Basic Tor Network Using VMWare for Security Research

Ruhama Mohammed Zain
CyberSecurity Malaysia
Seri Kembangan, Malaysia
Email: ruhama [AT] cybersecurity.my

Abstract—**This paper describes how to create separate virtual machines for each Tor relay, directory server and client using VMware Fusion. This approach offers a flexible and realistic research experience in terms of installing, configuring and updating each Tor node. The result was a working Tor network that accurately simulated each node's operating system resource requirement (memory, disk space and CPU resources) and utilisation. Another important aspect to the experimental setup is the ability to interact with each Tor node easily via graphical user interface.**

Keywords-component; Tor network; experimental testbed; virtualization; VMware, onion routers, relays

I. INTRODUCTION

Users have many different motivations for their desire to be anonymous on the Internet. There are safety and security apart from privacy reasons. Using Tor to stay anonymous may be the only option for political dissidents, social activists and freedom fighters around the world. Privacy and security researchers are interested in anonymity networks such as Tor because of the type of personal information at risk. Researchers seek experimental setups with enough fidelity to reproduce the character and response of a real Tor network. This is to avoid conducting privacy experiments on the live Tor network and putting actual Tor users in jeopardy. We describe how to create Tor relays, directory server and client using discrete VMware Fusion virtual machines. We propose that this method provides a customizable and realistic experience for the researcher by requiring them to install, configure and update the Tor nodes just like any other operating system. The end result is a functioning Tor network that reflects each node's resource requirement in terms of CPU, memory and disk space that can easily be interacted with via graphical user interface.

Security and privacy researchers have been interested in the Internet and privacy issues that it brings[1][2][3][4]. The subject of anonymity is a closely linked topic and is covered by [5][6][7][8]. The issues discussed range from the privacy of Facebook pictures [9], the need for anonymity when reading and reporting news inside oppressive regimes [10][11], and suspicions about government monitoring and recording of online activity [12]. Justifications like terrorism prevention have been used but privacy advocates are very concerned that less legitimate motives may threaten the lives and safety of innocent Internet users unless they remain untraceable and anonymous.

Tor (The onion routing) is a tool that can help provide the necessary anonymity and plausible deniability to users who want them. The Tor browser allows anybody with little technical expertise to send and receive information anonymously through the Internet. It is easy to determine that someone is using Tor because it is not a covert means to use the Internet. However, Tor makes it difficult to track and identify users who are accessing specific information or services on the Internet. In other words, Tor can help safeguard both privacy and anonymity of its users.

Tor was introduced by researchers at U.S. Naval Research Laboratory [13] in 2004 as a privacy tool for anonymous communication by users of the Internet. Tor is operated by volunteers distributed all over the Internet. User traffic is directed through three independent network relays called Onion Routers. The three relays make up a Tor circuit. Only a special type of Onion Router (the exit relay) is allowed to carry the exit traffic to the final destination. To ensure anonymity, each relay only knows the address of the previous and the next relay. As a result, it is not trivial to determine the originator and destination of a network traffic routed through Tor. In addition to that, encryption is also used at each relay where a different key is used for every communication between client to relay, relay to relay and relay to destination.

The motivation for this paper stems from a requirement by security researchers to create a Tor network that is isolated but realistic enough to perform experiments to improve understanding about threats and their countermeasures. A few methods have already been used to simulate or emulate the live Tor network. This paper presents another option that may be useful for certain class of experiments.

Virtualization technology has enabled researchers to build experimental networks using commodity hardware and software easily. This allows the researcher to perform network experiments without jeopardizing the performance and security of a live production network. Virtualization has eliminated the need to maintain separate physical machines with the associated cost savings. It also allows the setup to be easily replicated and transported for teaching or demonstration.

According to [14], there are many options to conduct Tor network research. Direct experiments on the Tor network, formal modelling, simulation and emulation of the Tor network are some of the examples mentioned. It should be mentioned that there are ethical issues associated with conducting experiments on the live Tor network. The potential issues include harming the security of real Tor users and the performance of the Tor network itself. Thus, security researchers are always looking for safer alternatives to run Tor experiments.

In this paper we used VMware virtualization platform to generate discrete virtual machines that play the role of directory server, relays, client and web server. The set of virtual machines are set up as a small-scale but functioning Tor network. It can be used to perform Tor experiments or to teach about how Tor network functions in general.

II. TYPES OF VIRTUALIZATIONS

There are three broad categories of virtualization solutions.

Software virtual machines or Type 2 Hypervisor manage both the host operating system and guest operating system [15]. VMware Workstation, VMware Fusion, Oracle Virtualbox and Microsoft Virtual Server are examples of this category. This is shown in Figure 1. There is a host operating between the host hardware and the hypervisor.

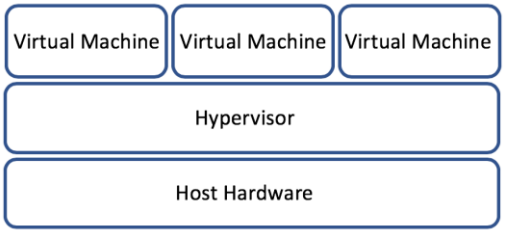

Figure 1: Software virtual machine

Hardware virtual machines, also called Type 1 Hypervisor, have the virtualization technology reside directly on the host hardware. The Hypervisor is directly above the host hardware as shown in Figure 2. This baremetal technology enables the hypervisor, modified code, or APIs to facilitate faster transactions with hardware devices [15].

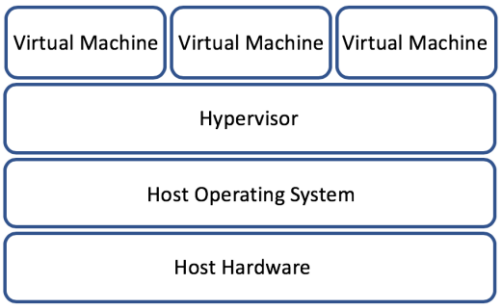

Figure 2: Hardware virtual machine

Virtual OS/containers, partitions the host operating system into containers or zones [15]. Docker, Netkit, BSD jail and chroot environment belong in this category. As Figure 3 shows, there is no hypervisor in this setup.

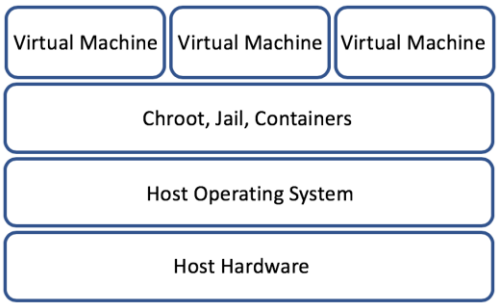

Figure 3: Virtual OS/Containers

III. BENEFITS OF VIRTUALIZATION

Virtualization technology can be used to save cost by eliminating the need to have many physical machines. In addition to just buying one or two physical hardware there will be less overall power consumption and space required to house the machines. This is beneficial in scenarios such as server consolidation and creation of virtual test environments [16]. It is not surprising that cloud service providers like Google Compute Engine and Amazon EC2 use virtualization to provide their services [17]. Modern central processing units (CPUs) can be utilized to their fullest by having them serve multiple virtual machines. Likewise, the random access memory (RAM), disk storage and input/output (I/O) devices can be utilized fully if they are shared among many virtual machines [15]. As an added benefit for the Tor experimenter, having a virtualized Tor network within a single powerful physical machine makes it easy to start and stop relays in order to make changes or record an observation. It is easy to make backups of virtual machines and create "snapshots" of the disk state which give ability to the experimenter to restore the setup to a previous desired state very quickly. All these are much more tedious to accomplish with physical machines.

IV. Ethical Issues with Using the Live Tor Network for Collecting Data

As mentioned in the Introduction, there are ethical issues if activities such as usage data is collected from the live Tor network [18]. Some of the issues are described next.

Legal requirements that disallows user data collection. This is in regards to existing wiretapping and data protection laws in many countries that prohibit unauthorized user data collection. In the Tor network the problem can be further amplified because of the need to collect data at many points in the Tor network in order to give a full view of the end-to-end Tor traffic. It is easy to see that the problem is even more complicated if the Tor relays are located in different countries with different sets of law. Maintaining user anonymity remains another ethical issue even if all the laws are complied with.

User privacy that must be maintained and protected. The data collected from the live Tor network must not fall into the wrong hands and be subsequently used to correlate and compromise user anonymity. Techniques such as data sanitization may be required to remove any personally identifiable information from the dataset.

Getting consent from Tor users. Users must be informed that they are subject of research and that their consent is required. Users of anonymity networks might not be agreeable to the idea of being identified as a matter of principle.

Some of these ethical issues can be overcome if testing is done inside an isolated Tor network. On the other hand, it is not possible to conduct all manner of testing such as statistical data collection because the traffic type and network scale will not match the real Tor network. However, it is argued that at least for a certain class of experiments and to avoid the ethical issues mentioned above, it is preferable to have an isolated, albeit small-scale Tor network.

Even if no user data collection will be performed on the live Tor network it still carries some other risks. For example, the introduction of experimental Tor relay nodes might cause real users to be routed through the test relays. The users might then be compromised because of misconfiguration or errors in the relays. It might cause network performance problem or exposure of user identity (loss of anonymity). Another thing that cannot be done on the live Tor network is to independently create a Tor directory server as an experiment on the live Tor network because the list of directory servers is hard-coded into the Tor source code [19].

V. Tor Security and Performance Features

A few Tor security and performance features will be described next. Different aspects of security will be addressed by different security features. Additionally, performance related features will be explained briefly in the relevant sections.

Perfect forward secrecy

Instead of encrypting a single data structure many times over, Tor employs incremental or telescoping path-building design [13]. The node that initiates the circuit negotiates session keys individually with each hop in the circuit. Session keys are expired after some time and thus cannot be used to decrypt old traffic. This is to ensure perfect forward secrecy. Therefore a hostile node cannot simply record a session traffic and start a replay attack by asking a compromised node to decrypt the traffic. As a result, the circuit building process is more reliable and no replay detection is necessary.

Support of TCP-based programs using SOCKS proxy

No modification of any Transmission Control Protocol (TCP)-based programs is necessary to run them over Tor. This is because Tor uses the standard Socket Secure (SOCKS) proxy interface [13]. Therefore, there is no need to write separate application proxies for each application protocol. Most modern web browsers can function as SOCKS proxy client, including the Tor web browser client which is based on Mozilla Firefox.

No traffic shaping

Tor does not support giving priority for certain type of traffic over another or any traffic shaping [13]. Torchestra [20] is a traffic shaping algorithm that has been proposed by researchers. Additionally, there has yet to be any economical and practical traffic padding scheme that can increase security against anonymity compromising attacks [13].

Multiplexing many TCP streams over a single circuit

Tor multiplexes many TCP streams onto a single circuit [13]. One justification for doing this is because public key cryptography operations are expensive in terms of CPU cycles [21]. Factor in the fact that every Tor circuit requires multiple public key cryptography operations and it is clear why multiplexing was chosen. The second justification is because if every single request requires a fresh circuit, it might reveal more information than is necessary and that might cause a threat to user anonymity.

Leaky-pipe circuit topology

In certain scenarios, it is desirable to allow the Tor circuit initiators to route traffic to nodes in the middle of the established circuit [13]. This way, any adversary listening at the end of the circuit cannot see the traffic leaving the end node. This so-called “leaky-pipe” topology may therefore defend against certain traffic shaping and volume based attacks. Tor hidden service providers can use this topology to securely offer their services. Under this scheme, the server that offers the hidden service can do so anonymously by setting up a rendezvous point that utilises the ability to exit the Tor circuit from the middle.

Centralised congestion control

Tor uses a series of end-to-end acknowledgements (ACKS) as a form of centralized congestion control [13]. This allows congestion or flooding detection by end nodes while maintaining anonymity. Congestion can then be brought under control by throttling the amount of data sent by the end nodes.

Directory servers

Tor uses trusted nodes called directory server as a means to disseminate information about known routers and their current state such as uptime, bandwidth, exit policies, Tor version, platform, public keys, et cetera [13]. This approach is chosen instead of flooding state information through the whole network. In order for the directory information to be fully trusted, the directories are cryptographically signed, thus ensuring only reliable and trustworthy information are stored inside them.

Flexible exit policies

Each Tor node has exit polices that describe which hosts and ports it will connect to [13]. Since Tor is a volunteer-based, distributed network infrastructure the exit policies make it convenient for volunteers to participate based on their comfort level and willingness to route certain type of traffic and to certain destinations of their choice.

End-to-end integrity checking

Integrity checks are done on data before it leaves the Tor network. It is therefore difficult and almost impossible to alter the contents of data cells that passes through a node without failing the integrity check [13]. For example, alterations such as marking or tagging the data for later observation as it leaves the Tor network is difficult to do. Likewise, the integrity test will also fail if the destination web server inside a request is changed arbitrarily.

Rendezvous points for hidden services

Rendezvous points are used to protect the actual location of hidden servers. The client and the hidden server can negotiate the location where they wish to "meet" to request and receive services. This way neither the client nor the server need to know each other's real identities.

We have just described some characteristics of the Tor network that must be faithfully replicated in the experimental setup.

VI. Tor Experimental Research Requirements

Some of the parameters that are necessary in order to accurately model the proper dynamics of the live Tor network are (Bauer et al., 2011):

- Distribution of Tor router bandwidth
- Tor router exit policies
- Tor client behaviours
- Application traffic models

The actual selected parameters depend upon the chosen method of study. Apart from measuring and analyzing the live Tor network, current methods include simulations, analytical modelling, simulations and small-scale network emulation [14]. Certain parameters are more suited to certain method of study. It can be argued that conducting experiments directly on the live Tor network would yield the most accurate results but this method has some serious ethical implications and risks that must be considered [18]. The implication and risks include divulging the actual Tor users identities and adversely affecting the Tor network performance.

In order to maintain fidelity, the Tor code must be run unmodified. This way the experiment can actually mimic the actual Tor behaviour.

Research on Tor network may be about hardening it to withstand anonymity attacks or about improving its network performance. The challenge is to make it safe enough so as not to put user anonymity in jeopardy while at the same time keeping it as realistic as possible. The results can then be relied upon to improve understanding about the live Tor network.

Experimentor [14] is a Tor network emulation toolkit that can model the distribution of Tot router bandwidth, client traffic loads and applications. It provides a high degree of realism without the risk of harming the live Tor users either through loss of anonymity or reduced network performance [14].

In this paper, unmodified Tor code is chosen to ensure that experiments are run with sufficient fidelity. Default values are kept where possible or a set of static values are predefined in the Tor configuration file for each Tor relay (/etc/tor/torrc). Relay exit policies are configured by setting the appropriate ExitRelay parameter values to reflect whether the relay is an exit node. Tor client activities are limited to running the standard "curl" tool to retrieve web pages at certain Uniform Resource Locator (URL). The experimental testbed created using VMware Fusion does not allow easy scripting to start multiple clients with distinct applications. Therefore, no application traffic model is taken into consideration. This does not affect the goal of the experiment which is to conduct basic attack against anonymity and not to address network performance specifically.

VII. VMware Fusion Virtualization Technology

VMware virtualization solution runs on Linux, Microsoft Windows and MacOS platforms. The free version called VMware player are limited [22] in features compared to the paid version. VMware allows different types of operating systems to be installed including many Linux variants, Microsoft Windows

and MacOS. This presents the opportunity to experiment with Tor running on multiple virtual machines running on different operating system types.

There are three basic networking options to link up the created virtual machines. The first option is a "Host-only" network which is a virtual network that only allows communication with the underlying host operating system. The second option is a "Network Address Translation (NAT)" network. This option allows the virtual machines to access the Internet through the host network interface. The third option is to connect the virtual machines directly to the physical network of the host machine, called the "Bridged" network. Additionally, VMware Fusion allows a maximum of six more isolated virtual networks. Taken together, VMware Fusion appears to provide more flexibility and options for the researcher.

VIII. VMware Fusion as a Tor Virtualization Platform

Under VMware, the choice of operating systems is not limited to Linux or any particular Linux distribution. The experimenter is therefore free to choose any of the supported variants of Linux or Microsoft Windows platforms as a base to install the Tor directory server, relays and client.

VMware Fusion is specifically designed for the MacOS (Apple Macintosh operating system). It is mostly identical to the VMware virtualization technology for the Microsoft Windows and Linux platforms called VMware Workstation. Unless there are specific differences in features, this paper shall assume that any discussion about VMware Fusion is applicable to VMware Workstation as well.

Using VMware forces the Tor experimenter to first install the base operating system before installing the Tor components. It is proposed that this provides greater learning opportunity because of the need to be familiar with the operating system installation process, the networking configuration steps, and the process of updating and maintaining software components. This is very close to running and maintaining a real Tor server for the live Tor network. On the other hand, using Docker and Netkit often means using pulling a Linux image (Docker) or a set filesystem (Netkit), which may arguably require less work but at the expense of realistic system administration experience. This results in a trade-off between ease of use and realistic administrative experience.

Installing a Tor node using VMware produces a virtual machine but the operating system acts and functions exactly like it would be on a dedicated hardware. Features such as graphical user interface (GUI) is available for the experimenter if desirable. All available tools on the operating system are accessible, including GUI-based ones. The trade-off is a bigger footprint, higher resource requirement and heavier demand on the host machine's central processing unit (CPU).

Keeping the individual virtual machine operating system up to date is also easier using the VMware approach. All the package manager tools are available which makes it easier to update the Tor nodes compared to running Docker containers, for example. It is also possible to only update a specific instance of a Tor node because it is independent of all other instances. This is not the case with other virtualization technologies like Docker or Netkit.

After considering the advantages mentioned, it was decided to use VMware Fusion as the virtualization platform to create and install Tor nodes and gain realistic experience of assembling the actual setup and behaviour of the Tor network.

IX. Physical Setup

The experiments described in this paper was conducted on the following hardware setup. The host hardware is an Apple MacBook Pro (Retina, 15-inch, Mid 2015), with a 2.8 GHz Intel Core i7 processor, 16 GB 1600 MHz DDR3 Random Access Memory, AMD Radeon R9 M370X 2048 MB graphics card and 1 TB flash storage. The virtualisation platform is VMware Fusion Professional version 8.5.6 (5234762) running on top of OS X Yosemite version 10.10.5.

One new virtual network called vmnet7 was added to the VMware Fusion networking stack by making changes to the networking configuration settings. This new virtual network and the built-in NAT (network address translation) network were used to create a single flat network.

The Tor nodes were installed on Ubuntu 16.04.2 LTS (Xenial Xerus) which was then the latest stable long-term support version of Ubuntu. This means configuration information and support were easily available on the web. Tor software version 0.2.9.10 was installed using Ubuntu "apt-get" command to ensure the latest stable and supported release of Tor is used (circa April 2017).

The real Tor network consists of Tor nodes and directory servers distributed in different networks and autonomous systems (AS) across different Internet Service Providers (ISPs) in different countries. This requires them to be connected via internetwork routers that exchange and maintain routing information via routing protocols or static routes. A decision had to be made whether to create multiple networks for the experiments in this paper to add some degree of realism to the experimental setup.

It was finally decided after much trial and error that a single flat network will be used for the experiments. The reasons are as follows:

- For the purpose answering the research question of this paper it was decided that having multiple networks will complicate the setup unnecessarily.
- Routing functionality would have to be introduced for multiple networks. This can be done by installing and configuring additional virtual machines that will function as routers. In the actual Tor network the routers are usually special purpose devices that implement routing functions. It was decided that implementing routing functionality inside of general purpose operating system installed on a virtual machine adds little benefit to the research purpose. The same argument

applies even if virtual routers based on customized Linux (such as Smoothwall) or FreeBSD (such as pfSense) are used.

• Routing configuration can be done using routing protocols such as BGP, EIGRP, RIP or using static routes. Some level of expertise is required to set up and troubleshoot those protocols and settings. The research purpose for this paper has more to do with general Tor behaviour and anonymity features and less to do with network performance. It was decided to choose ease of implementation over realistic but complicated internetwork routing setup that would have minimal benefit to the research purpose.

X. Logical Setup

The experimental setup consists of 3 non-exit relays, 3 exit relays, a directory server, a DNS server, a Tor client machine and a web server. A mixture of 3 non-exit relays and 3 exit relays was chosen so that the Tor circuits created will vary in composition. This will generate multiple distinct circuits in the experimental setup such that Tor relay selection and behaviour can be studied.

The experimental Tor network has a web server that is used as a target by the client. The web server runs Apache software with PHP add-on library. Whenever the client makes a successful connection, the web server returns a PHP generated page that displays the IP address of the client. If the Tor network works as expected the IP address will belong to the exit relay being used at the time the connection is made through the Tor network.

Only the three exit relays (exit1, exit2 and exit3) and the DNS server are configured with gateways to the Internet. For this purpose, they are each configured with two virtual network interface cards (NICs). The first NIC is connected to the NAT interface of the host machine to provide Internet gateway functionality. The second NIC is connected to the experimental Tor network. The Tor client machine's DNS resolver is set to the DNS server's IP address and no default gateway or any route to the Internet is set on the client machine. This is to ensure that the client machine can only use the DNS server to resolve a fully qualified domain name (FQDN) of a host on the Internet but cannot by itself make the actual connection. However, if the connection request is made over the experimental Tor network it should succeed because it will be one of the exit nodes that is making the final outbound connection to the Internet. This is the expected result if the experiment is successful.

All virtual machines is set up to be in the same private network of 192.168.1.0/24. All exit nodes and the DNS server have a second connection to 192.168.177.0/24 network which is the NAT network that has a gateway to the Internet.

Tor allows the port numbers for making Tor connections and for fetching updates to the Tor directory to be set by the experimenter. The two other ports that can also be set are the SOCKS proxy port that the client uses to connect to the Tor network and the Tor control port that is used to control aspects of the Tor operations. The ports configured in this experimental setup are described in Table 1.

Table 1: Configured port numbers

Port Name	Port Number	Purpose
SocksPort	9050	Connecting to SOCKS proxy (by the client)
DirPort	7000	Fetching updates to the directory
OrPort	5000	Making Tor connections
ControlPort	9051	Controlling aspects of Tor operations

To summarize, the small-scale experimental Tor network in this project comprises a single directory server, 3 Tor non-exit relays, 3 Tor exit relays, a target web server, a Tor client and a DNS server. The configuration was based on hints and ideas mentioned by Ritter [23], Liu [24], Fukui [25] and antitree [26].

XI. Connectivity via Tor Network to a Web Server

For this experiment, the Tor client attempts to connect to a web server through the created Tor network. A total of 3 relays and 1 client are participating in the experiment. One exit relay (exit1) and 2 non-exit relays (relay1 and directoryserver) form the Tor circuit. The web server and the DNS server are not part of the Tor circuit. The setup is shown in Figure 4.

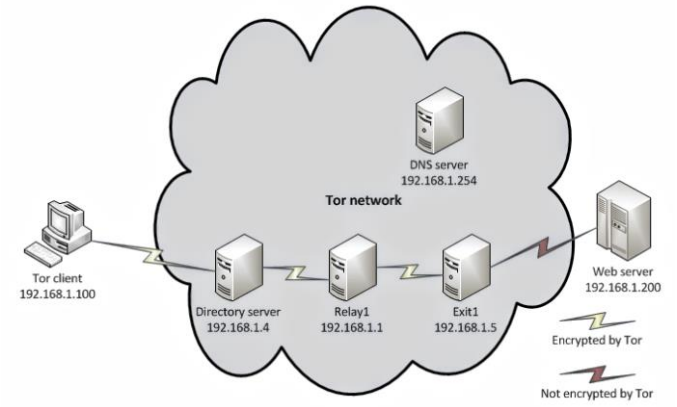

Figure 4: Basic connectivity via Tor network to a web server

The client uses the Linux "curl" command to retrieve a web page that contains the remote address of the connecting machine. This is the first step towards proving that the setup actually functions as a Tor network. Traffic originating from the client must be routed to the entry relay, passed on to the middle relay and then to the exit relay before finally sent to the destination web server. It must also be shown that traffic sent into the Tor

network setup are properly encrypted at each Tor relay to preserve the Tor client's anonymity.

Figure 5 and Figure 6 show that the Tor client connection into the Tor network to reach the web server is successful. It is also proved that the remote IP address of the machine making the connection to the web server is that of the exit relay (relay1) instead of the Tor client.

```
root@client:~#
root@client:~# curl -x socks5://localhost:9050 http://192.168.1.200
Remote address: 192.168.1.5
Forwarded for:
root@client:~#
```

Figure 5: Output of the curl command running on the client showing the IP address of exit1

Figure 6: Using Firefox web browser on the client to access the web server via Tor

The 3-hop Tor circuit created for use by the Tor client is shown in Figure 7. The circuit comprises directoryserver (192.168.1.4) which also functions as a Tor relay, relay1 (192.168.1.1) and exit1 (192.168.1.5).

```
arm - client (Linux 4.4.0-31-generic)      Tor 0.2.9.10 (none recommended)
Relaying Disabled, Control Port (cookie): 9051
cpu: 0.0% tor, 1.0% arm    mem: 12 MB (1.2%)   pid: 3429   uptime:

page 2 / 5 - m: menu, p: pause, h: page help, q: quit
Connection Details:
  address: 192.168.1.4:5000
  locale: ??   fingerprint: EC2E2B60DA8B2E2E4E92882FD28041644B75B294
  nickname: directoryserver          orport: 5000       dirport: 7000
  published: 12:19:17 2017-05-22
  flags: Authority, Exit, Fast, Guard, HSDir, Running, Stable, V2Dir, Valid
  exit policy: unknown

192.168.1.100:47832     -->  192.168.1.4:5000                1.4d (OUTBOUND)
192.168.1.100           -->  192.168.1.5                    51.2s (CIRCUIT)
  192.168.1.4                  directoryserver                 1 / Guard
  192.168.1.1                  relay1                          2 / Middle
  192.168.1.5                  exit1                           3 / Exit
192.168.1.100           -->  192.168.1.5                    46.2s (CIRCUIT)
  192.168.1.4                  directoryserver                 1 / Guard
  192.168.1.1                  relay1                          2 / Middle
  192.168.1.5                  exit1                           3 / Exit
```

Figure 7: Tor circuits created on the client machine

In order to verify that the anonymity of the Tor client is preserved by this Tor network, a series of packet captures was done at the following machines using the TCPDUMP utility:

- client
- directoryserver
- relay1
- exit1
- webserver

Wireshark protocol decoder was used to view the captured packets graphically. The order of decoded traffic viewed are as follows:

client → entry relay → middle relay → exit relay → web server

The first packet capture decoded are the packets captured at the Tor client, which is the SOCKS proxy traffic over port 9050, shown in Figure 8. The SOCKS proxy connection is made via the loopback interface, 127.0.0.1.

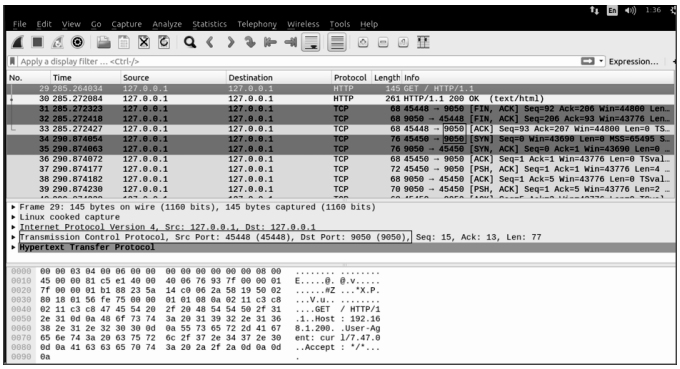

Figure 8: Protocol decode of the SOCKS proxy connection over port 9050 at the client

The second packet capture decoded are the client packets going to port 5000, which is the Onion Router port setting in the Tor configuration files of all the Tor relays in the setup. In this case, these are the packets sent by the Tor client to the entry relay.

Figure 9 displays the protocol decode which shows the client (192.168.1.100) making a connection to the entry relay (192.168.1.4).

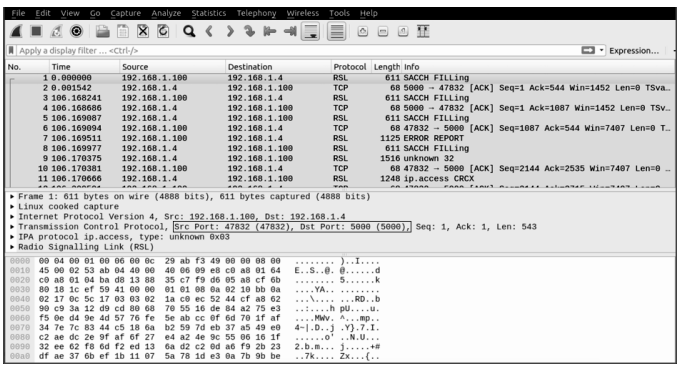

Figure 9: Protocol decode of the client connection to the entry relay of the Tor network (client perspective)

The third protocol capture decoded was for the packets captured at the entry relay (192.168.1.4), as shown in Figure 10.

It is possible to clearly see the client IP address (192.168.1.100). This is expected because the entry relay must know about the client that is making the connection into the Tor network.

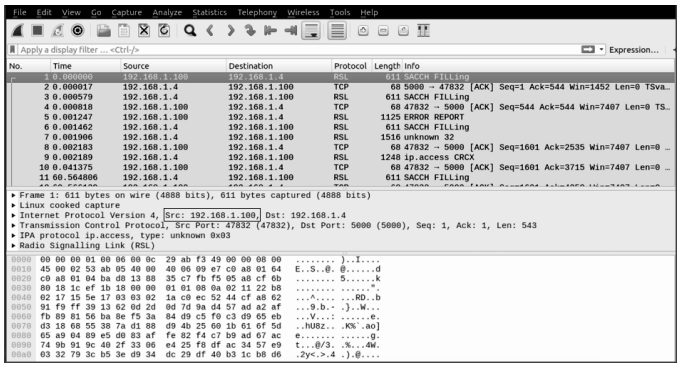

Figure 10: Protocol decode of the client connection to the entry relay of the Tor network (entry relay perspective)

The fourth protocol capture decoded was for the packets captured at the middle relay (192.168.1.1). As shown by Figure 11, there is a connection made by the entry relay (192.168.1.4) to the middle relay (192.168.1.1) but the client's IP address (192.168.1.100) is not shown.

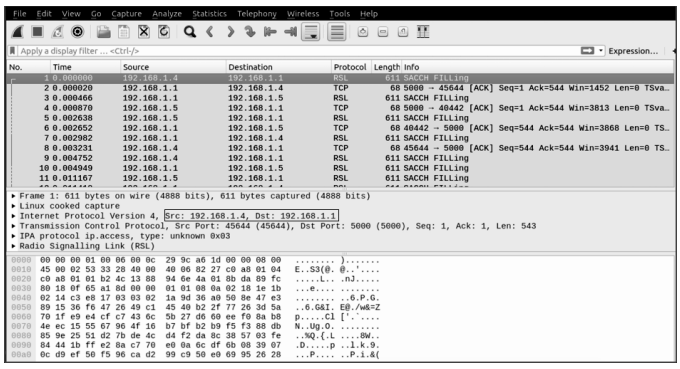

Figure 11: Protocol decode of the entry relay making a connection to the middle relay

The fifth protocol capture decoded was for the packets captured at the exit relay (192.168.1.5). As can be seen in Figure 12, the middle relay (192.168.1.1) is making a connection to the exit relay (192.168.1.5). Once more the client IP address (192.168.1.100) is not visible in the protocol decode.

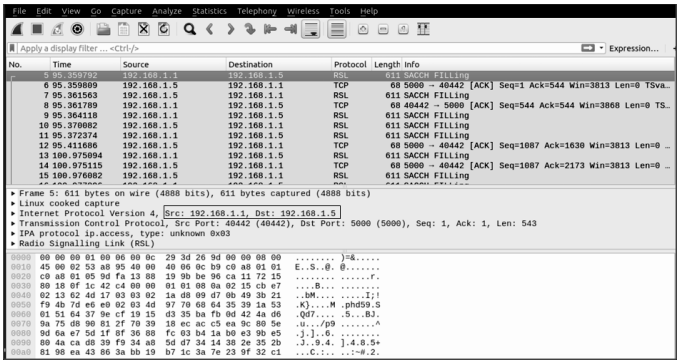

Figure 12: Protocol decode of the middle relay making a connection to the exit relay

The sixth packet capture decoded was for all packets destined for port 80 (HTTP) at the exit relay. This is what is visible to the exit relay as it makes the final outgoing connection to the target webs server. As Figure 13 shows, the exit relay (192.168.1.5) is making the connection to the web server (192.168.1.200). The client identity stays hidden and invisible under protocol analysis.

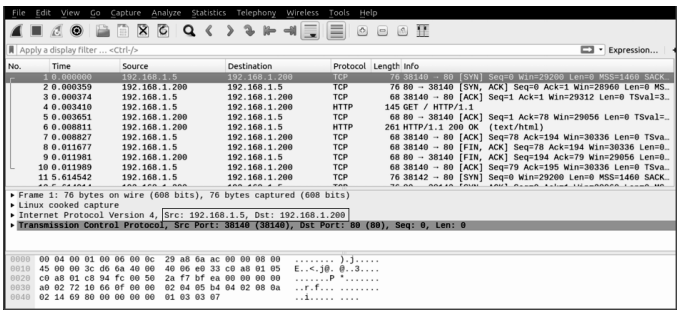

Figure 13: Protocol decode of the exit relay making a connection to the web server (exit relay perspective)

The final protocol capture decode was for traffic captured at the web server (192.168.1.200). It is confirmed here too that the web server only sees the request coming from the exit relay (192.168.1.5) and not from the client (192.168.1.100). The protocol decode is shown in Figure 14.

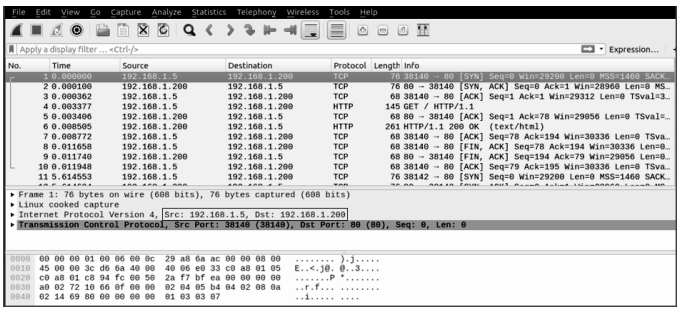

Figure 14: Protocol decode of the exit relay making a connection to the web server (web server perspective)

This experiment shows that the created Tor network behaves as expected in the most fundamental sense. Client anonymity is preserved except to the entry relay. Likewise, only the exit relay knows the final destination of the client request.

XII. Analysis and Discussion

VMware virtualization technology proved to be a viable platform to install, configure and manage Tor nodes. The installation process requires no Tor source code modification and closely resemble actual real-world installation steps. The results from the experiments show that the created network behaves just like a Tor network.

Each Tor node had to be installed from scratch, starting with the base operating system. Each virtual machine had to be provisioned and allocated the required memory and disk space, which is similar when dealing with discrete physical machines. VMware supports the Open Virtualisation Format (OVF) or Open Virtual Appliance (OVA) format to create virtual machine templates. This enables additional nodes to be created faster compared to using installation disk images each time.

When running extended tests it is advisable to turn off automatic updates in the host operating system. This is to prevent automatic reboots that will terminate the running tests prematurely.

In order to ensure the network remains isolated and not route traffic to the actual Tor network, the "TestingTorNetwork" option was enabled on all nodes and private IP addresses (RFC 1918) was used.

As expected, the created Tor network can generate Tor circuits that can be used for basic traffic analysis experiments. The Tor client is able to automatically select and use any available exit relays.

The results of the packet sniffing experiment along the various Tor nodes show that the virtual Tor network supports the Tor encryption layers and anonymity is not compromised by packet sniffing and decoding.

XIII. Future Work

It was demonstrated that the experimental Tor network created using VMware Fusion virtualisation platform produced results that mimic the behaviour of the real Tor network. As a future work, an evaluation of whether the same virtualisation platform can support more nodes is of research interest. Another anticipated future work would be to evaluate whether the same virtual network can be expanded to include multiple networks in order to more closely mirror the real Tor network. In the experiment, the type of traffic generated by the Tor client was limited to web browsing and fetching a URL using the curl command. This is not because the platform cannot support it but more to do with the need to find creative ways to generate other traffic types, something that is left for future work. An evaluation of whether the same platform can be scripted to generate multiple types of user traffic is also of great research interest.

XIV. Conclusion

This research wanted to find out whether a functional, isolated Tor network can be created using VMware virtualisation platform. Experiments were conducted over the network to confirm the Tor network functionality. The results show that the virtual Tor network behaves much like the actual Tor network even though it is limited by the number of nodes and client created.

Creating a virtual Tor network comprising discrete virtual machines for directory server, relay, client and target web server offers a realistic and rich experience for the researcher. It is also convenient to investigate what is happening inside each Tor node using the available graphical user interface and the ability to have multiple terminal windows opened at the same time. The researcher also has access to all Linux tools, making it easy to manage the nodes. In conclusion, the virtual machine based Tor network provides another useful tool for the researcher.

REFERENCES

[1] Chen, H., Beaudoin, C.E. & Hong, T., 2017. Securing online privacy: An empirical test on Internet scam victimization, online privacy concerns, and privacy protection behaviors. Computers in Human Behavior, 70(January), pp.291–302. Available at: http://dx.doi.org/10.1016/j.chb.2017.01.003.

[2] Henze, M. et al., 2016. A comprehensive approach to privacy in the cloud-based Internet of Things. Future Generation Computer Systems, 56, pp.701–718.

[3] Weber, R.H., 2010. Internet of Things - New security and privacy challenges. Computer Law and Security Review, 26(1), pp.23–30. Available at: http://dx.doi.org/10.1016/j.clsr.2009.11.008.

[4] Anton, A.I., Earp, J.B. & Young, J.D., 2010. How Internet Users ' Privacy Concerns Have Evolved. IEEE Privacy & Security, 1936(February), pp.21–27.

[5] Zarsky, T., 2003. Thinking outside the box: considering transparency, anonymity, and pseudonymity as overall solutions to the problems in information privacy in the internet society. U. Miami L. Rev., 1, pp.1–54. Available at: http://heinonlinebackup.com/hol-cgi-bin/get_pdf.cgi?handle=hein.journals/umialr58&section=57.

[6] Christopherson, K.M., 2007. The positive and negative implications of anonymity in Internet social interactions: "On the Internet, Nobody Knows You're a Dog." Computers in Human Behavior, 23(6), pp.3038–3056.

[7] Kennedy, H., 2006. Beyond anonymity, or future directions for internet identity research. New Media & Society, 8(6), pp.859–876. Available at: http://journals.sagepub.com/doi/10.1177/1461444806069641.

[8] Muftic, S., Abdullah, N. & Kounelis, I., 2016. Business Information Exchange System with Security , Privacy , and Anonymity. , 2016.

[9] Child, J.T. & Starcher, S.C., 2016. Fuzzy Facebook privacy boundaries: Exploring mediated lurking, vague-booking, and Facebook privacy management. Computers in Human Behavior, 54(January), pp.483–490. Available at: http://dx.doi.org/10.1016/j.chb.2015.08.035.

[10] Dawson, M. & Cárdenas-Haro, J.A., 2017. Tails Linux Operating System. International Journal of Hyperconnectivity and the Internet of Things, 1(1), pp.47–55. Available at: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/IJHIoT.2017010104.

[11] Cole, J., 2016. Dark Web 101. Air & Space Power Journal, (May), p.6. Available at: http://www.dtic.mil/docs/citations/AD1005862.

[12] Stoycheff, E., 2016. Under Surveillance. Journalism & Mass Communication Quarterly, 93(2), pp.296–311. Available at: http://journals.sagepub.com/doi/10.1177/1077699016630255.

[13] Dingledine, R., Mathewson, N. & Syverson, P., 2004. Tor: The second-generation onion router. SSYM'04 Proceedings of the 13th conference on USENIX Security Symposium, 13, p.21. Available at: http://portal.acm.org/citation.cfm?id=1251375.1251396.

[14] Bauer, K. et al., 2011. ExperimenTor : A Testbed for Safe and Realistic Tor Experimentation. CSET.

[15] Daniels, J., 2009. Server Virtualization Architecture and Implementation www.acm.org/crossroads Crossroads. , 16(1). Available at: http://www.science.smith.edu/dftwiki/images/7/7f/ServerVirtualizationArchitectureAndImplementation2009.pdf [Accessed May 16, 2017].

[16] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A., 2007. kvm: the Linux virtual machine monitor. Proceedings of the 2008 Linux Symposium, 1, pp.225–230.

[17] Pék, G. et al., 2014. On the Feasibility of Software Attacks on Commodity Virtual Machine Monitors via Direct Device Assignment. Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (AsiaCCS'14), pp.305–316. Available at: http://dx.doi.org/10.1145/2590296.2590299.

[18] Loesing, K., Murdoch, S.J. & Dingledine, R., 2010. A Case Study on Measuring Statistical Data in the Tor Anonymity Network. International Conference on Financial Cryptography and Data Security. Available at: http://sec.cs.ucl.ac.uk/users/smurdoch/papers/wecsr10measuring.pdf [Accessed May 17, 2017].

[19] The Tor Project, I., 2017d. tor - Tor's source code. Available at: https://gitweb.torproject.org/tor.git/tree/src/or/config.c [Accessed June 18, 2017].

[20] Gopal, D., 2012. Torchestra : Reducing interactive traffic delays over Tor. University of California, San Diego.

[21] Uludag, S. et al., 2014. Secure and Scalable Communications Protocol for Data Collection with Time Minimization in the Smart Grid. , (2122), p.14. Available at: http://hdl.handle.net/2142/49985.

[22] VMware, I., 2017. Workstation for Windows - VMware Products. Available at: http://www.vmware.com/products/workstation.html#compare [Accessed May 15, 2017].

[23] Ritter, T., 2014. Run Your Own Tor Network - ritter.vg. Available at: https://ritter.vg/blog-run_your_own_tor_network.html [Accessed May 21, 2017].

[24] Liu, F., 2015. How to Setup Private Tor Network - Programming Languages | Fengyun Liu. Available at: http://fengy.me/prog/2015/01/09/private-tor-network/ [Accessed May 21, 2017].

[25] Fukui, T., 2017. Anonymous Routing of Network Traffic Using Tor. Available at: https://witestlab.poly.edu/blog/anonymous-routing-of-network-traffic-using-tor/ [Accessed May 21, 2017].

[26] antitree, 2016. Run a private tor network using Docker. antitree.com. Available at: https://www.antitree.com/2016/07/01/run-a-private-tor-network-using-docker/ [Accessed May 11, 2017].