|  |
| --- |
| ci6240 – INTERNET SECURITY |
| DETERLAB PORTFOLIO |
| Daniel Carnovale |
|  |
| **K1336511** |
| **20/03/2017** |

|  |
| --- |
|  |

Table of Contents

[Experiment 1 - SynFlood 4](#_Toc477529383)

[Introduction 4](#_Toc477529384)

[Objective 4](#_Toc477529385)

[Experimental Method 4](#_Toc477529386)

[Testing 8](#_Toc477529387)

[Results and Discussion 9](#_Toc477529388)

[Example of Dump File Breakdowns (period before 30s mark, before attack) 11](#_Toc477529389)

[Tcpdumpfile\_cookies\_on 11](#_Toc477529390)

[Connection at 14:55:06 11](#_Toc477529391)

[Opening Handshake 11](#_Toc477529392)

[Data Connection 11](#_Toc477529393)

[Closing Handshake 11](#_Toc477529394)

[Duration (Last ACK segment – First SYN segment) 11](#_Toc477529395)

[Connection at 14:55:07 11](#_Toc477529396)

[Opening Handshake 11](#_Toc477529397)

[Data Connection 11](#_Toc477529398)

[Closing Handshake 11](#_Toc477529399)

[Duration (Last ACK segment – First SYN segment) 11](#_Toc477529400)

[Connection at 14:55:08 11](#_Toc477529401)

[Opening Handshake 11](#_Toc477529402)

[Data Connection 11](#_Toc477529403)

[Closing Handshake 12](#_Toc477529404)

[Duration (Last ACK segment – First SYN segment) 12](#_Toc477529405)

[Connection at 14:55:09 12](#_Toc477529406)

[Opening Handshake 12](#_Toc477529407)

[Data Connection 12](#_Toc477529408)

[Closing Handshake 12](#_Toc477529409)

[Duration (Last ACK segment – First SYN segment) 12](#_Toc477529410)

[Connection at 14:55:10 12](#_Toc477529411)

[Opening Handshake 12](#_Toc477529412)

[Data Connection 12](#_Toc477529413)

[Closing Handshake 12](#_Toc477529414)

[Duration (Last ACK segment – First SYN segment) 12](#_Toc477529415)

[Tcpdumpfile\_cookies\_off 13](#_Toc477529416)

[Connection at 14:48:14 13](#_Toc477529417)

[Opening Handshake 13](#_Toc477529418)

[Data Connection 13](#_Toc477529419)

[Closing Handshake 13](#_Toc477529420)

[Duration (Last ACK segment – First SYN segment, discounting initial establishment timestamps) 13](#_Toc477529421)

[Connection at 14:48:15 13](#_Toc477529422)

[Opening Handshake 13](#_Toc477529423)

[Data Connection 13](#_Toc477529424)

[Closing Handshake 13](#_Toc477529425)

[Duration (Last ACK segment – First SYN segment, discounting initial establishment timestamps) 13](#_Toc477529426)

[Connection at 14:48:16 13](#_Toc477529427)

[Opening Handshake 13](#_Toc477529428)

[Data Connection 14](#_Toc477529429)

[Closing Handshake 14](#_Toc477529430)

[Duration (Last ACK segment – First SYN segment, discounting initial establishment timestamps) 14](#_Toc477529431)

[Connection at 14:48:17 14](#_Toc477529432)

[Opening Handshake 14](#_Toc477529433)

[Data Connection 14](#_Toc477529434)

[Closing Handshake 14](#_Toc477529435)

[Duration (Last ACK segment – First SYN segment, discounting initial establishment timestamps) 14](#_Toc477529436)

[Connection at 14:48:18 14](#_Toc477529437)

[Opening Handshake 14](#_Toc477529438)

[Data Connection 14](#_Toc477529439)

[Closing Handshake 14](#_Toc477529440)

[Duration (Last ACK segment – First SYN segment, discounting initial establishment timestamps) 15](#_Toc477529441)

[Graphs from Breakdown 16](#_Toc477529442)

[TCPDump.exe Graphs for entire recording of traffic 17](#_Toc477529443)

[Conclusion 18](#_Toc477529444)

[Experiment 2 – SSL Apache 19](#_Toc477529445)

[Introduction 19](#_Toc477529446)

[Objective 19](#_Toc477529447)

[Experimental Method 19](#_Toc477529448)

[Appendix 29](#_Toc477529449)

Experiment 1 - SynFlood

## Introduction

A common internet service (TCP/IP) hosted by a server is sent several fake packets from an attacker as if they were being sent from the client, and in a way overloads/slows down/increases the traffic between the true client and server nodes. This results in a very low data transfer speed, to a point where the service is virtually unusable. The attack will not be as severe in this experiment, however it will still be evident that the traffic has been slowed down slightly during this period. This experiment provides an insight into both the results of and what is carried out behind the scenes of a “TCP SYN flood” or DoS server attack. This type of attack is surprisingly easy to implement and utilise as an attacker, so the experiment also helps demonstrate methods to defend against these sorts of attacks using cookies, or in this context “SYN cookies”.

## Objective

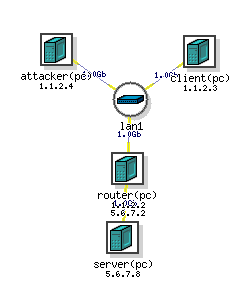
For this experiment we can pretend to be the server, client or attacker by connecting to these nodes in turn and performing operations such as installing the services on the server, starting a normal transfer of data from the client to the server using this server (generating traffic) or attacking the server after installing a separate service that makes use of this stream of data traffic and exploits the fact that several packets can be sent to overload it, given the stateless nature of the connection.

## Experimental Method

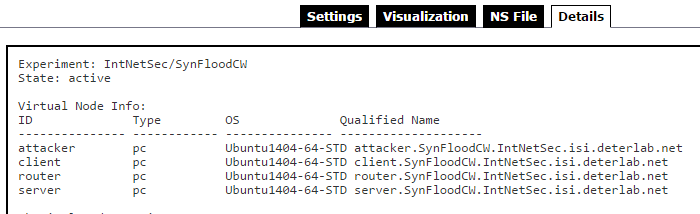
This first Experimental Method section includes comprehensive screenshots, to illustrate the typical tasks undertaken to create these experiments.

A unique “ns” file was uploaded to this new Deterlab experiment; construction parameters for the virtual environment including nodes, their network IP addresses, layout in relation to each other and so on.

Below is a topology visualisation of the four nodes (of which only three will be used for the experiment) in relation to the virtual LAN server which is shared between them.

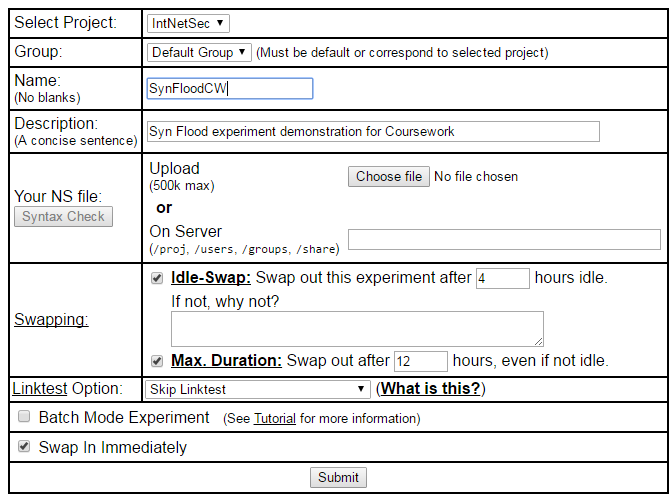


The IP addresses for each node can be found on the Details tab of the experiment’s management window, as shown below

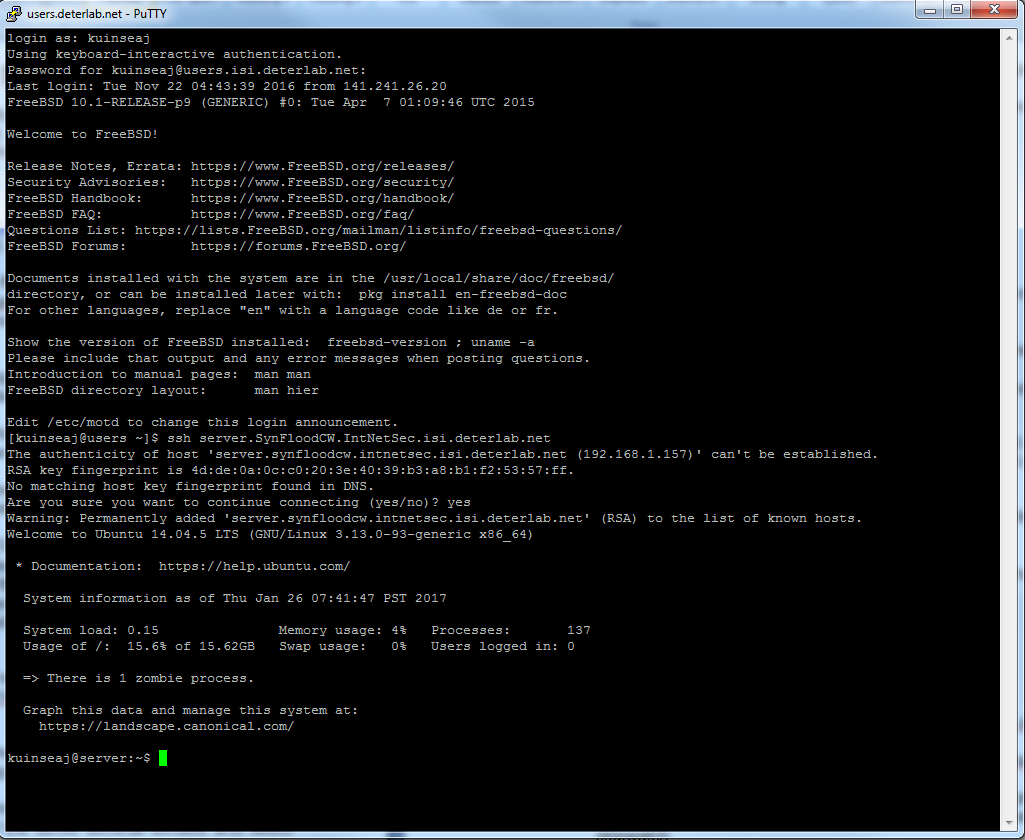


A summary of the steps performed accompanied with screenshots is as follows:

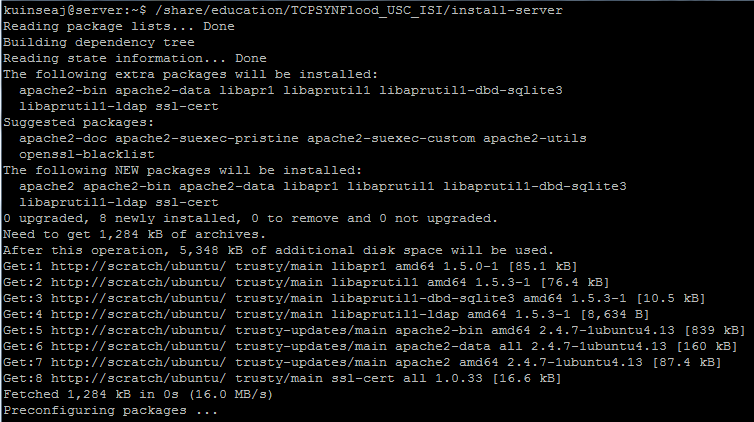
Swapped experiment in



Connected to users Deterlab address and SSH’edto server node (can also be saved as a PuTTy session for ease of access, if Experiment name is always kept the same)



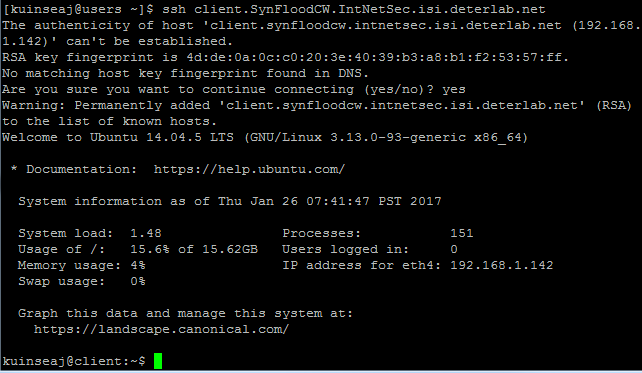
Installed Apache WebServer



Disabled SYN Cookies

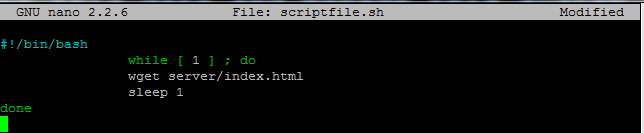


SSH’edto client node

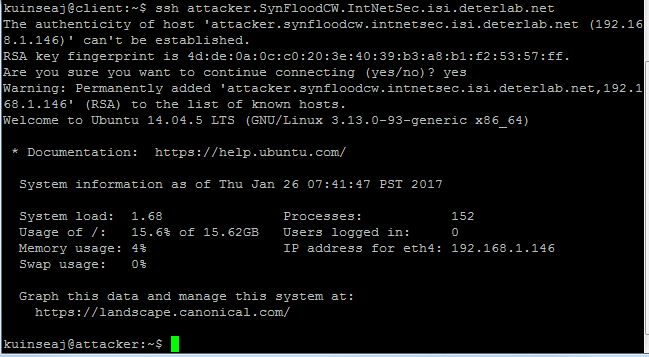


Used Nano editor to create a new script file, and edited to act as a “web page downloader”.

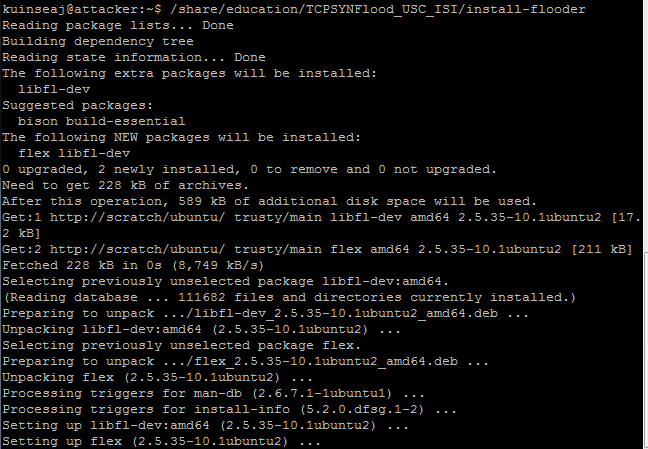




SSH’ed to attacker node

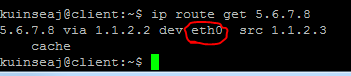


Installed and started Flooder





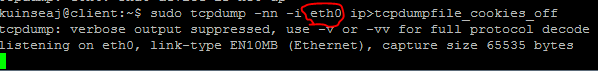
Discovered Ethernet port for Server access



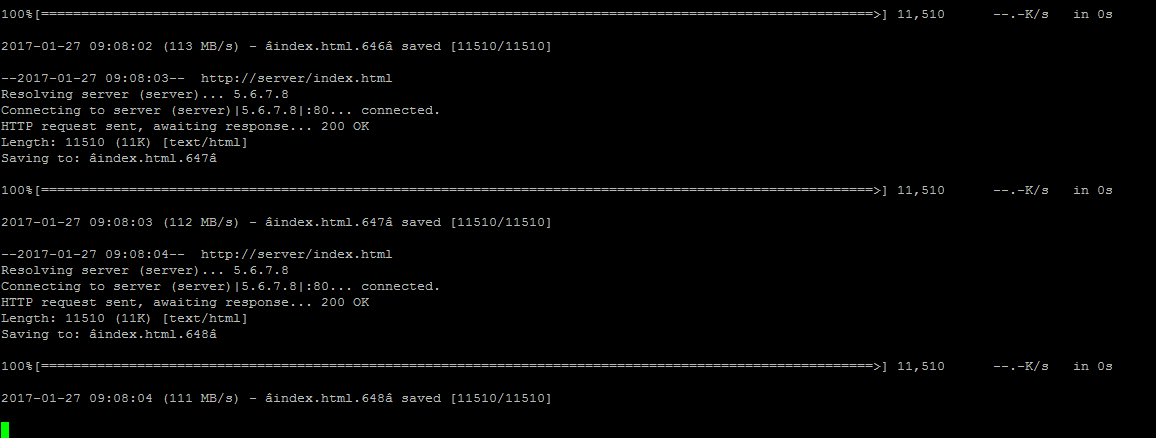
## Testing

Got stopwatch on mobile phone ready

Recorded traffic on one client terminal (stats are being recorded to file “tcpdumpfile\_cookies\_off”)



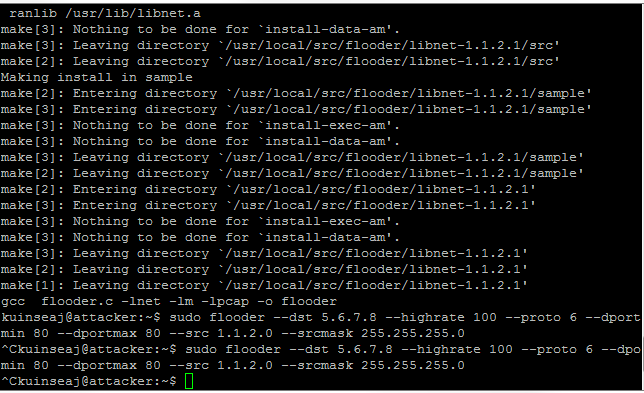
Started downloading web pages (flooder) on another client terminal, started timing



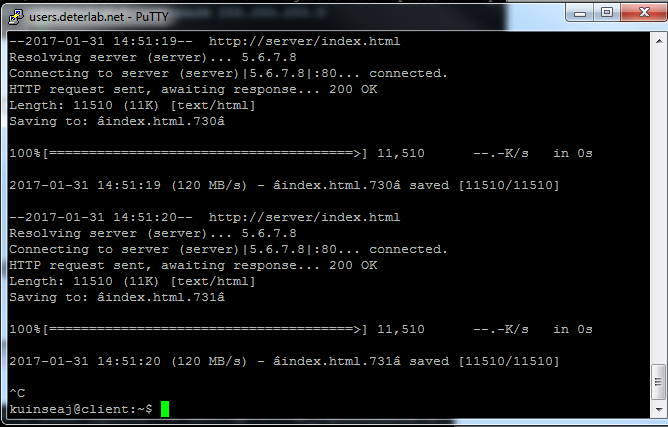
Started attack after 30 seconds



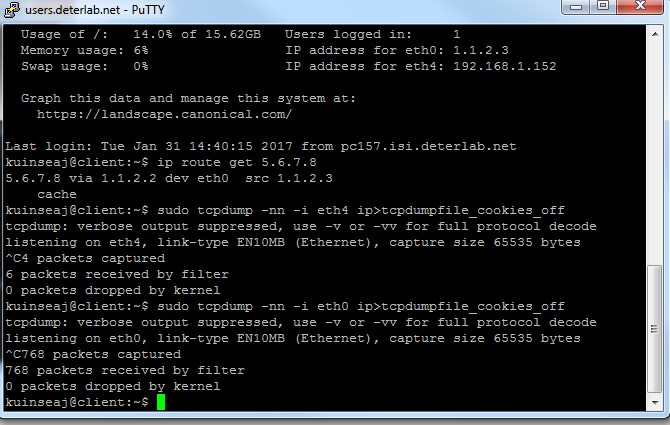
After 120 seconds stopped the attack



After 30 seconds stopped the legitimate traffic



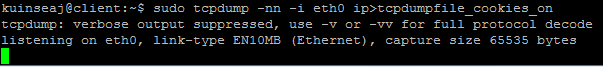
At same time Stopped the TCP dump



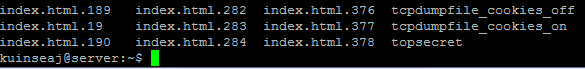
Enabledsyn cookies



Repeated using the “tcpdumpfile\_cookies\_on” file



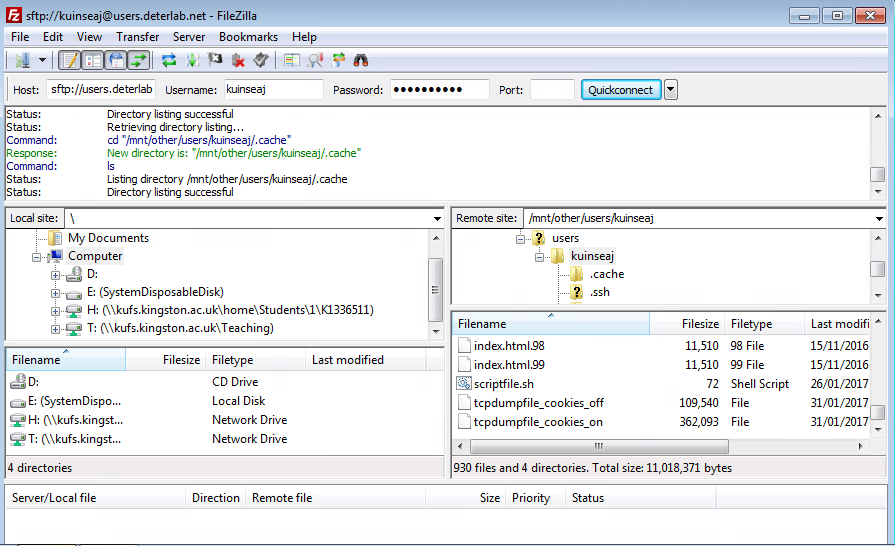
Verified that two dump files were created



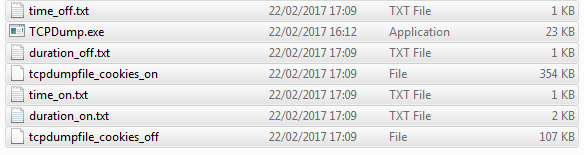
# Results and Discussion

Analysing these dump files provided an insight into the connections taking place at the time and how long each connection lasted. The three way handshake was broken up into three sections, and identified by the notations S, P and F for Opening Handshake, Data and Closing Handshake connections respectively. The period symbol signifies an acknowledgement or closing period for each section, hence the full handshake always starts and ends with a connection from the client to the server (in the case of the first connection below for Cookies enabled at timestamp 14:55:06, this is from the client IP of 11.2.3:55367 to the server IP of 5.6.7.8:80).

An FTP connection was made to the virtual environment using FileZilla



This allowed the dump files to be downloaded for analysis to the H:\ drive. Initially I manually started analysing the dump files in a text editor. A tool TCPDump.exe was provided for us to extract solely the time and duration from these connections, though their details including the timestamp could only be found in the dump files themselves. The tool did however make it easier to understand how the attack affected the connections at different times represented in seconds instead of their timestamps.



## Example of Dump File Breakdowns (period before 30s mark, before attack)

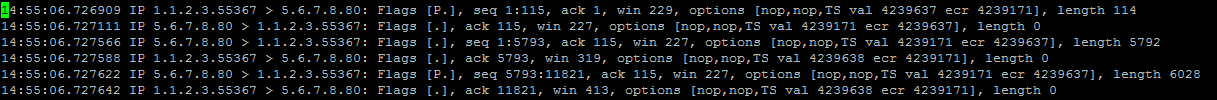
## Tcpdumpfile\_cookies\_on

## Connection at 14:55:06

### Opening Handshake



### Data Connection



### Closing Handshake



### Duration (Last ACK segment – First SYN segment)

= 6.734931 – 6.726487

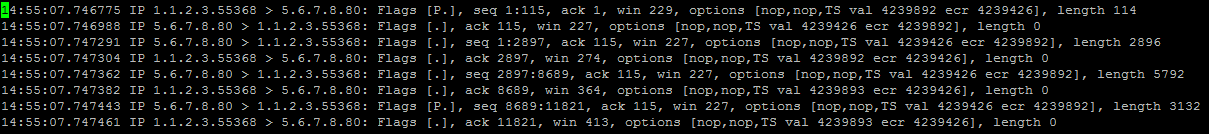
= 0.0084 (2 DP)

## Connection at 14:55:07

### Opening Handshake



### Data Connection



### Closing Handshake



### Duration (Last ACK segment – First SYN segment)

= 7.753650 – 7.746403

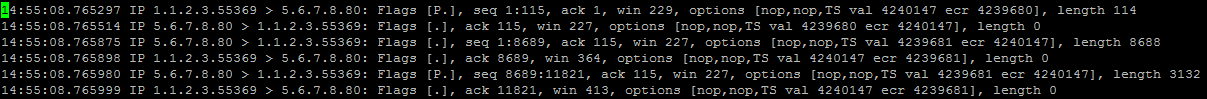
= 0.0072 (2 DP)

## Connection at 14:55:08

### Opening Handshake



### Data Connection



### Closing Handshake



### Duration (Last ACK segment – First SYN segment)

= 8.771981 – 8.764901

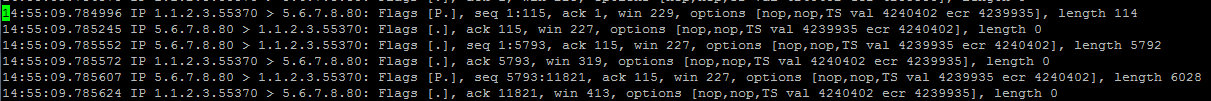
= 0.0071 (2 DP)

## Connection at 14:55:09

### Opening Handshake



### Data Connection



### Closing Handshake



### Duration (Last ACK segment – First SYN segment)

= 9.791909 – 9.784671

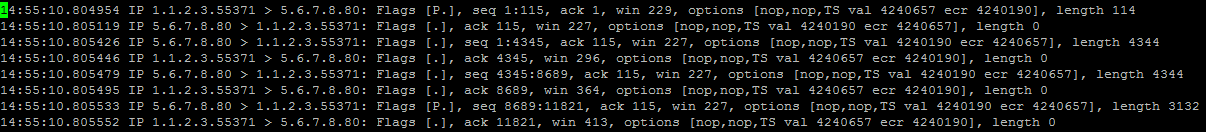
= 0.0072 (2 DP)

## Connection at 14:55:10

### Opening Handshake



### Data Connection



### Closing Handshake



### Duration (Last ACK segment – First SYN segment)

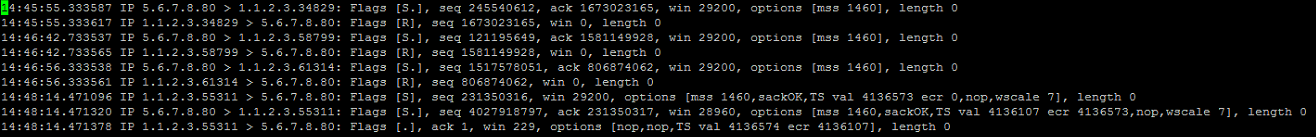
= 10.811530 – 10.804577

= 0.0070 (2 DP)

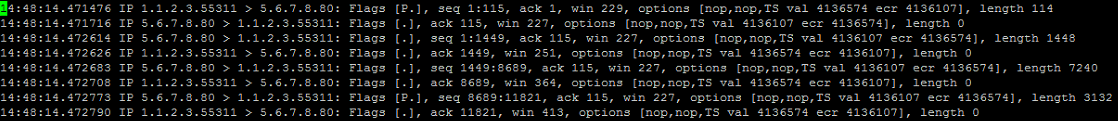
## Tcpdumpfile\_cookies\_off

## Connection at 14:48:14

### Opening Handshake



### Data Connection



### Closing Handshake



### Duration (Last ACK segment – First SYN segment, discounting initial establishment timestamps)

= 14.478489 –14.471096

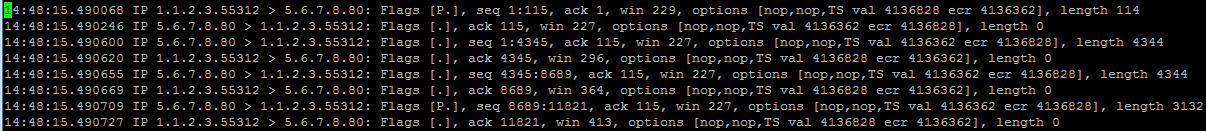
= 0.0074 (2 DP)

## Connection at 14:48:15

### Opening Handshake



### Data Connection



### Closing Handshake



### Duration (Last ACK segment – First SYN segment, discounting initial establishment timestamps)

= 15.496718 – 15.489728

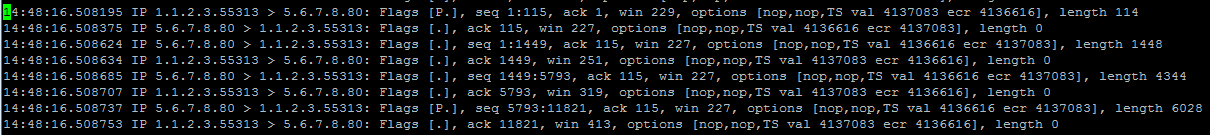
= 0.0070 (2 DP)

## Connection at 14:48:16

### Opening Handshake



### Data Connection



### Closing Handshake



### Duration (Last ACK segment – First SYN segment, discounting initial establishment timestamps)

= 16.514788 – 16.507905

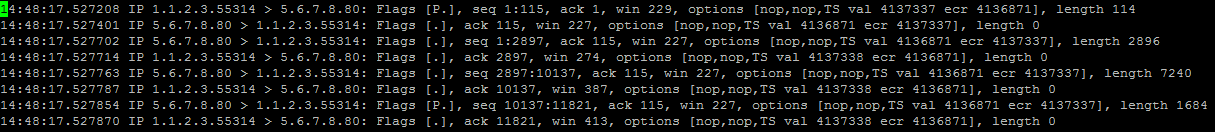
= 0.0069 (2 DP)

## Connection at 14:48:17

### Opening Handshake



### Data Connection



### Closing Handshake



### Duration (Last ACK segment – First SYN segment, discounting initial establishment timestamps)

= 17.533808 – 17.526880

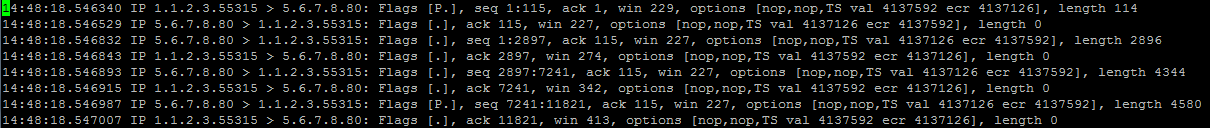
= .0070 (2 DP)

## Connection at 14:48:18

### Opening Handshake



### Data Connection



### Closing Handshake



### Duration (Last ACK segment – First SYN segment, discounting initial establishment timestamps)

= 18.552743 – 18.545914

= .0068 (2 DP)

## Graphs from Breakdown

The graphs above show the duration of a connection and packets that were being sent between the client and server node without the attacker node intervening. The initial connection is an initial communication period between the nodes and seems to be slightly longer than the following connections, most likely due to the initial handshake or verification of the unique client node’s ip address.

As indicated in the three Duration vs Timestamp graphs, the time it takes to send the first packet (from above results) is about 0.001 seconds longer with Cookies Enabled than it is for Cookies Disabled. This is due to the fact that enforcing security against the DOS attacks requires a short amount of time to be put in place to handle the flooding program from the attacker node.

## TCPDump.exe Graphs for entire recording of traffic

## Conclusion

Experiment 2 – SSL Apache

## Introduction

An SSL Certificate is installed on a web server to allow secure connections between the webserver and a browser. Its most common usages involve credit card transactions and the like, and is rapidly becoming the common way to browse social media websites. These certificates link a domain name, server name, organisational identity and its location together as one binding.[1]. Only the main steps will be shown here, as most steps are more associated with initial setup procedures rather than presentation of the experiment itself.

## Objective

This experiment aimed to achieve the installation of such a certificate on apache, on a virtual node within the virtual Deterlab web server. After the installation of Apache and the secure certificate, a web browser was used to attempt to gain access to the server and mainly identify whether it was truly secure. If so, attempts to proceed to the web page and communicate with it securely would be performed.

## Experimental Method

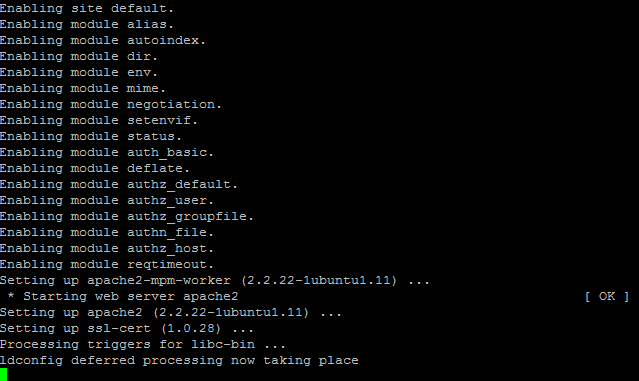
The experiment was instantiated through a Network Simulator (NS) file within the Deterlab environment, similarly to the first experiment. Below is its topology visualisation (only one virtual node named intro(pc) is present and required for this experiment).



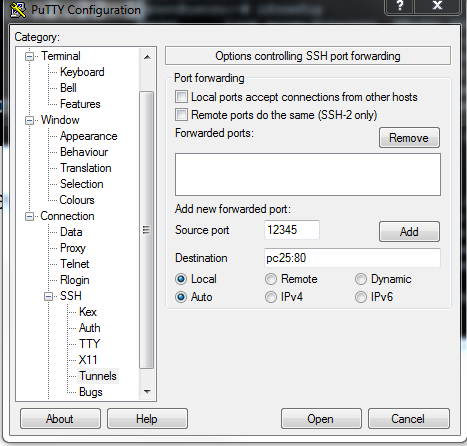
After the typical connection procedures to the server and virtual node and modification of the sources file (to specify where the package should be installed to) were performed, the apache package was installed. The installation of the apache package was performed with the command

“sudo apt-get install apache2”

A snippet of the installation progress is shown below.

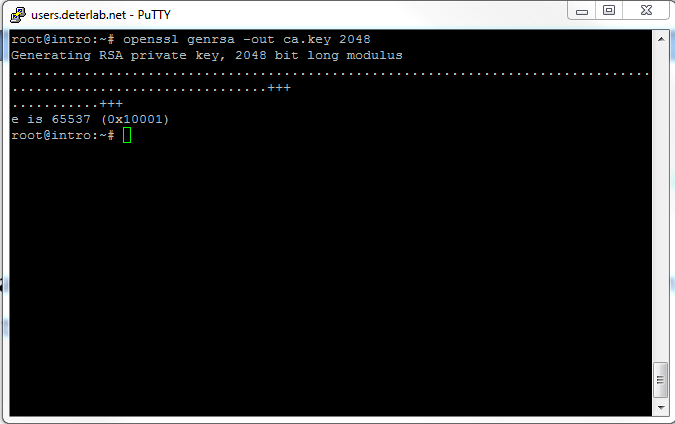


After SSL is enabled, a new PUTTY connection was made to check the status of the Apache web server. This SSH connection is to users.isi.deterlab.net, though specific options are set in the Tunnels category. The node name is displayed in the details section on Deterlab’s experimentation page.

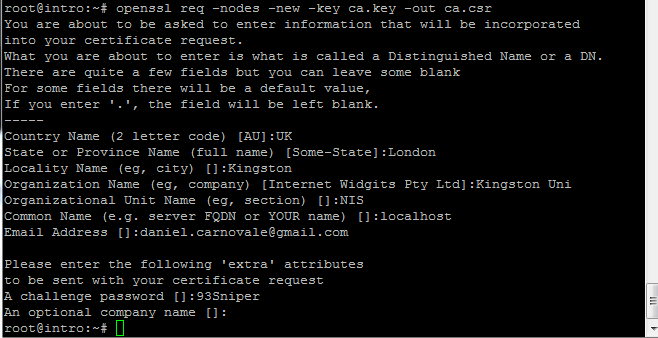


It was then time to create the Self-Signed SSL Certificate on Apache. An open source project named OpenSSL was to be used for this.

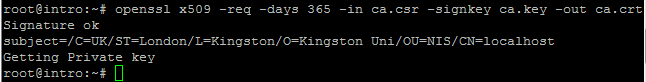
The package was installed (similar to the Apache package installation), and keys for the Certificate Signing Request (CSR) were generated.



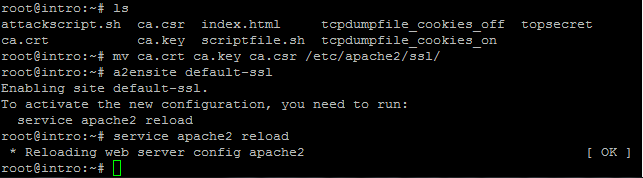
A file to be used to request the certificate was then created.



A self-signed certificate was created.

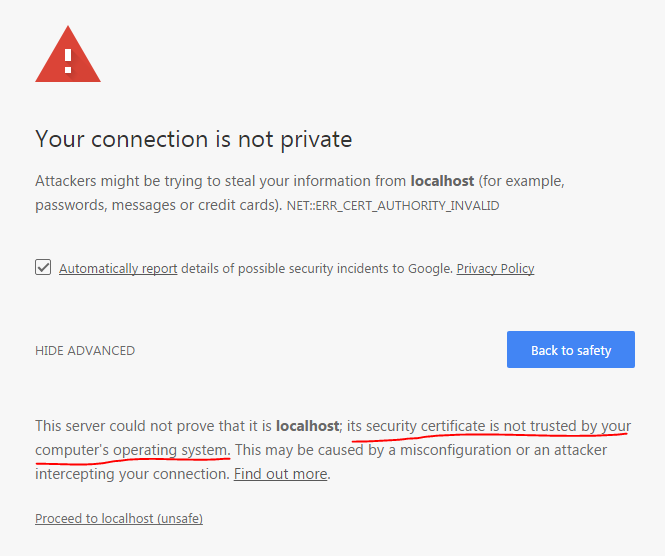


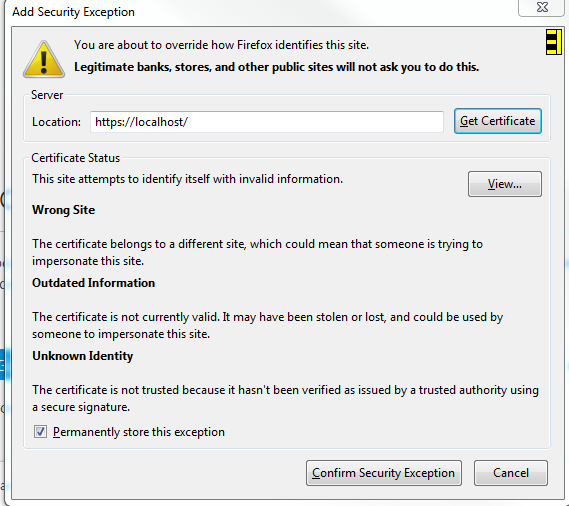
After moving these three certificate files into a new folder named “ssl”, ssl was enabled and apache was reloaded.

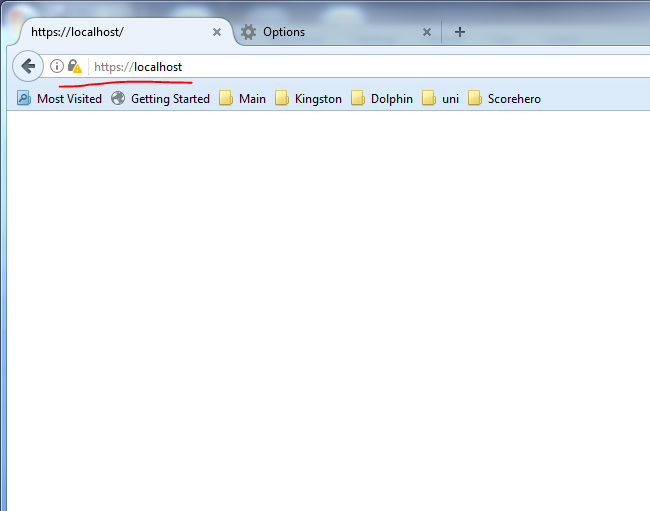


The SSL configuration file for apache was edited to allow connections from localhost, using the ca.crt and ca.key files. Localhost must now be port forwarded to 443, so that these certificate files can come into play.

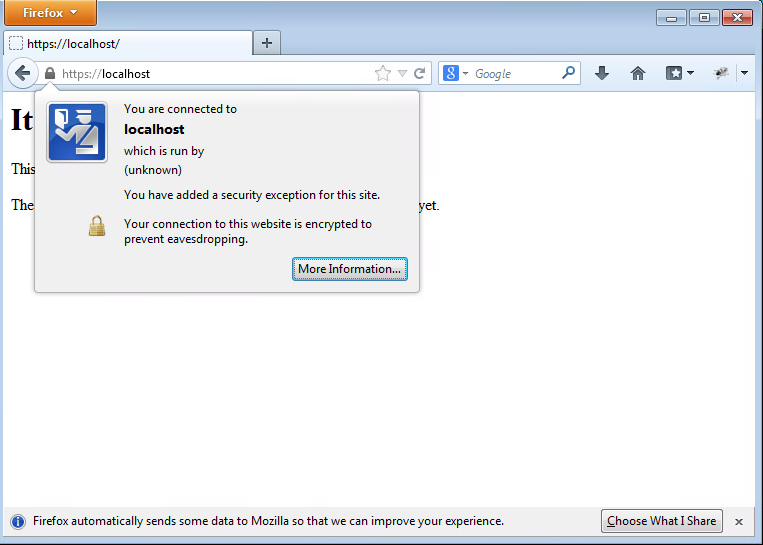
After connecting to localhost:443 this message should be displayed, and it is in fact encouraging. As highlighted here it claims to have found a certificate on localhost, though it is simply not trusted as an exception needs to be added.

****

****

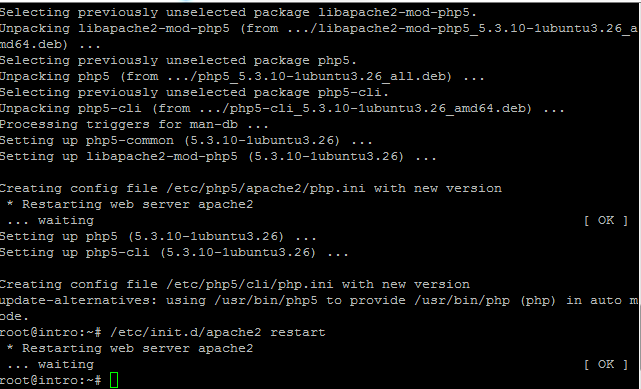


The padlock here is a standard grey (no problems with the certificate this time). The content underneath can barely be seen under the security padlock’s message I had clicked on. If needed, extra information can be viewed through clicking the “More Information...” button, though I will proceed with the main experiment.

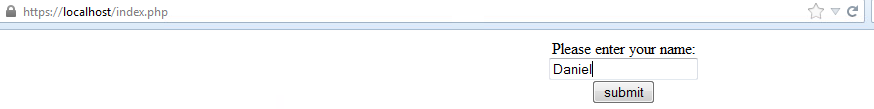


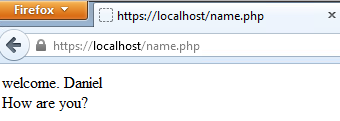
Cross-Site scripting is an attack using malicious scripts, being executed on trusted web sites. Measures were taken in this experiment to demonstrate how to help protect the server from such attacks. The first step is to install the PHP language package, achieved through the command

“apt-get install php5”, after which apache must be restarted through “/etc/init.d/apache2 restart”

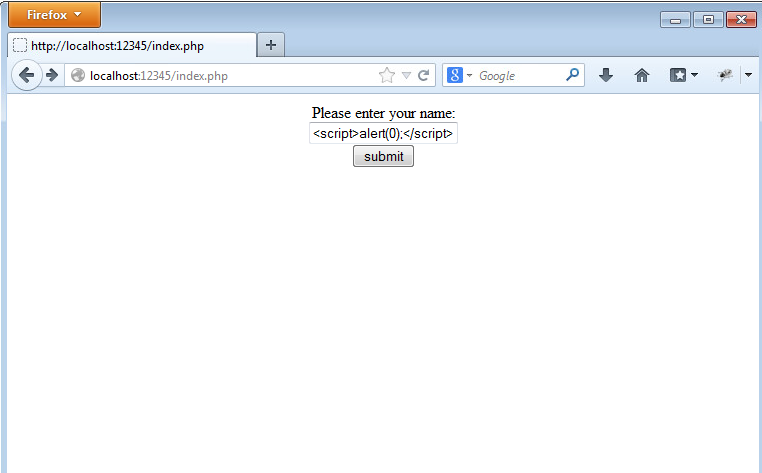


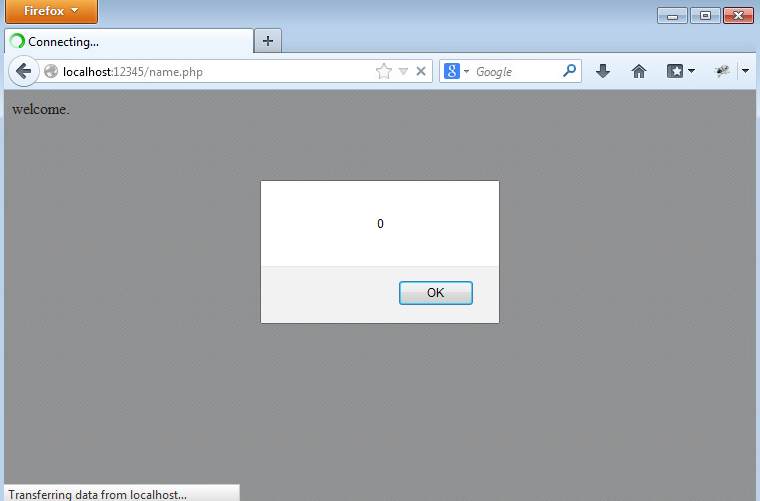
The experiment requires two files to be created, named index.php and name.php. Their creation will be absent from this report, as their implementation is the focus here. After their creation, the direct URL can be entered and the user can enter their name into the form and have it reported back to them, as it is being posted to name.php .





To demonstrate vulnerability here (XSS attack) a script statement was inserted into the name field, resulting in its execution after submission. An “onmouseover” statement can also be used (more significant demonstration).

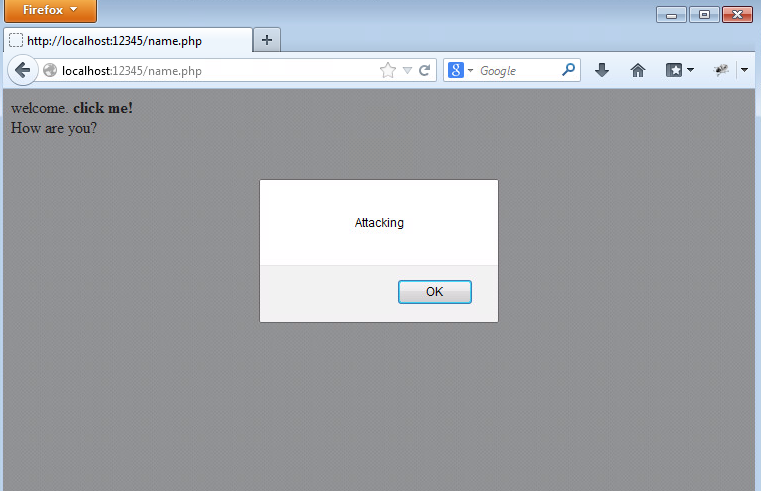




(using

**<b onmouseover=alert('Attacking')>click me!</b>**

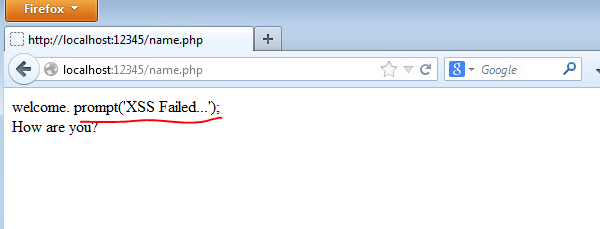
) :



The following screenshots illustrate how sanitisation can be used, meaning the data is modified in that unwanted bits are extracted. The name.php file must be edited once more to the following, to now include the strip\_tags() function:



All scripts the user intends to execute in the field would from this point be converted to pure text. The screenshot below shows the pure text of the attempt to show a prompt as shown in a previous screenshot, though this time it has been output instead on the page with the <script> tags eliminated.



Experiment 3 – Man In The Middle

## Introduction

The Address Resolution Protocol (ARP) is a set of instructions implemented by the Internet Protocol (IP). Its purpose is to map IP network addresses to the hardware addressed used, achieved through a data link protocol. In other words, it helps users find the address of a computer in a network. The client sends a request for the ip of the server, and this protocol (after identifying the source and destination information) reports back the correct IP address for the server resolved from its hardware address; the physical address of the Network Interface Card (NIC).

The protocol maintains a list of the ip-to-ethernet address mappings after each operation, for which a command also named arp makes use of. This is related to the attack, as the protocol can be manually triggered to activate and use one of the mappings in the list.

## Objective

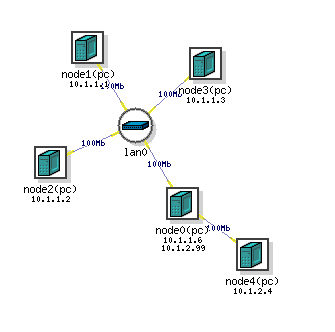
This experiment demonstrates an attack that utilises this Protocol (ARP), however instead of the client receiving the address resolution of its target server, its request is redirected to retrieve that of an attacker’s machine. The attack like many other network attacks simply makes use of the protocol, rather than modifying it. This protocol instructs computers on how to reach another, meaning this can be exploited into instructing a computer to reach an attacker’s machine instead of the legitimate destination machine. Hence this results in the client being deceived through sending information to the wrong person, as the attacker is intercepting all information that should be sent to the original destination server. This makes him in effect the “Man in the Middle”.

## Experimental Method

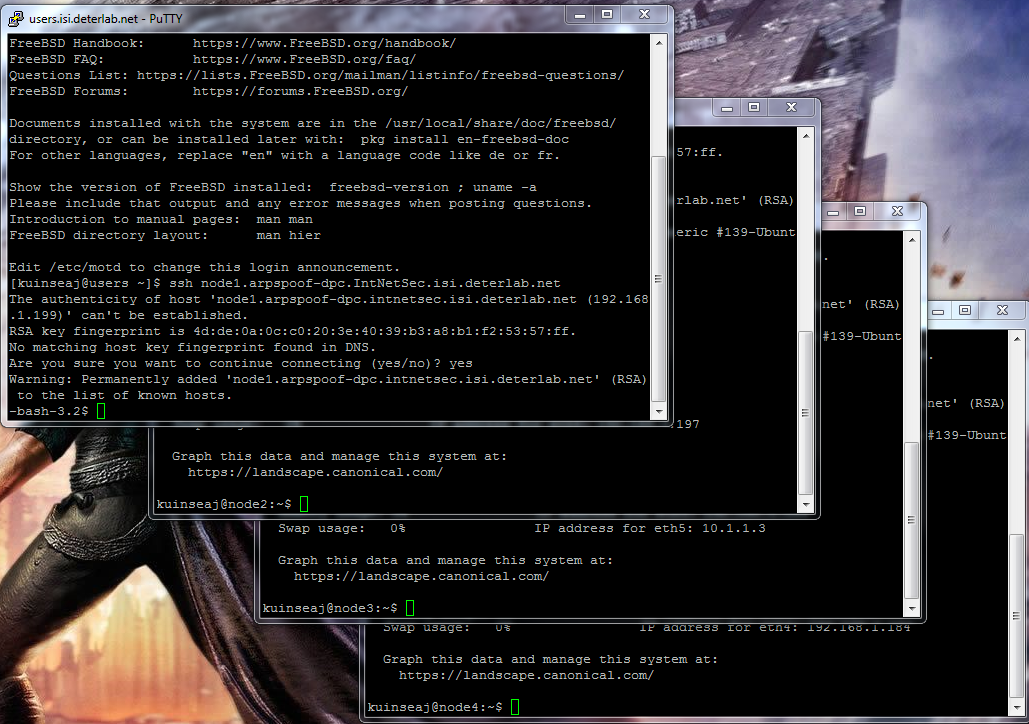
As always this experiment was instantiated through a Network Simulator (NS) file within the Deterlab environment, although this time the file resides on the server and cannot be manually downloaded and uploaded like the previous two experiments. The experiment was created through specifying this path for the ns file:

/share/education/MITMARPPoisoning\_USC/arpspoof.ns

Below is its topology visualisation (Four nodes connected separately around the router, and a fifth node connected to one of the four).



The initial step was to connect to each of the four nodes (excluding node0). The standard ssh command was used on a PUTTY terminal, and the terminal was duplicated 3 times resulting in each window having a connection to a separate node.

****

The nodes represent the following:

Node 0 – Victim, Router

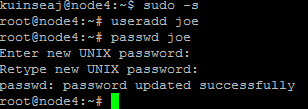
Node 1 – Attacker

Node 2 – Victim, Web Server

Node 3 – Victim, Web Client

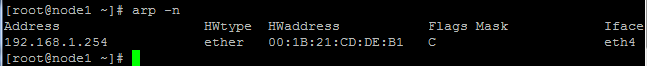
Node 4 – Web Server, FTP Server

On node 4, a generic username and an associated password were created (password is ‘superpw’):

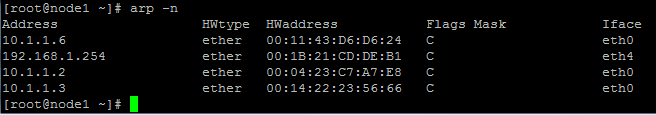


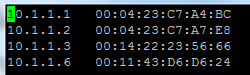
On node1, the ARP table was manipulated. It was viewed, added to and deleted from as a test of its functionality.

The list of the ARP table:

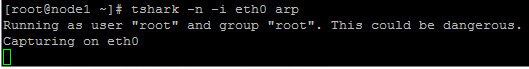


This table was then populated through pinging the other machines (nodes in the case of this experiment). Their IP addresses were visible through the visualisation tab on the Deterlab experiment interface page. These were pinged one by one resulting in new entries in the ARP table:

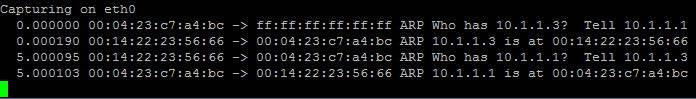
A file was created with these mappings, and looks like this:



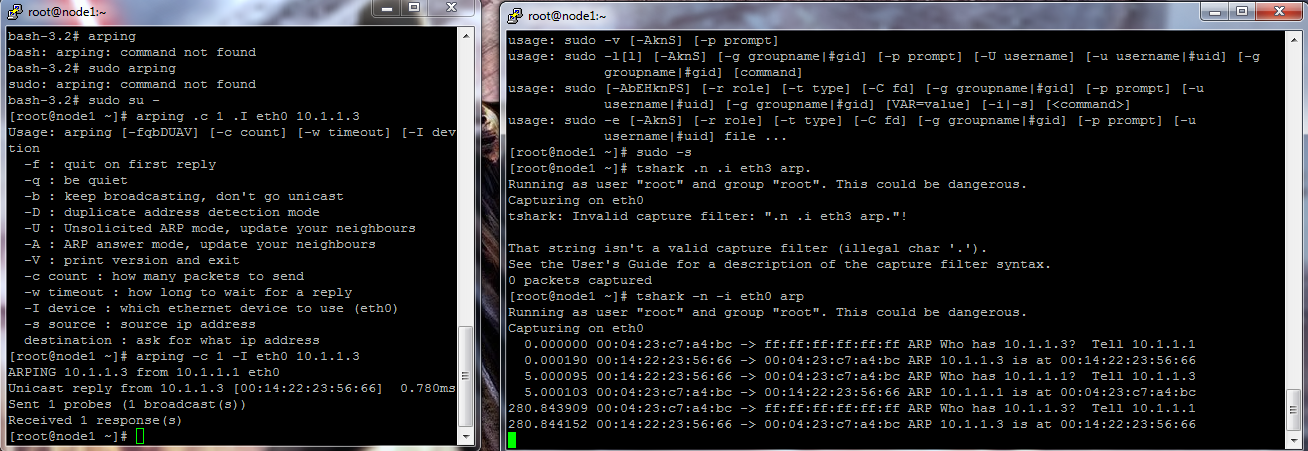
The connection to node1 was duplicated, and the Tshark program was run which captured all ARP traffic on the local interface.

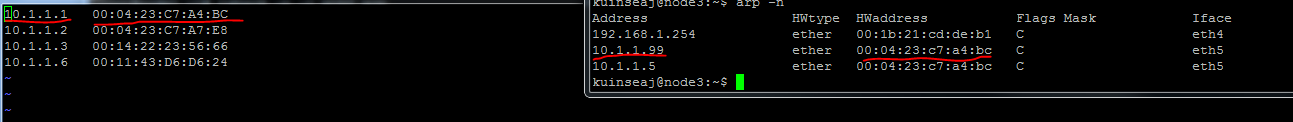


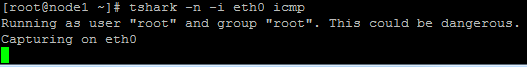
ARP activity was initiated through the original node1 window, through a ping of one of the machine addresses on eth0.



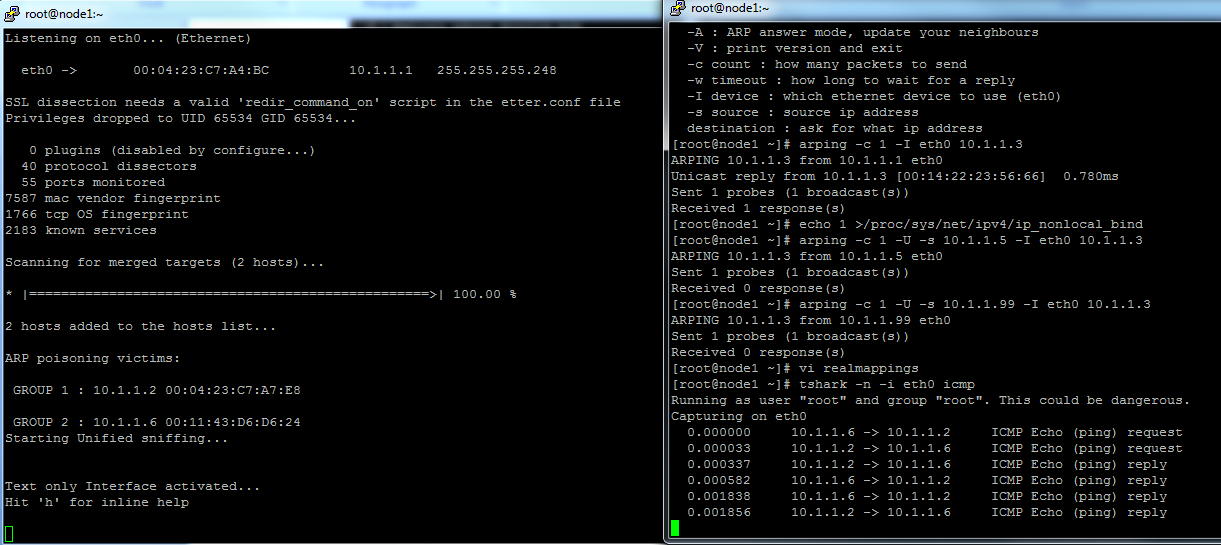
A special ARPING command was used to dump more information into the window

 Back on the original node1 terminal, source IP address spoofing was experimented with. Two addresses were used: one that does not exist on the subnet (10.1.1.99) and another one that does exist (10.1.1.3). The arp table of node3 was compared to the realmappings file, note how the fake address 10.1.1.99 is mapped to the MAC address of 10.1.1.1 (Node 1, 00:04:23:C7:A4:BC). Ignore the 10.1.1.5 entry, I abandoned that and went with 10.1.1.99 to easily distinguish it from the rest.

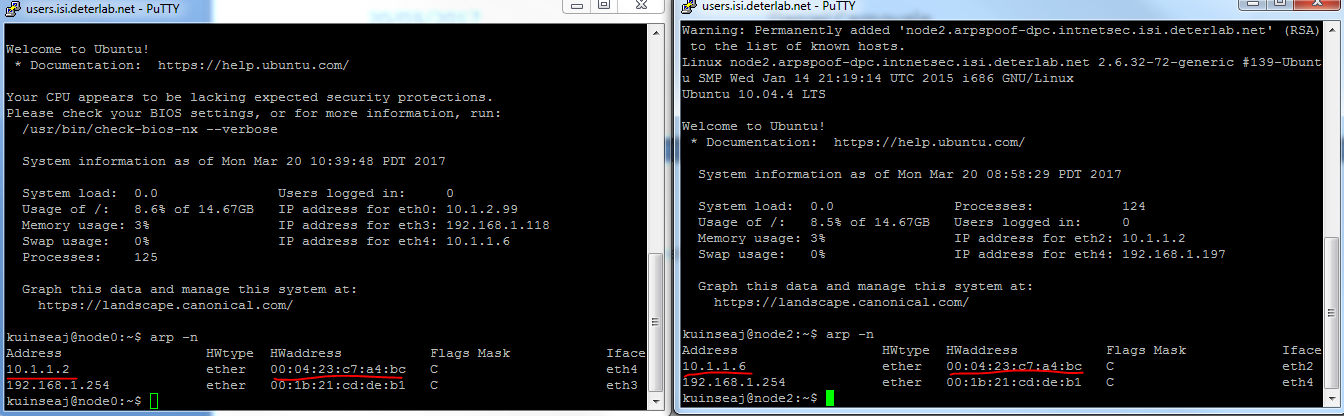
The first node can now act as the “Man in the Middle” between node 2 and node 0. The tool “ettercap” was used, an automatic version of what was manually done in the previous step. Firstly, the tshark service was restarted with different parameters, to monitor icmp traffic instead:



Then on node1 the ettercap command was used, populating the Tshark dump window.



The arp tables between node0 and node2 were compared, highlighting how node1 has become the “man in the middle”.



## Appendix