf) \* 4

th = ctypes.cast(ctypes.cast(pih, ctypes.c\_void\_p).value + ip\_len,

ctypes.POINTER(tcp\_header)).contents

if (ih.proto==6):

print("TCP: {}.{}.{}.{}:{} -> {}.{}.{}.{}:{} Protocol: {}".format(ih.saddr.byte1, ih.saddr.byte2, ih.saddr.byte3, ih.saddr.byte4, th.source\_port,ih.daddr.byte1, ih.daddr.byte2, ih.daddr.byte3, ih.daddr.byte4,th.destination\_port))

Next generate some traffic by accessing a Web site (or refreshing the cache). Note the IP addresses and TCP ports of the Web connections:

If you go to https://Google.com, and run your script, which server port is used?

If you go to http://asecuritysite.com, and run your script, which server port is used?

Why do the two sites differ in the server ports?

## 7 Tutorial

1. Most of the IP packets are IP Version 4. Read the IP Version number from the first four bits with:

ip\_ver = (ih.ver\_ihl & 0xf0) >> 4

print "IP Version: ",ip\_ver

Run the Python program and capture some traffic. Which is the version number defined in the packets, and what does the "& 0xf0" and ">> 4" parts of the code do?

2. In the previous Python program, the TCP fields have not been fully defined. Figure 1 shows the Ethernet, IP and TCP fields. Using the TCP header definition, update the TCP class definition in your Python program to include all of the fields:

class tcp\_header(BigEndianStructure):

\_fields\_ = [("source\_port", u\_short),

("destination\_port", u\_short),

("seq", u\_int),

("ack", u\_int),

("flags", u\_short),

("window", u\_short),

("checksum", u\_short),

("urgent", u\_short),

("options", u\_int)]

Test the output. Do the SEQ and ACK tie-up on a connection?

2.1 What values do you get for the Flags field?

2.2 Now mask off the flags with:

print " Flags: ",th.flags & 0x0ff,

2.3 What values do you know get? Can you match them to the TCP flags?

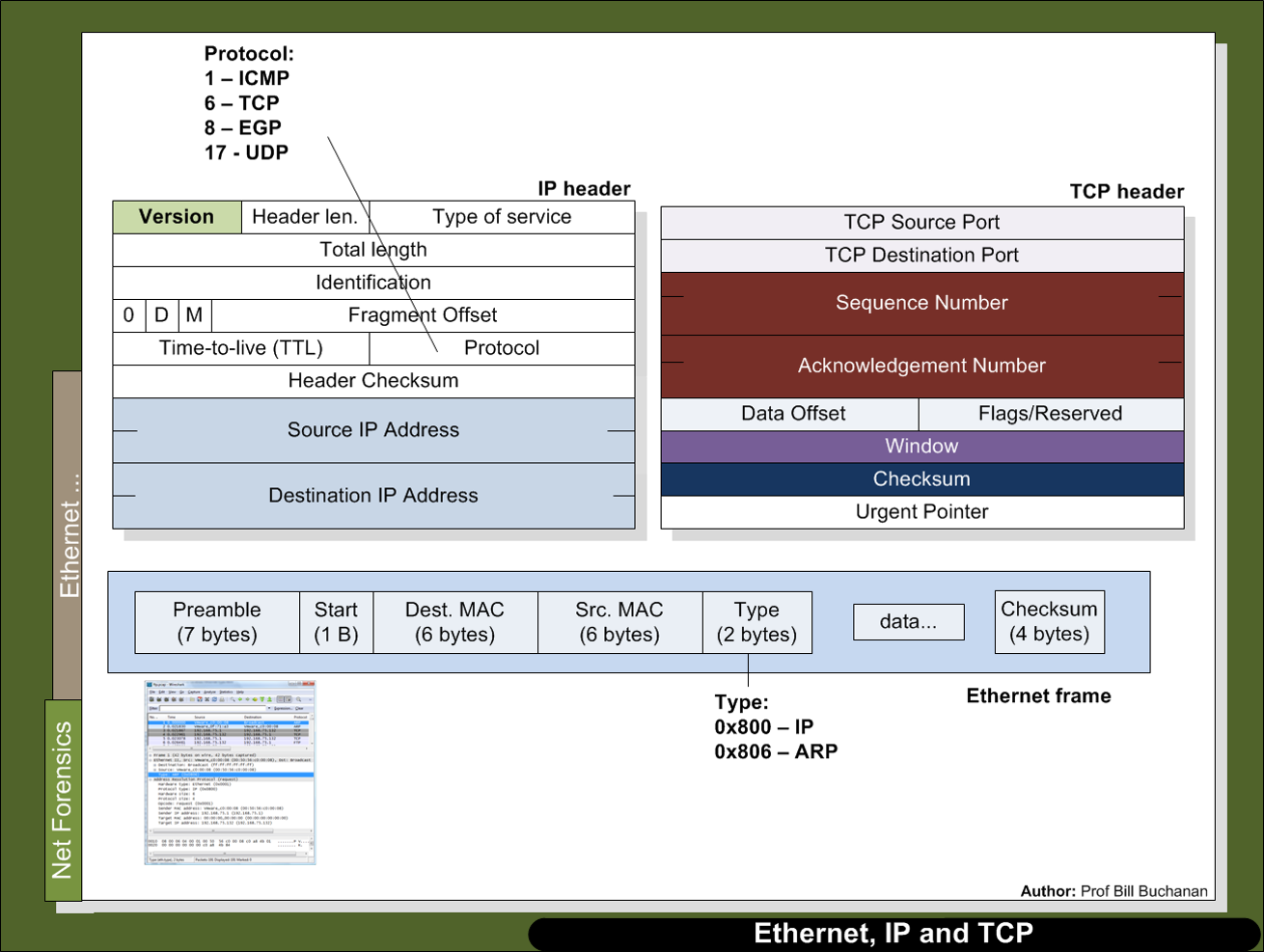


Figure 1: Ethernet, IP and TCP

3. We will now put the program into an infinite loop and break when there is a keypress. For this we use the Pywin32 library, which contains the pyHook class:

<http://sourceforge.net/projects/pywin32/files/pywin32/Build%20219/pywin32-219.win32-py2.7.exe/download>

Next replace:

## Get 20 packets

pcap\_loop(adhandle, 20, packet\_handler, None)

pcap\_close(adhandle)

with:

import pyHook,pythoncom

def OnKeyboardEvent(event):

exit()

pcap\_breakloop(ahandle)

pcap\_close(adhandle)

hm = pyHook.HookManager()

hm.KeyDown = OnKeyboardEvent

hm.HookKeyboard()

while True:

try:

while True:

pcap\_loop(adhandle, 1, packet\_handler, None)

pythoncom.PumpWaitingMessages()

except KeyboardInterrupt:

exit()

You should now be able to capture until a key is pressed.

## 8 Appendix

The files for Part 2 are:

* Part 1. [Here](https://dl.dropboxusercontent.com/u/40355863/dump01.py).
* Part 2. [Here](https://dl.dropboxusercontent.com/u/40355863/dump02.py).
* Part 3. [Here](https://dl.dropboxusercontent.com/u/40355863/dump03.py).
* Part 4. [Here](https://dl.dropboxusercontent.com/u/40355863/dump04.py).
* Final solution. [Here](https://dl.dropboxusercontent.com/u/40355863/dump07.py).

And you’ll need to download the following:

<https://winpcapy.googlecode.com/files/winpcapy.zip>