Cybersecurity Paper Compilation FEUP – Fall 2017

Ricardo Morla
ricardo.morla@fe.up.pt
INESC TEC, Faculty of Engineering, University of Porto
Porto, Portugal

***Abstract*—This is a compilation of cybersecurity papers developed by master-level network engineering students at FEUP during the Fall of 2017. This compilation covers different topics in cybersecurity – from past and new cryptography algorithms and applications to side channel attacks, intrusion detection, denial of service mitigation, authentication mechanisms, software defined networking, IoT middleware, and the design of a CTF – all seen through the perspective of a network engineer.**

I. Topics

Performance is an important issue to network engineers. As such we start this compilation by looking at the performance of a set of well known cryptographic libraries implemented in different programming languages, followed by a study of the overhead of the number of circuits in the TOR onion routing privacy protecting system, a simulation of a bitcoin network to study the impact of the number of miners, nodes, and block size in block propagation time. We also characterize the processing delay of a well known intrusion detection system under different web attacks, compare the overhead of two Virtual Private Network solutions, and characterize the authentication delay of an open source single sign on solution with increasing load of authenticating users.

Understanding and developing new solutions is equally important for a network engineer. To explore this we look into the architecture of the Signal protocol, compare different solutions for the security of IoT middleware, and propose an implementation of a network access control mechanism using the software defined approach and tool set. We also propose early stage solutions for web fingerprinting – which uses dynamic time warp to cluster and classify encrypted web traffic to different web pages and web sites – and for DDoS mitigation – which uses cookies to split traffic from signed on users from other traffic at comparable overhead with commercial reverse proxy solutions. Finally, we attempt to characterize the different aspects of a Capture the Flag competition organizing challenges by topic and assessing a publicly available CTF platform.

II. Notes on Extent and Quality

Although all projects were supervised and had the potential for development, much was left to the responsibility of the students and, as such, the extent and the quality of the papers and of the actual projects varies greatly. The DDoS and fingerprinting projects have not only proposed and developed relatively complex solutions to their problems but also measured their performance and accuracy and presented results in clear and well written papers. The TOR, crypto library, and bitcoin projects are also worth highlighting given the extent of their performance evaluations.

The papers were not edited after submission and can have a significant number of errors – depending on the paper. One noticeable example is using the word stenography when referring to the concept of steganography and of concealing a secret in an image or other plaintext information. Typos and formatting issues also abound, and while most papers are written in English, some are in Portuguese. In any case, I believe these papers are better in the public than stored away somewhere. Hopefully other students can pick up where these have left off.

III. Paper List

Performance

- *Crypto algorithms and libraries Performance characterization*, Alexandre Truppel, Luis Paulo Durão, p.2
- *Tor Network Overhead Characterization*, Vitor Fernandes, Tiago Dias, p.6
- *BitCoin – Performance Characterization*, Diana Almeida, Pedro Rodrigues, p.10
- *Impact of Network Threat Detection Engines*, Ricardo Araújo, p.14
- *VPN benchmarking*, João Polónia, Luis Barbosa, p.17
- *Authentication – Architecture Review, Latency, and Scalability*, Felipe Bahamonde, Luis Silva, p.23

Architecture and Implementation

- *Architecture of the Signal Protocol*, Joana Veiga, João Francisco, p.27
- *Overhead of Network Security Protocols and Viability for IoT*, Joana Enes, José Pintor, p.30
- *SDN/NFV Review and Performance of Network Security Functions*, Pedro Gomes, Erick Caetano, p.34
- *Side Channel Attacks*, Eduardo Rodrigues, Hélio Puga, p.36
- *DDoS Attack and Defense Strategies*, João Magalhães, Luís Zilhão, p.41
- *SSRE CTF: Capture The Flag Competition Organization*, Eduardo Sobrinho, Rodrigo Vaz-Pires, p.44

Crypto algorithms and libraries Performance characterization

Alexandre Truppel
MSc in Electrical and
Computer Engineering
FEUP,
Porto, Portugal
Email: up201303442@fe.up.pt

Luis Paulo Duro
MSc in Electrical and
Computer Engineering
FEUP,
Porto, Portugal
Email: up201306126@fe.up.pt

***Abstract*—Cryptography is the ultimate key to the digital world and is what keeps it secure. Thus, the performance of its implementation is also paramount given the amounts of data that are encrypted every day. Here, the performance (CPU load, memory load and IO usage) of various libraries for various algorithms is analysed and discussed. Various languages are also compared. The tool developed for this analysis is explained in detail.**

I. Project Description and Approach

The project was proposed by the professor, Eng. Ricardo Morla. Its goal was to create a framework to evaluate the performance of different encryption algorithms and of the libraries that implemented them in different programming languages.

We were to also use the created framework to take some conclusions about the encryption algorithms and their respective implementations.

A. Selected algorithms

We selected the algorithms with preference to the ones referenced in the classes throughout the semester.
For the hash algorithms, we chose:

- MD5
- SHA-1
- SHA-256

For the block cyphers, we chose:

- 3DES
- AES-256
- Blowfish

The CBC mode was chosen for the block cyphers due to its widespread use and because it was the only one implemented by all the libraries.
For asymmetric encryption, we chose:

- RSA

B. Selected languages and libraries

We selected three of the most commonly compiled languages, C, C++ and Java, and Python from the list of the most common interpreted languages.

Next are presented the libraries chosen for each language.

For C:

- OpenSSL
- Libgcrypt

For C++:

- Crypto++

For Java:

- BouncyCastle

For Python:

- CryptoPy

Not all libraries implement all the chosen algorithms so the table I represents whether a library implements a given algorithm or not.

	MD5	**SHA-1**	**SHA-256**	**3DES**	**AES-256**	**Blow-fish**	**RSA**
Open-SSL (C0)				CBC	CBC	CBC	OAEPad
Libg-crypt (C1)				CBC	CBC	CBC	OAEPad
Crypto-++ (Cpp0)				CBC	CBC	CBC	OAEPad + SHA
Bouncy-Castle (Java0)				CBC	CBC	CBC	OAEPad + SHA
Py-Crypt (Python0					CBC		

■ - Implements
■ - Doesn't implement

TABLE I
Library/Algorithm implementation

C. Metrics

In order to compare, one needs metrics of performance. We chose:

- CPU time
- Max memory used
- Bytes read from disk

CPU time was chosen over execution time because it is the time actually dedicated to the encryption process which is proportional to the number of CPU cycles required to perform that same process. The goal of this is to make it possible to compare the results measured in different machines and to make the results independent of the CPU load imposed by other processes running at the time of the test.

II. Testbench Architecture

We opted for a modular architecture so it is easily scalable and can quickly allow other implementations, languages and algorithms to be added.

The testbench is divided in three parts: a framework for measuring the resources used by a Process, the master and the slaves.

A. Framework

The framework was created in C++ for Linux and measures the defined metrics. A code example is presented in figure 1.

```
Process* test = new Process("python
  ↪ slave.py data10M 1000 4 0 0");
test->start();
test->join();
test->getResources();
```

Fig. 1. Example of master's way of calling a slave

The CPU and memory are measured by parsing the output of */usr/bin/time*.

The master measures the IO by constantly reading the */proc/$1/io* file until the slave ends. When it does, present the last read value as the read IO.

B. Master

The master is responsible for calling slaves and measuring the resources used by the slaves using the framework. It also outputs the results to a csv file so they can be easily interpreted by using any number processing program like *Octave* and *LibreOffice Calc*.

1) Configuration file: The master is configurable using a configuration file, making it easy to add new test combinations.

Each line must that the format *<key>=<value>*.

It allows the following keys to be used:

- *SLAVES_DIRECTORY* - The path to the directory where the slaves are located
- *PRE_COMMAND_STRING* - This key contains the code that will be run before any slave-execution code (for changing working directory, for example). If more that one command is required, chain with ";", "&&" or "||"
- *DATA_SIZES* - Array of sizes (in bits) of files to be encrypted. Data sizes can have the following multiples:
  - *K* - kylobits, $*1024$
  - *M* - megabits, $*1024^2$
  - *G* - gigabits, $*1024^3$
- *N_REPEATS_<Data size>* - There must be one of these keys per data size input in "DATA_SIZES"

Following is a possible configuration file:

```
SLAVES_DIRECTORY = Projeto/
#PRE_COMMAND_STRING = cd ./ &&
DATA_SIZES = 10K 500K 1M 5M 10M 1G

N_REPEATS_10K = 4000
N_REPEATS_500K = 300
N_REPEATS_1M = 120
N_REPEATS_5M = 60
N_REPEATS_10M = 50
N_REPEATS_1G = 1
```

Fig. 2. Configuration file example

C. Slaves

The slaves have the sole purpose of encrypting whatever file, whatever number of times with the algorithm and library which are passed to it by arguments.

They are designed to have the less possible overhead so the resources used by them are, as much as possible, for the encryption itself.

The slaves receive the following arguments:

1) Path to the file containing the data to be encrypted
2) Number of times the file is to be encrypted
3) The algorithm to use
4) The library to use
5) 0 or 1 to encrypt or decrypt, respectively, when applicable (does not apply to hash functions)

Usage:

```
$ python slave.py $DATA_FILE
  ↪ $N_ITERATIONS $ALGORITHM $LIB $MODE
```

Fig. 3. Slaves' usage

III. Tests performed

Because of the great number of tests and dimensions required to represent those tests, the creation of a clear and easy way of showing the results was a challenge.

Our solution was to create two phases of testing.

A. First phase

In this phase, we intend to illustrate, for each algorithm the way the resources for each language-library combination are used as the size of the data increases.

The objective is to compare libraries that implement the same algorithms.

Each test was run five times on the same computer and the maximum and minimum values were represented in graphics to get the range of values the test was in.

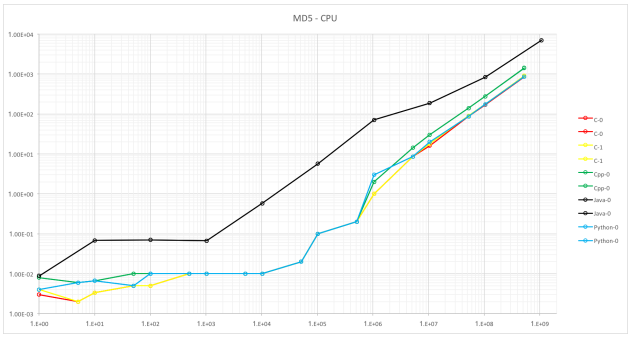

Fig. 4. 1st phase results of MD5 for CPU

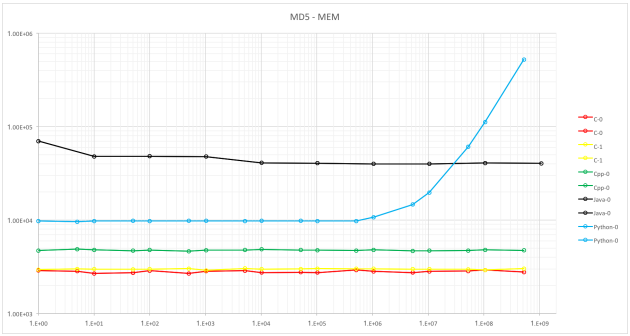

Fig. 5. 1st phase results of MD5 for MEM

The graphics related to this part can be found in the first annex [1] but the figures 4, 5 and 6 were added directly here as an example.

For the CPU time, there is an initial zone which is constant due to overhead and the fact that the encryption is done in blocks of fixed size and all data smaller than the block size will take the same time to encrypt, followed by another linear zone. The linear zone makes perfect sense because, if we discard the overhead which is less and less significant with data sizes (due to the increase in processing time), the amount of CPU time (which is proportional to the number of CPU cycles) required to run an algorithm should be proportional to the size of the data.

For the memory used, it also makes sense that it is always constant because the algorithm runs in blocks which are always of the same size, which means that, because the files will be read as they are encrypted and then that memory is freed, the memory will be constant. A peculiar thing happens in Python for high file sizes in which the time becomes exponential. Despite our research on Python's memory management, we couldn't find the reason to this spike in memory usage.

For the IO (disk bytes read), it also makes sense that it should increase with data size because, in the bare minimum, the slaves should read at least one time the size the file they are encrypting. There is, although, a singularity in the graphics between 1kB and 500kB due to the OS's caching.

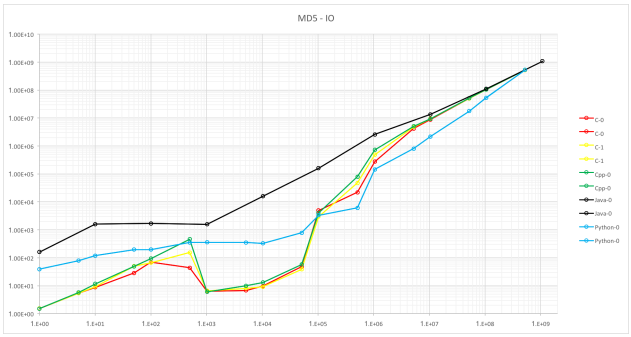

Fig. 6. 1st phase results of MD5 for IO

B. Second phase

In this phase, we intend to illustrate, for AES256 implemented by OpenSSL for C, the minimum and maximum values in the metrics using 3 different computers.

The figures 7, 8 and 9 represent the measured values.

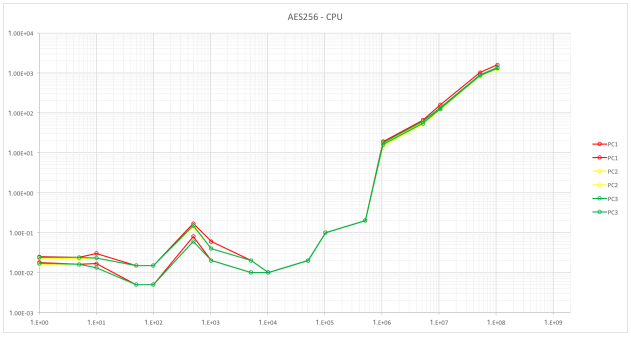

Fig. 7. 2nd phase results for CPU

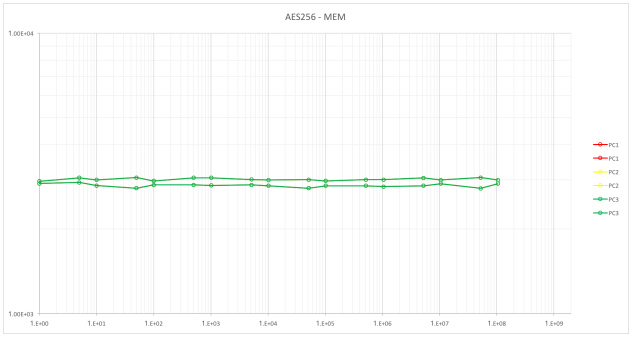

Fig. 8. 2nd phase results for MEM

All the conclusions regarding the shape of the graphics are the same as the ones taken in the first phase.

We can see that there is very little difference throughout all computers because the CPU time is relative, making it independent of CPU clock and because memory and IO should also be independent of the computer because all had the same architecture and OS (Debian Jessie).

The computers used were:

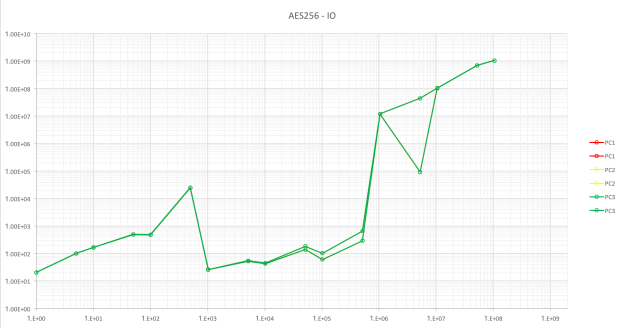

Fig. 9. 2nd phase results for IO

1) CPU: *Intel Core i5-7200U @ 3.1GHz*; Disk:SSD; Memory:DDR3@1600MHz
2) CPU: *Intel Core i7-6700U @ 4GHz*; Disk:HDD; Memory:DDR3@1333MHz
3) CPU: *Intel Core i3-8350U @ 4GHz*; Disk:SDD; Memory:DDR4@2133MHz

IV. Conclusion

The project was completed successfully and the following conclusions can be deduced regarding encryption performance:

- All CPU graphics have a constant and linear zones, except for RSA.
- RSA is always constant in memory and CPU because data is considered as an integer (which must be smaller than N, where N depends on the key size) and then is encrypted thus making the data size irrelevant for those metrics
- Python is very bad in memory utilization for encrypting big files
- Java is consistently the worst in all metrics
- OS caches files between 1kB and 500kB because the slave (the same process) repeats the encryption of the same file
- All C and C++ libraries are very alike performance-wise

From the second phase we can deduce that the results are approximately the same among all computers.

We should choose a library firstly based on the security/quality, secondly based on the performance and only then, based on the easiest to use and for the language you are already using. Assuming that all these libraries are equally secure and given that the performance is not too different, we can say that the only factor most users will need to worry about is the commodity of the usage of a given library.

References

[1] https://goo.gl/2JmiEm

Tor Network Overhead Characterization

Vitor Fernandes
Master in Electrical and Computers Engineering
Email: up201304744@fe.up.pt

Tiago Dias
Master in Electrical and Computers Engineering
Email: mieec1205017@fe.up.pt

***Abstract*—Nowadays, TOR is one of the most used tools that can provide user anonymity. Along the time, it has been increasing its user base, which allows for less delay and more load balance. Each hop on the network means one additional layer of encryption. In this work, the goal is to measure the overhead that results from the circuit building mechanism, in various use cases. Our approach was to measure transfer time of a webpage and a 1MB file, changing the length of circuits and bandwidth in each node. We obtained results showing that circuit length has no improvement in terms of anonymity, although it adds more encryption layers and, consequently, more delay. We also concluded that changing bandwidth can be critical for file downloading, while visiting websites is more likely to have the same behavior.**

I. Introduction

In this project, we will be discussing Tor Project network, that is a network which enables anonymous communication between certain nodes. Tor is an acronym for The Onion Router and is known as a free tool for anonymous traffic over the Internet.

Our motivation in this work is to create a private TOR network, in order to experiment and measure on a network which has nodes that we can control (because TOR nowadays exists as a distributed network around the world, with nodes that have some role on the organization of the network itself).

We will start by explaining the most important concepts of the TOR network and how it works, then we will show the ways that we chose to implement the private network. In the end, we will show the measurements results. This measures will be fully described after we understand the trade-offs behind this network.

II. Background

TOR is an implementation of onion routing, which consists in a technique for anonymous communication over a computer network, where the messages are encapsulated in layers of encryption. The main goal of this network is to allow organizations and individuals to share information over public networks without compromising their own identity. Tor is also an effective censorship circumvention tool, allowing its users to reach otherwise blocked destinations or content [2]. The encrypted data is transmitted through a series of network nodes called onion routers or relays. This means that the packets that travel over the tor network are fully encrypted, so the IP address remains hidden. [16]

One way to understand onion routing is to start with the concept of proxy servers. A proxy server is a server that relays your connection through that server, which basically adds a step in the path of your data packets. If someone traced your IP address, theyd see it as the proxy servers IP address instead of your home address [13]. The big difference, is that proxy servers are not anonymous and the logs of the traffic are not confidential. Thats a point of onion routing: each node adds encryption to the packet and then we have multiple layers of encryption just like an onion.

There are three important elements in a TOR network that we should understand:

- Input node - the entry point to the network. The input nodes are selected from those that work for a long time and have proven to be stable and high.
- Exit Nodes - gateways of the network where it is possible to see unencrypted traffic although it isn't possible to determine the source.
- Middle relays - Middle relays are the ones who add speed and robustness to TOR network. They advertise their presence to the rest of the network and they just have the information of its neighbors. So, if we think of some malicious user in the network, he wouldnt know the source IP address. All TOR traffic passes through at least three relays (default) before it reach is destination.
- Bridges - Entry points to the network, which are not listed publicly. The bridges are solution to the relays problems because when a TOR client starts up, it needs a way to fetch a list of all the entry, middle and exit relays available. This list is not secret, so, the relays can be blocked or even the user can be blocked. Bridges are a clever solution to this problem. At their core, bridges are just unpublished entry relays. Users that are behind censored networks can use bridges as a way to access the Tor network.
- Directory Authorities - Powerful relays that keep a list of bridges and relays. Trusted volunteers of the TOR network have the power of maintaining the status of the entire network, are known as DA's. They are distributed around the world and are in charge of updating the master list of TOR relays, choosing if a relay is valid and

publishing it in a document called consensus, accessible by all relays and clients. This document is updated every hour by a vote with some process [18]:

- Each DA compiles a list of all known relays.
- Each DA then computes the other needed data, such as relay flags, bandwidth weights, and more.
- DA then submits this data as a status-vote to all the other authorities
- After that, each DA will go get any other votes that are missing from the other authorities.
- All the parameters, relay information, etc. from each vote are combined or computed and then signed by each DA.
- This signature is then posted to the other DAs.
- There should be a majority of the DAs that agree on the data, validating the new consensus.
- The consensus is then published by each DA.

III. TOR Performance

Anonymous communication is full of surprises and TOR itself, proposes a system that faces some performance difficulties. Tor works on real-world Internet and requires no special privileges or kernel modifications, requiring only little synchronization or coordination between nodes and providing a reasonable trade-off between anonymity, usability and efficiency [3]. The mechanism behind Tor is such that the client, when accessing the Tor network, will fetch info from a certain DA and create a three hop circuit made based on the relays that are in that list. After that, symmetric keys are exchanged with Diffie-Hellman mechanism: client sends a CREATE packet to the first relay. This relay finishes the key exchange and sends a CREATED cell. Cell size is 512bytes. The client uses this secure connection with the first relay to extend the circuit to a second relay, sending a RELAY EXTEND cell. This also contains informations for a DH key exchange. The second relay sends a CREATED cell back. This process is repeated until the last relay of the circuit is reached.

When the client has three keys from three circuit nodes, encryption of the message will be made with three layers of encryption, that will be decrypted through the communication by each node. The last node will deliver client's message to the server. Tor works with TCP streams. By default, a circuit is used for 10 minutes and keeps being used until the stream associated to it closes.

Basically, anonymous communication is provided and, if we use HTTPS on server connection, we ensure data encryption in the last hop (last node - server). If that isn't done, that connection can be a sniffed and it can compromise data while still not revealing our identity, being difficult to track back. Our main goal, will be attend to that characteristic of the TOR network, so we will be focusing on analyzing it's overhead.

IV. Experiment Setup

A. TOR Public Network

. In this project, we ran some measurements in the public TOR network, in order to understand the behavior of the network, and, most of all, try to identify if there was some pattern in the results. The importance of this step is such that, we have to gather some information and verify if theoretical problems that we read about, really happen.

The approach for these measurements was simple because, since the network is public we don't have an amount of liberty degrees to change some parameters. So, we use a very important one, we increase the length of the circuit, meaning, as a running client, we increase the number of hops and we generate some web access.

How we do that? First, we download the source code and we read the manual for perceiving the parameters that we needed to change, in order to make that happen. Then, we use a web service that is running in our classroom, netlab.fe.up.pt, which can be accessed from the outside network.

B. TOR Network Implementation on Netlab

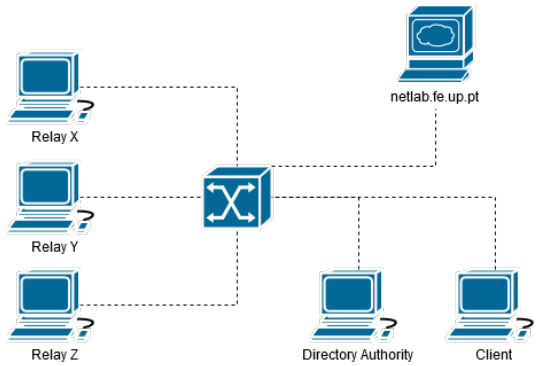

Fig. 1. Network diagram

It's not possible to characterize TOR network using measurements from the public network, because it contains nodes we can't control and might lead to bad conclusions. Therefore, the implementation of a private and controlled network was the solution. To accomplish that, we used Netlab's machines to create the network, by using 23 of those machines, meaning a network with 23 relays. Since, the web page netlab.fe.up.pt has the possibility to be accessed from the outside and also from inside, it was chosen to run measurements.

In order to generate website accesses and downloads and measure their times, we created a python script, that logs results into a text file. The amount of iterations run was 100 for each measurement made. First configuration to be changed, on TOR's source code, will be the number of circuit hops (circuit length) which is identified by ROUTE DEFAULT LENGTH. Secondly, Bandwidth measurements will be performed by limiting the bandwidth of each relay with the variables that are related to that - Bandwidth Rate and Bandwidth Burst. The first one is nothing more, nothing less a token bucket which limits the average incoming bandwidth usage on this node to the specified number of bytes per second and the average outgoing bandwidth usage to that same value. According to TOR's manual, each relay needs at the very least 75 KBytes of Bandwidth, the recommended value is 250 KBytes and the

default is 1Gbyte. The second one, Bandwidth Burst, limits the maximum token bucket size (also known as the burst) to the given number of bytes in each direction and the default is also 1GByte. In order to quickly setup the network on the lab's environment, we created some bash scripts that compile TOR from source code on several machines, through SSH connection, which allowed us to change any parameter of the network in all machines, without having to do it manually. To plot results into a CDF chart, we also made use of a python script, including the appropriate libraries.

C. Other approaches

After a long search, we came across some solutions that we could use to implement our network, that were Docker and Shadow. They are different tools. Lets understand what they are and why we dont use them.

1) Docker: Docker is an open standards platform for developing, packaging and running distributed applications. Docker simplifies the application development and executing by packaging all the required software for application including the dependencies into a single software called a Docker image that may be run on any platform and environment [12].

So, docker is a virtualization alternative where the host machine kernel is shared with the virtualized machine or the operation software, which uses some specific tools, like Docker compose for defining and running multi-container Docker applications and Docker Swarm that provides native clustering functionality for Docker containers.

Based on this, Docker seems a fantastic tool in order to provide more nodes in the network. For that, we create some container, then we pull an Ubuntu distribution and we run TOR inside as a different relay as the machine where the container was. But, there were some disadvantages for us to use Docker: first of all, by using different machines, although they are in the same network and the communication is pretty fast, its less instant communication then a machine-container communication. Second, since we create bash scripts to provide an effective way to run the network, its not very linear and easy to manage the same script for adding docker containers. Basically, it is a fantastic tool but in terms of our conditions, it will become difficult and not so manageable in order to have an automatic way of creating a TOR network.

2) Shadow: Shadow is a unique discrete-event network simulator that runs real applications like Tor and Bitcoin, and distributed systems of thousands of nodes on a single machine. Shadow combines the accuracy of emulation with the efficiency and control of simulation, achieving the best of both approaches. This tool creates an isolated simulation environment where virtual hosts can communicate and it is possible to control the experiments by changing network topology, latency or even bandwidth [4].

It also uses plugins, that are shared libraries linked to real applications. Shadow dynamically loads these libraries to natively execute the application code. For this project in particular, the plugin that is useful is shadow-plugin-tor. As the name implies, it provides a simulation of the Tor anonymity network by wrapping the Tor source code with the necessary hooks that allow it to communicate with the Shadow simulator, thereby leveraging Shadow's unique functionality to allow rapid prototyping and experimentation of Tor. It contains scripts that assist in analyzing results, generating Tor topologies, and running experiments using the generated topologies.

After a proper setup of Shadow, guiding by the tutorial provided in GitHub, we were able to run an experiment and edit the XML files provided, where network topology is specified, as well as network properties such as link latency, jitter and packet loss rates.

We run some experiments but our goal is not to validate this tool, it is to characterize the network and the results could be helpful or not to provide some term of comparison.

V. Results Analysis

A. Public Network

TOR's public network is an enigma because it isn't possible to know and control each node's resources, meaning that each node's responsiveness might be affected when dealing with a lot of traffic or with low bandwidth available. As it was already mentioned, 100 measurements were made. Theoretically, increasing the number of nodes in a circuit will increase delay in website loading, which is what happens practically - 7 hops circuit is more than 4 seconds higher than a 3 hops circuit.

In the 1MB download case, it's different because only a single request is made, where in websites several get requests are performed.

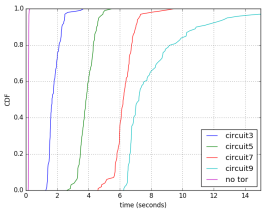

Fig. 2. Netlab.fe.up.pt access

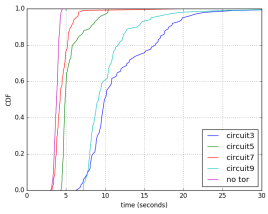

Fig. 3. Download 1Mb file

Our expectation was a similar behavior between those 2. Strangely, it is possible to see that circuit length 3 performs worst than circuit length 9, while circuit length 7 gets the best result. As we mention before, the explanation is related with the bandwidth available in each node, so, probably in the 3 hop circuit there wasn't enough bandwidth to produce a good result while in the 7 hop circuit all nodes had good amount of bandwidth.

B. Private Network

In terms of results obtained in the lab's environment, the behavior shows that higher circuit length takes more time to download. Graphics presented in this section show a comparison between each circuit length when using tor and when not using tor, as well as different bandwidth settings (250KByte or

1GByte - default), so that we can see the cost for anonymous communication.

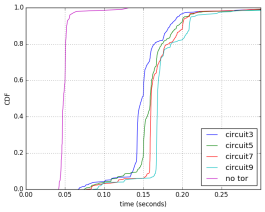

Fig. 4. Netlab 250KB bandwidth

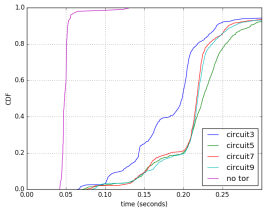

Fig. 5. Netlab default bandwidth

For the case of visiting Netlab website, results of different circuit length show a similar behavior. When accessing the website without TOR, the get request took approximately 0.05 seconds. It is possible to see on Fig.4) that almost 80% of values are near to the 0.15 second mark. As for Fig.5), it is possible to see that 5 hop circuit behaved worse than 7 hop and 9 hop.

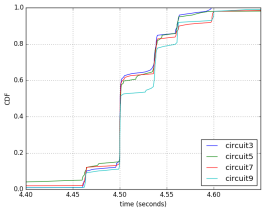

Fig. 6. 1MB Download 250KB bandwidth

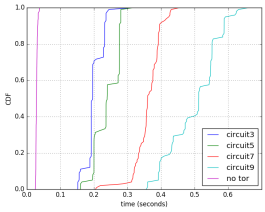

Fig. 7. 1MB Download default BW

For the case of downloading a 1MB file from Netlab website, results of different circuit length show a similar behavior when it comes to 250KByte Bandwidth. When accessing the website without TOR, download took approximately 0.03 seconds. It is possible to see on Fig.6) that almost 90% of values are within 4.50-4.55 second mark and there is not a visible difference when bandwidth is limited. As for Fig.7), it is possible to see that, as expected, higher circuit length also results in higher delay, with a visible difference in this case, which can be acceptable for user experience but within applications, it can be a big amount of delay between using 3 hop circuit or using 9 hop circuit.

VI. Final considerations

Increasing the number of circuit nodes will not bring any advantages in terms of security and thats because with more nodes, even if we have a lot more encryption layers and, in tracking terms, it is more difficult to find the source, at the same time it will reflect in more probability in finding an intruder node on our connection, being less reliable and more failures might occur. So, all in all, increasing the number of hops will not profit you much and may harm your anonymity. It seems that the three hops circuits initially picked from the authors of Tor is the wise choice for anonymity and performance.

In terms of throughput, it is logical that, with three nodes, we have more throughput and less delay. So, based on our measurements, if we want to have less possible delay while still guaranteeing anonimity, three hops is the right choice for website accessing. Bandwidth is a curious parameter because, as our measurements show, we can have less available bandwidth without having an substantial impact on the delay, thus being an important factor.

References

[1] Fallon Chen and Joseph Pasquale *Toward Improving Path Selection in Tor*, available in http://cseweb.ucsd.edu/ pasquale/Papers/globecom10c.pdf

[2] TOR Overview, available in https://www.torproject.org/about/overview.html.en

[3] Roger Dingledine, Nick Mathewson, Paul Syverson *Tor: The Second-Generation Onion Router*, available in https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf

[4] Shadow Simulator Wiki, available in https://github.com/shadow/shadow-plugin-tor/wiki

[5] TOR Project FAQ, available in https://www.torproject.org/docs/faq.html.en

[6] TOR Project Manual, available in https://www.torproject.org/docs/tor-manual.html.en

[7] TOR Relay Confiuration, available in https://www.torproject.org/docs/tor-relay-debian.html.en

[8] TOR Onion Services, available in https://www.torproject.org/docs/tor-onion-service.html.en

[9] Nyx Project, available in https://nyx.torproject.org/

[10] TOR Anonymity Network, available in https://en.wikipedia.org/wiki/Tor(anonymitynetwork)

[11] Ian Paul, Contributor, PCWorld *How to use the Tor Browser to surf the web anonymously* available in https://www.pcworld.com/article/2686467/privacy/how-to-use-the-tor-browser-to-surf-the-web-anonymously.html

[12] Vohra, Deepak *Pro Docker* 2016: Apress.

[13] Joel Lee, MakeUseOf *What is Tor? A beginner's guide to the privacy tool* available in https://www.makeuseof.com/tag/what-is-onion-routing-exactly-makeuseof-explains/

[14] Robin Snader and Nikita Borisov, IEEE Members *Improving Security and Performance in the Tor Network through Tunable Path Selection* available in http://hatswitch.org/ nikita/papers/tuneup-tdsc.pdf

[15] Andriy Panchenko, Fabian Lanze, and Thomas Engel, Interdisciplinary Centre for Security, Reliability and Trust (SnT) - University of Luxembourg *Improving Performance and Anonymity in the Tor Network* available in https://lorre.uni.lu/ andriy/papers/ipccc12-tor-performance.pdf

[16] Jordan Wright Blog *How Tor Works - Part One* available in https://jordan-wright.com/blog/2015/02/28/how-tor-works-part-one/

[17] Jordan Wright Blog *How Tor Works - Part Two* available in https://jordan-wright.com/blog/2015/05/09/how-tor-works-part-two-relays-vs-bridges/

[18] Jordan Wright Blog *How Tor Works - Part Three* available in https://jordan-wright.com/blog/2015/05/14/how-tor-works-part-three-the-consensus

[19] Tom Lika, Tom Sochor and Hana Sochorov - University of Ostrava *Comparison between normal and TOR- Anonymized Web Client Traffic* available in http://www.sciencedirect.com/science/article/pii/S1877042810022998

[20] Sadegh Momeni Milajerdi and Mehdi Kharrazi - Department of Computer Engineering, Sharif University of Technology, Tehran, Iran *A composite-metric based path selection technique for the Tor anonymity network* available in https://www.sciencedirect.com/science/article/pii/S0164121215000035

BitCoin- Performance characterization

Diana Almeida
Faculdade de Engenharia da
Universidade do Porto
Email: up201502835@fe.up.pt

Pedro Rodrigues
Faculdade de Engenharia da
Universidade do Porto
Email: up201104253@fe.up.pt

***Abstract*—Bitcoin brought to light a concept already know for several decades and implemented worldwide. The blockchain is a concept that is currently revolutionizing not only the financial status of the world but also several other implementations arise with it (case of the distribution of the electricity [1]. The concept is simple, a peer-to-peer network solution that validates the changes to itself thus making it more reliable since, in theory, it would need to control several nodes to make a dent in the blockchain.**

This paper tries to test the scalability of the blockchain and to access if the real-world applications could handle the growth of the blockchain and the several transactions that occur per minute to it.

I. Introduction

Bitcoin is a digital currency and decentralized, meaning that Bitcoin do not have a central authority controls the system. Nodes regulate all operations, in other words, Bitcoin uses a peer-to-peer network architecture.

Each user has a wallet that can be a web, hardware, *app* or paper. In the wallet you can see the transactions list, send or receive and view the balance. The wallet has a public and private key necessary to make a transactions. The users can generate the amount of public key they want. The balance depends on previous transactions to verify that the balances user want to send do not exceed the balances user.

Whenever there is a transaction a message is sent to all nodes with the account address of the sender and the receiver and the amount sent. The transactions list is update in each node and this list contains the user name and balance. To verify that the transaction is valid is required a digital signature. The signature is created through the message of the transaction and the private key of the sender, thus, the signature is renewed in each transaction. The private key is used to verify if the signature matches the user. As in each transactions the signature depends on the message, nobody can reuse the message without confirm the signature. The mathematical models behind this process are: *mathematical trap door* and *elliptic curve digital signature algorithm*.

As a user can make many transaction in a short space of time, all nodes can not be update at the same time. Therefore, the nodes will have different transactions list. For the transfer to be confirmed, the nodes are voting to see what transaction has been made first. The transaction list that have the most votes from the nodes is the one that will be update in all nodes.

As it is possible to see all the transactions carried out, the Bitcoin system orders the transactions in the block chain, where each block has the reference to the previous block. The transactions that are in same block are considered to have been made at the same time and the transactions that are not in a block are transactions that are not confirmed. It is necessary several confirmations to consider the transaction done. To validate a new block the Bitcoin system uses cryptographic hash SHA256.

To prevent double-spending blockchain uses proof of work. This means that an attacker needs equivalent computing power to all computational power spent from that point in time to the present. Furthermore, the attacker would have to overcome the legitimate Bitcoin network.

II. Related developed

In order for us to understand how Bitcoin and the blockchain works we first read some official documentation a proceeded to build a testing environment and analyze the results. Our main testing objectives were:

- How to connect to the Bitcoin network (Problems and barriers)
- How information is exchange both in a P2P and on a collaborative basis (pool)
- Mining and how hard is the computation and the gain from it
- How can Bitcoin be used in a day-to-day basis

A. Setting up and understanding the protocol

First of all we set up a few nodes on the lab. Since the Bitcoin code is available as open-source on Git hub we proceed to download it and install on a bench on the lab. This bench was composed by three machines, every single one of them with 4 GB of RAM and a Dual-core processor (with hyper-threading). We neglected the Hard drive usage.

In the repository there is an option were we can quickly develop our own testing network. This is to facilitate the test of new features.

We quickly understood that we wouldn't see much using three PCs on the lab. Our solution was to create several docker instances and measure the parameters that we wanted to analyses.

This was not the optimal solution since the computer was a shared resource and quickly the disk prove to be the bottleneck of the simulation. Bitcoin uses several seeks in order to obtain information from the blockchain and process it and since

we were using HDD and not SSD the bottleneck made the experiment difficult for several instances.

B. Mining Bitcoin on a public poll

We also tried to mine in a pool to understand what were the implications both in the network and in the computer. We quickly realize that Bitcoin mining without specialized hardware isn't lucrative. This is due to the difficulty imposed by Bitcoin [4] that states that a transaction should take about 10 minutes to be processed, therefore and due to the reason that the network is constantly changing the difficulty changes. Not only the confirmations have an increase of difficulty but also mining, in order to keep the value of the coin.

We tested several pools but the result was approximately the same. Pool mining was a concept that derive for the need of adding a new block to the blockchain. Since rewards are given for those who mine said block there was a computations disparity between those who had large datacenters and those who had a small computer. To even the odds pools are created to accommodate people that use weaker computational resources to a large one. It's basically distributed computing where the pool gives order to slaves and slaves calculate, when a slave reaches a result it then uploads to the pool. The pool then tries to add the result to the blockchain dividing the profits regarding the computational resources that each node had used. The big difference is how this computational power/reward calculation is achieved. Each user send shares to a challenge made by the pool. This share is a small subset of the bigger problem to solve. There are mainly five approaches to prevent cheating and reward users:

- Slush
- Pay-per-Share
- Full Pay-per-Share
- Eligius
- P2Pool approach

The Slush approach follows a score where older shares have less weight than newer shares this prevents users using two pools and, when the challenge is equal for both pools, to steal a share from one pool and inject it into another trying to do a race condition. This is done by each round, so other rounds will not affect newer rounds).

The Pay-per-share (PPS) method is offers an instant payout for each valid share that the user send. This will cause each user to be rewarded by who share the most shares. This method offers the least variance and puts the responsibility solely on the poll operator (if the difficulty lowers, the poll if doesn't adapt may lose money)

The Full Pay-per-share is an optimization of the PPS that benefits from the hight transaction fees. By verifying transactions that are made with a certain amount of fees calculated by the poll it can leverage those high fees to reward the user more BTC than with the previous model.

Eligius is a strategy that benefits users with stale shares. Stale share is a share that although valid failed was submitted late and in the meantime another user had already solved it. Stale shares rewards are kept to be rewarded at later date. Rewards are also payd with a high value of BTC, this is to avoid high transaction fees. Some polls also delay the transaction fees to avoid those high fees making the user to be rewarded with an higher amount.

P2Pool rewards users with the most recent shares. By each block generation the users with most recent shares are rewarded equally. If the user has high shares, it will most likely receive an higher amount.

This information is important not only to try to understand what attacks could be done, not only on the blockchain but also on the poll itself (manipulating the reward system to create more profit).

C. Attacks to the network

We realize that the attack described on the first method could be done on the majority of polls although its hard to implement without hearing the wire (since the information isn't shared across the nodes, we just receive a "Stratum sent a new workload" message,Stratum being the poll, similar to NTP System [3]). Imagine the following scenario. A big entity, collects information about what comes on the wire for monitoring reasons, if there are people mining on different polls, they can delay their traffic and answer the poll challenge thus creating stale shares for the affected user, or they can simply change the issuing user to their users, keeping the user in the dark and not receiving and confirmation about the submitted hashes.

The PPS schemes suffer from a race condition where nodes compete to submit the answer. The poll tries to distribute compute but keep a redundancy in order to complete the process. This is not optimal and redundant work is done. If, by any reason, the node is slow down it will fail to commit the result and not get any earnings. The same goes for the P2Pool approach, however, this one won't get any earnings if the share takes too long (since it doesn't credit old shares).

All these attacks are based on race conditions however we noted that several polls use insecure connections prone to snooping and that would allow for people to change the authority of those shares. There are some polls that implement TLS based connection but not as many.

We also tried other crypto-currency (alt-coins) like Ethereum and Litecoin but we saw a very similar behavior. Ethereum looked like it gave the most payout for the energy but its difficult to mine. It requires a graphics card with at least 3GB of VRAM making it a somewhat large investment.

D. Mounting a simulation with several nodes

After gaining some insight of what can be done with Bitcoin we tried to perform some purchases in the real network. We noticed that for some payment systems requires more transactions confirms than the others. This is to prevent a situation where when a person make a buy and the transaction didn't make it to the blockchain it will be dropped meaning that a user will keep the goods and the money. This confirmations took around 30 minutes to complete. This is a too much time for a day-to-day use. Going to the supermarket and waiting 30

minutes to know if the payment is done or if the buyer needs to repeat the transaction.

We focused our work on this issue. To assess if Bitcoin could make it to the mainline payment systems and use for a everyday transaction. The scalability problem is hard to test, so we use several benches on the laboratory and configure similar computer to perform distributed computing. The computers were connected to the laboratory network through fast Ethernet switches and each bench was connected with Gigabit Ethernet. This architecture won't affect results since the simulation is done in a simulated environment. This architecture if only to connect nodes of the cluster to perform the calculations.

We decided to use NS3 [5] and the Bitcoin module [6]). Sparing the details of compiling both codes lets summarize that every machine has the same code compiled with OpenMPI to be able to use the power of several computers to conclude the simulations in a smaller time window (the time is also simulated on the experiment).

We did tests varying some specific parameter in order to understand how the network adapts. the parameter that we testes were:

- Number of Miners
- Number of Nodes
- Size of each block

For the variance of miners we use the following parameters 100 generated blocks, 100 nodes and block size of 0.128 kB (kilobytes).

For the variance of Nodes we use 16 miners and 100 blocks generated with the size of 0.128 kB (kilobytes).

For the variance of size block we use 16 Miners and 100 blocks generated.

Our tests are time based meaning that we want to understand how the performance of the network is affected variating a parameter.

We want to bring to attention that the Miners are also Nodes of the network although nodes can verify the data being sent to them, miners will create new blocks with transactions. So, for a node to mine, there needs to be nodes to confirm that the new block follows the rules of the Bitcoin protocol. This is the strength of the blockchain, several nodes confirm the answers of other nodes, if a new block is rejected the node will send a message and notify that the block was rejected.

This bring us to other attack that could occur. The 51% attack states that, if a malicious entity controls over than half the Bitcoin network, it can reject certain transactions or try to manipulate the blockchain. The Bitcoin network relies on Peer-to-Peer (P2P) to keep the blockchain in a state of integrity. If a malicious person wants to attack the blockchain at a certain point of time, it needs to keep the same computational power since the blockchain inception till that point in time and then the following transactions.

III. Solution

The solution for this problem is rather problematic. Since the protocol is so widespread and with a general poll of 14. 000. 000 TH/s its difficult to implement a solution without causing a disruption on the service. Whoever this could be achieved with a new standard (new coin derived from the blockchain) due to the differences on the protocol a hard fork will need to be set in. This means that older versions of software won't recognize the new standard. Bitcoin cash emerged from it. On August 1st, 2017 Bitcoin cash was the standard the was created to solve some problems but mainly the problem of scalability. Since no major services implement it yet, we couldn't test it and re-creating the simulation would take some time (include code on NS3 and test it properly).

IV. Evaluation

In the first test we increase the number of nodes:

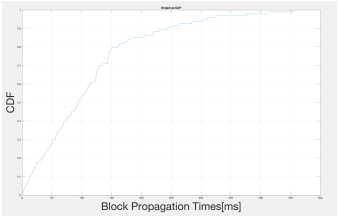

Fig. 1. Block Propagation Times with 100 nodes

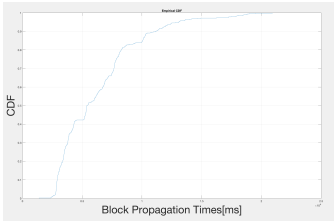

Fig. 2. Block Propagation Times with 500 nodes

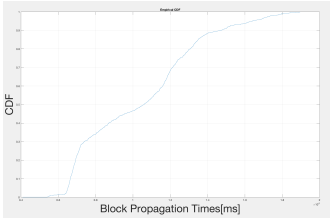

Fig. 3. Block Propagation Times with 1000 nodes

As we can see with increase the number of nodes, the total time of block propagation increase but time of each block propagation decrease. Since an increase of nodes there is an increment in number of peers on the network that block needs to be confirmed.

In the second test we increase the number of miners:

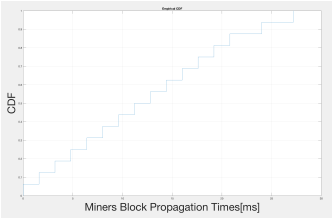

Fig. 4. Miners block Propagation Times with 100 nodes and 16 miners

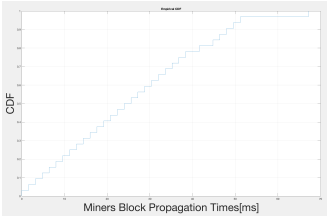

Fig. 5. Miners block Propagation Times with 100 nodes and 32 miners

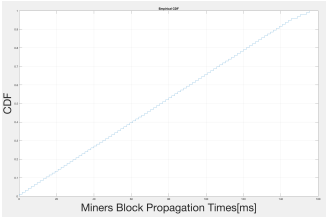

Fig. 6. Miners block Propagation Times with 100 nodes and 96 miners

We can see with the increase of the number of miners, the total time of the block propagation increase because the block needs to be confirmed by all miners and consequently the time for transactions increase.

In the third test we increase the block size:

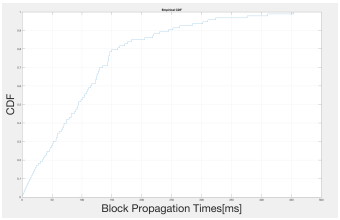

Fig. 7. Block Propagation Times with 100 nodes and block size 0.128KB

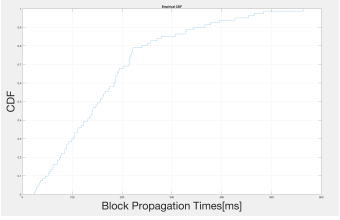

Fig. 8. Block Propagation Times with 100 nodes and block size 3KB

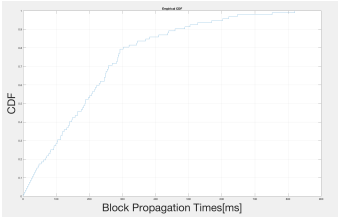

Fig. 9. Block Propagation Times with 100 nodes and block size 4KB

In the figures 7, 8, 9 we can observe that with the increase of block size the time of the block propagation decreases because the blocks can take more transactions and results in faster confirmation time.

V. Conclusion

We see that Bitcoin has a scalability problem. With even more nodes and even more transactions the network can't simply be used for day at day transactions. This is due to the network being slow to converge that's the reason some payment systems only need to have 2 confirmations for the payment to be set as done. However this doesn't guarantee that the payment has been done. Our conclusions were supported by a recent fork in the Bitcoin blockchain where a new coin has emerge, Bitcoin cash. This coin emerged on the August 1st, 2017 to address the scaling issue of Bitcoin and it hopes that will solve the issue. The coin is now being implemented. In order to keep users the amount of money that the users had on August 1st will be the same amount as the amount of Bitcoin cash. This was an hard fork witch means that older software won't recognize the change. This is important since it can affect the coin value where systems need to change to accommodate the fork. In essence we believe that our work proves one of the coins problems and should be addressed on the new fork to ensure that the coin is stable and can be used worldwide.

References

[1] Opray, Max. Could a Blockchain-Based Electricity Network Change the Energy Market? The Guardian, Guardian News and Media, 12 July 2017, www.theguardian.com/sustainable-business/2017/jul/13/could-a-blockchain-based-electricity-network-change-the-energy-market.
[2] "Storing bitcoins", http://en.bitcoin.it/wiki/Storing_bitcoins(Online: Accessed on 22. December 2017)
[3] NTP Server Stratum Levels Explained. Edited by NTP Server blog , NTP Server, 15 Sept. 2008, http://ntpserver.wordpress.com/2008/09/10/ntp-server-stratum-levels-explained.
[4] Bitcoin Difficulty and Hashrate Chart. Edited by BitcoinWisdom.com , BitcoinWisdom, 22 Dec. 2017, bitcoinwisdom.com/bitcoin/difficulty.
[5] Ns-3. Edited by NS3 , ns3 RSS, 22 Dec. 2017, http://www.nsnam.org/.
[6] Gervais, Arthur. Arthurgervais/Bitcoin-Simulator. GitHub, 13 Oct. 2016, https://github.com/arthurgervais/Bitcoin-Simulator.
[7] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". https://bitcoin.org/bitcoin.pdf (Online: Accessed on 22. December 2017)

Impact of network threat detection engines

Ricardo Arajo
School of Electrical and
Electronics and Computer Engineering
Faculdade de Engenharia e Universidade do Porto
Porto, Portugal 4200–465
Email: up201304917@fe.up.pt

***Abstract*—As the Internet becomes available to a large number of people and businesses emerge from it, securing it becomes both a chalenge and a necessity. Any compromise on the system could cost you million of clients but securing it must not affect the services overall performance. Over the past few years Intrusion Detection Systems (IDS) such as Suricata and Snort have been developed with the porpuse of detecting any sort of attack a malicious individual might seek to perform from traffic patterns. This paper explores how IDS may have impact on things such as response delay in the network or the CPU and Memory usage in the machine where the intrusion detection is installed on.**

I. Introduction

With the growth of the internet it has become impossible to provide a large scale and secure service from a singular location so various hardwares are placed on differents points of the network with the purpose of detecting and preventing malicious content from reaching the host.

Detection of intrusion exists in various forms, such as anti-virus software, which protecting important host system files, and network threat detection engines such as Suricata which monitors network traffic coming from and going to specific points of the network. These engines match the traffic against a certain database of patterns and ,if any of them match, a warning is given. If the user choses to do so they can also be configured to serve as Intrusion Prevention Systems which perform a given action when a recognized pattern is found.

II. Tests

To perform tests on a IDS a network was built consiting of 3 main components: the attackers which send malicious traffic to detect, the system that does the intrusion detection using Suricata and the vulnerable web server which implements using the already developed web application called DVWA. The attackers in this experiement consists of one computer that will establish 100 different connections with the web server and send up to 100 requests. Each connection did only one specific attack but during the experiment differents combinations of parallel attacks were tried to better see their individual impact and what synergy they would have when using suricata. For this experiment two different scenarios were also created in order to have more data on the subjet.

Scenario 1:

In the first scenario the IDS is on a standalone system in a spefic point of the network where a portion of the network traffic goes through. The attacker is situated somewhere in that part of the network and performs a combinations of attacks consisting of SQL injection, Cross-Site Scripting(Reflected) and Command Injection.

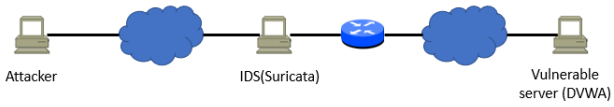

Fig. 1. Setup

Scenario 2:

In the second scenario the IDS is tested on the system where the webserver is in as well. This serves as way to test how well Suricata performs with other concurrent processes that are also CPU intensive (in this case the webserver). The attacks done are the same as in scenario one.

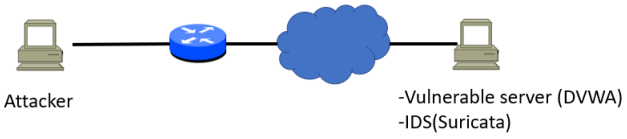

Fig. 2. Setup2

III. Evaluation

In the following experiments related to scenario 1 and 2 respectively it was extracted the requested response delay with the various combinations of attacks the the cpu usage related to Suricata. In the graphs related to the delay the line in blue represents the experiment without Suricata and the line in red represents the experiment with Suricata.

Exp1:

In the first scenario there wast much difference in delay with and without Suricata, however you can still see that the CDF related to the SQL injection shows some slow down with Suricata in comparison to the other attacks as it is slightly shifted to the left. The combination of SQL injection and XSS also shows some difference most likely as a result of the SQL injection.

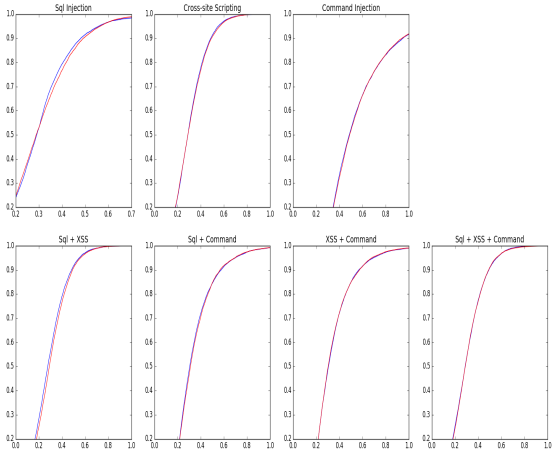

Fig. 3. Exp1-Cumulative distribution function of the delay (seconds)

The next figure shows the cpusage in throughtout the entire experiment. With the exception of the command injection attacks it is impossible to diferentiate between the diferent attacks. This results from the fact that the command injection request packages have a slower response time and therefor require less processing power from Suricata.

s

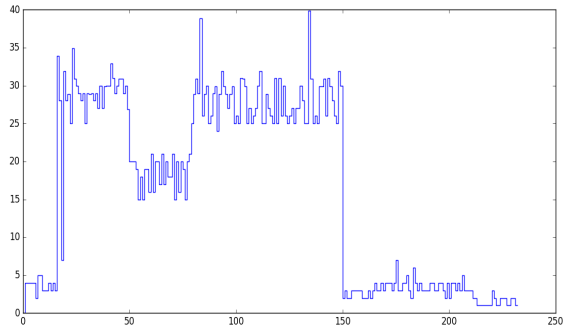

Fig. 4. Exp1-Cpu usage

Exp2:

In the second experiment the differences in delay noted in the first are now more visible but now it is possible to see differences in the command injection attack as well. This is explained by the fact that Suricata uses less cpu (10 to 20 % instead of 15 to 30%) as shown in figure 6 and causes more delay as a result of the slower package processing. The XSS however is almost unaffected most likely as result of not needing as much processing as SQL injection attacks which had the same cpu usage but higher delay.

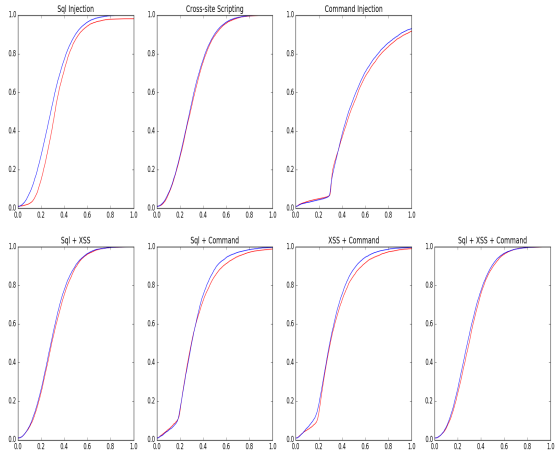

Fig. 5. Exp2-Cumulative distribution function of the delay (seconds)

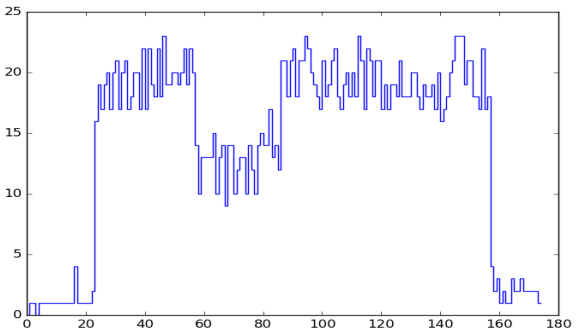

Fig. 6. Exp2-Cpu usage

IV. Conclusion

In both experiments conducted it was possible to see that Suricata reacted to different traffic differently especially in the cases related to SQL injection. This results from the fact that each type of traffic has its own list of rules and patterns that Suricata has to compare and some attacks are easir to identify than another. In the case of the XSS because it was easy for Suricata to identify it even a drop in available cpu did not cause any notable delay but the same can not be said for the other two that needed more pattern checking to be done.

VPN benchmarking

Joo Polnia
SSRE
Faculdade de Engenharia
da Universidade do Porto
Email: joaovpsilva@hotmail.com

Luis Barbosa
SSRE
Faculdade de Engenharia
da Universidade do Porto
Email: up201607769@fe.up.pt

***Abstract*—Virtual Private Network (VPN) became one of the most reliable technologies to provide data security - protection, confidentiality, integrity, data origin authentication, replay protection and access control. This paper attempts to provide a common sense definition of a VPN, an overview of different approaches to building them and set the best configuration for each test environment.**

I. Introduction

VPN stands for "Virtual Private Network", the goal of this paper is to compare two of the most common open source VPN and preform benchmarking test using benchmarking tool for VPN and Internet speed. For this purpose the OpenVPN and StrongSwan was chosen. The performance tests were made by changing some of the configuration of the VPN such as the length of the encrypting key or the type of configuration itself, changing the network conditions or the CPU usage in the VPN server.

II. Related work

A. What is a VPN

A VPN is a communications environment in which access is controlled to permit peer connections only within a defined community of interest, and is constructed through some form of partitioning of a common underlying communications medium, where this underlying communications medium provides services to the network on a non-exclusive basis[1]. We can abstract this formal definition to more simpler one: a VPN is a private network constructed within a public network infrastructure, such as the global network.

There are 3 main types of VPN:

- PPTP-protocol based VPNs
- IPSec-protocol based VPNs
- SSL-based VPNs - OpenVPN

PPTP - Point-to-Point Tunneling Protocol is one of the oldest VPN protocols developed by Microsoft and Ascend in 1999.Nowadays, the PPTP protocol is considered fundamentally insecure, as the strength of the security of the connection is directly related to the strength of the authentication mechanism chosen (for example, the password). Thus, an insecure password leads to an insecure VPN connection.

In this paper we will only benchmark the other two types of VPN since we believe the PPTP is outdated and should not be used.

B. SSL-based VPN - OpenVPN

The most commonly used VPNs nowadays are SSL-based VPNs, which are based on the SSL/TLS protocol.OpenVPN is often called an SSL-based VPN, as it uses the SSL/TLS protocol to secure the connection. However, OpenVPN also uses HMAC in combination with a digest (or hashing) algorithm for ensuring the integrity of the packets delivered. It can be configured to use pre-shared keys as well as X.509 certificates.

C. Internet Protocol Security

Internet Protocol Security (IPsec) is a network protocol suite that authenticates and encrypts the packets of data sent over a network, it provides confidentiality, data integrity, access control, and data source authentication to IP datagrams.

IPsec includes protocols for establishing mutual authentication between agents at the beginning of the session and negotiation of cryptographic keys to use during the session. IPsec can protect data flows between a pair of hosts (host-to-host), between a pair of security gateways (network-to-network), or between a security gateway and a host (network-to-host).[2] Internet Protocol security (IPsec) uses cryptographic security services to protect communications over Internet Protocol (IP) networks. [3]

III. Solution

A. OpenVPN

The network architecture to benchmark the OpenVPN implementation is the following:

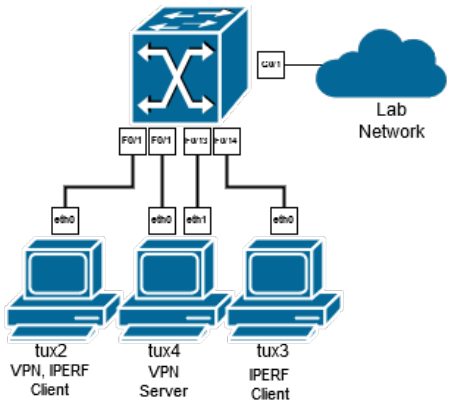

Fig. 1. Security Gateway to Security Gateway Tunnel

The VPN client will connect to the VPN Server and we will measure how much is difference in throughput, latency and overhead by changing the key size, the ip protocols(TCP vs UPD) and the cpu load in server.

In VPN server host we will configure the certificates for the server and for the client using OpenVPN-server script. We will also configure an Iperf server in another host so we can measure the difference when connecting with and without OpenVPN.

B. StrongSwan

StrongSwan is an OpenSource IPsec implementation, it implements both the IKEv1 and IKEv2. The IKE is is a component of IPsec used for performing mutual authentication and establishing and maintaining Security Associations (SAs). [4] IKE is used to negotiate ESP or AH SAs in a number of different scenarios, each with its own special requirements, in our configuration we used:

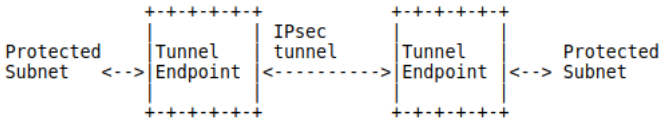

Fig. 2. Network Architecture for OpenVPN benchmark

The left side is called *MOON* and the right side is called *SUN*, for each one we run some configurations:

```
sudo ./00_config.sh
sudo ./01_secrets.sh
ipsec reload
ipsec up host-host
ipsec status
```

The first script run and install the *strongswan* software and copy the configuration into the configurations file:

```
sudo nano /etc/ipsec.conf
```

This file can be found on the attachements VII-B and VII-A

The second script set the shared secrets. Strongswan allows *RSA authentication* or *PSK authentication*. For this report we use a pair of *pre-shared-keys (PSK)* as we can see at VII-C, the file must me stored at */etc/ipsec.secrets*. To set other kind of authentication we should change the file as we see at VII-C, where *"/etc/ipsec.d/private/key.pem"* is the path to the private key for each tux.

The command *ipsec reload* was run to update the settings and then we "turn *up*" the *host-host* configuration, on the image3:

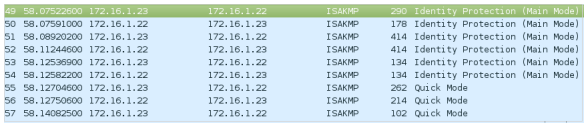

Fig. 3. Ipsec wireshark logs

We can check the logs of the connection been established. The connection can be confirmed with the *ipsec status command*, image:4:

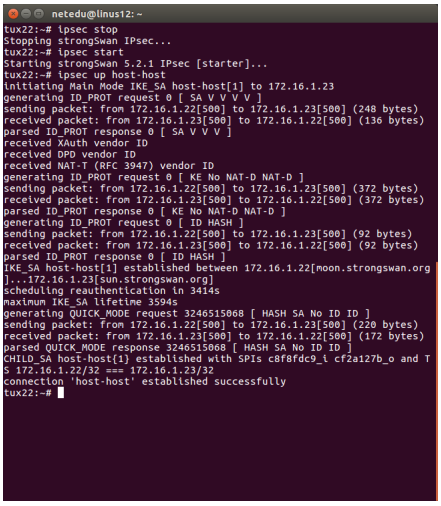

Fig. 4. Ipsec status

IV. Evaluation

A. OpenVPN

The first test was to measure the overhead from the OpenVPN protocol, so we transfered file with scp from tux2 to tux3:

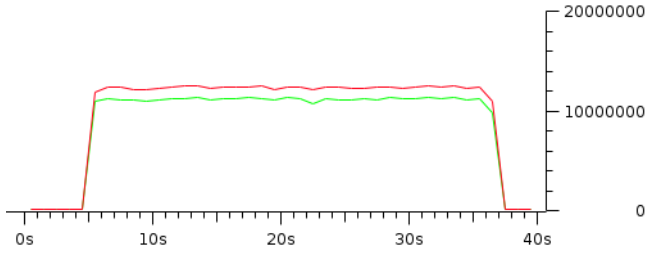

Fig. 5. OpenVPN Throughput(RED) vs SSH Troughput(GREEN) - Key size: 2048bits

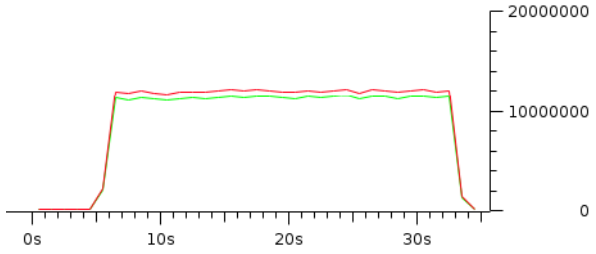

Fig. 6. OpenVPN Throughput(RED) vs SSH Troughput(GREEN) - Key size: 1024bits

The overhead can be seen as the difference between the lines.

The second test is for latency:

Key-Size	Latency
no VPN	0.471 ms
1024 bits	0.845 ms
2048 bits	0.850 ms
4096 bits	0.863 ms

Fig. 7. Latency Test

Finally we will test the effect of the CPU load in the OpenVPN server:

No Clients	Server CPU %
0	3%
1	9-15%
5	50-65%
+7	100%

Fig. 8. Server CPU load while clients are transferring files

B. StrongSwan

After set and confirm all configuration some test were preformed using some tools: ping, iperf tests and stress. As we can confirm by comparing both ping The *ipsec* connection makes the connection slow down:

```
tux23:/etc# iperf -c 172.16.1.22 -u -b 100m -t 60
------------------------------------------------------------
Client connecting to 172.16.1.22, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 172.16.1.23 port 57237 connected with 172.16.1.22 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.0 sec   685 MBytes  95.7 Mbits/sec
[  3] Sent 488351 datagrams
[  3] Server Report:
[  3]  0.0-60.0 sec   685 MBytes  95.7 Mbits/sec   0.006 ms    0/488350 (0%)
[  3]  0.0-60.0 sec  1 datagrams received out-of-order
```

Fig. 9. Iperf transmit connection between Sun and Moon

```
tux23:/etc# iperf -c 172.16.1.22 -u -b 100m -t 60
------------------------------------------------------------
Client connecting to 172.16.1.22, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 172.16.1.23 port 40998 connected with 172.16.1.22 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.0 sec   635 MBytes  88.8 Mbits/sec
[  3] Sent 452967 datagrams
[  3] Server Report:
[  3]  0.0-60.0 sec   635 MBytes  88.8 Mbits/sec   0.375 ms    0/452966 (0%)
[  3]  0.0-60.0 sec  1 datagrams received out-of-order
```

Fig. 10. Iperf transmit connection between Sun and Moon with Ipsec

The *ref* in time show the time of the resquest/reply of a ping.

The analysis of the *iperf* test we can see the interference of the ipsec connection on the bandwidth: It shows clearly the

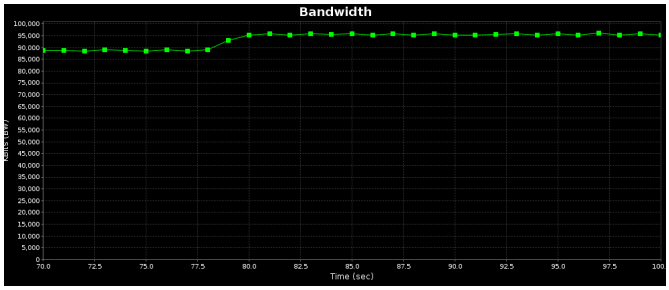

Fig. 11. Jperf transition from Ipsec to normal connection

bandwidth transition effect when ipsec connection stop.

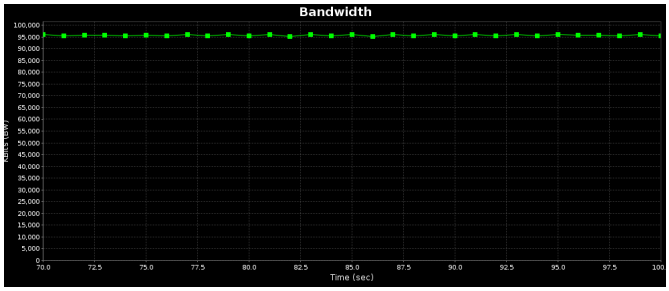

Fig. 12. Jperf test with normal connection

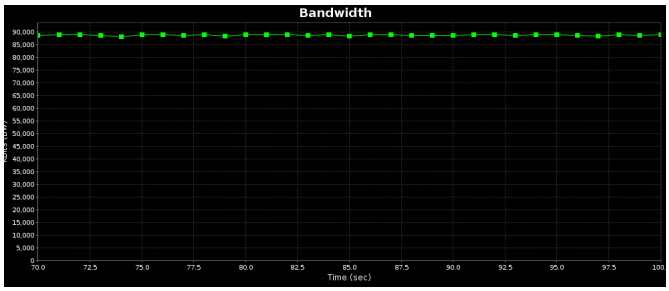

Fig. 13. Jperf test with Ipsec connection

To run this test we use the jperf, a graphical software of iperf.

V. Conclusion

OpenVPN and StronSwan are both very secure VPN implementations.

We observed that key size of the certificates in the OpenVPN protocol doesn't degrade too much the connection and gives double or triple security in exchange only the process of generating a big key, for example 4096bits, can be bit slow depending on the hardware of the server.

The OpenVPN implementation also doesn't consume too much resources in the server for one client, however it doesn't scale very well as it can reach almost 100% with more than 10 simultaneous connections. This is because for each client the server must process encryption/decryption of the packets.

One solution could be setting up multiple OpenVPN server make sure the clients are using them (e.g. through DNS round-robin), we can configure a dynamic routing protocol (e.g. RIP due to its simplicity) and our infrastructure would be capable of supporting an arbitrary number of clients as long as we have enough hardware.

StronSwan implements an Internet Protocol Security (IPsec) connection, IPsec can automatically secure applications at the IP layer. We observed the bandwidth is affected by this implementation, reducing it for about 8%, it means strongSwan shouldn't be used we the bandwidth connection is too small, and the speed connection decreases twice with the ping test and it's possible download 50Mbytes more without the ipsec connectio. However the security advantage makes it suitable for a vpn implementation. Also to refer that we just tested the basic PSK implementation, it's expected to find a worst result with RSA authentication.

References

[1] Paul Ferguson and Geoff Huston, *What is a VPN?*, Revision 1. April 1998.

[2] Kent, S. and Atkinson, R. *IP Encapsulating Security Payload (ESP)* IETF. doi:10.17487/RFC2406. RFC 2406.

[3] Wikipedia: IPsec, `https://en.wikipedia.org/wiki/IPsec`

[4] INTERNET STANDARD, `https://tools.ietf.org/html/rfc7296` C. Kaufman, P. Hoffman *Internet Key Exchange Protocol Version 2 (IKEv2)* Microsoft Standards Track

VI. Anexos: OpenVPN

A. Configuration file for OpenVPN server

```
#!/bin/bash
#Install OpenVPN
apt-get update
apt-get install openvpn easy-rsa

#Configure OpenVPN
gunzip -c /usr/share/doc/openvpn/examples/sample-config-files/server.conf.gz >
  /etc/openvpn/server.conf
cp server.conf /etc/openvpn/server.conf

#Enable Packet Forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward

#Install and Configure ufw
apt-get install ufw
ufw allow ssh
ufw allow 1194/udp
cp ufw /etc/default/ufw
cp before.rules /etc/ufw/before.rules

#Configure and Build the Certificate Authority
cp -r /usr/share/easy-rsa/ /etc/openvpn
mkdir /etc/openvpn/easy-rsa/keys
cp vars /etc/openvpn/easy-rsa/vars
openssl dhparam -out /etc/openvpn/dh2048.pem 2048
cd /etc/openvpn/easy-rsa
. ./vars
./clean-all
./build-ca

#Generate a Certificate and Key for the Server
./build-key-server server

#Move the Server Certificates and Keys
cp /etc/openvpn/easy-rsa/keys/{server.crt,server.key,ca.crt} /etc/openvpn
service openvpn start

#Generate Certificates and Keys for Clients
./build-key client1
cp /usr/share/doc/openvpn/examples/sample-config-files/client.conf
   /etc/openvpn/easy-rsa/keys/client.ovpn
gedit /etc/openvpn/easy-rsa/keys/client.ovpn

#Creating a Unified OpenVPN Profile for Client Devices
echo '<ca>' >> /etc/openvpn/easy-rsa/keys/client.ovpn
cat /etc/openvpn/ca.crt >> /etc/openvpn/easy-rsa/keys/client.ovpn
echo '</ca>' >> /etc/openvpn/easy-rsa/keys/client.ovpn
echo '<cert>' >> /etc/openvpn/easy-rsa/keys/client.ovpn
cat /etc/openvpn/easy-rsa/keys/client1.crt >> /etc/openvpn/easy-rsa/keys/client.ovpn
echo '</cert>' >> /etc/openvpn/easy-rsa/keys/client.ovpn
echo '<key>' >> /etc/openvpn/easy-rsa/keys/client.ovpn
cat /etc/openvpn/easy-rsa/keys/client1.key >> /etc/openvpn/easy-rsa/keys/client.ovpn
echo '</key>' >> /etc/openvpn/easy-rsa/keys/client.ovpn
```

VII. Anexos: StrongSwan

A. Configuration file for strongSwan: SUN

```
config setup

conn %default
        ikelifetime=60m
        keylife=20m
        rekeymargin=3m
        keyingtries=1
        keyexchange=ikev1
        authby=secret

conn host-host
        left=172.16.1.11
        #leftsubnet=172.16.10.0/24
        leftid=@sun.strongswan.org
        leftfirewall=yes
        right=172.16.1.23
        #rightsubnet=172.16.50.0/24
        rightid=@moon.strongswan.org
        auto=add
```

B. Configuration file for strongSwan: MOON

```
config setup

conn %default
        ikelifetime=60m
        keylife=20m
        rekeymargin=3m
        keyingtries=1
        keyexchange=ikev1
        authby=secret

conn host-host
        left=172.16.1.23 #192.168.0.1
        #leftsubnet=172.16.50.0/24 #10.1.0.0/16
        leftid=@moon.strongswan.org
        leftfirewall=yes
        right=172.16.1.11
        #rightsubnet=172.16.10.0/24
        rightid=@sun.strongswan.org
        auto=add
```

C. Secrets file strongSwan

Secrets for the *PSK* implementation:

```
@moon.strongswan.org @sun.strongswan.org : PSK 0sv+NkxY9LLZvwj4qCC2o/gGrWDF2d21jL
```

Secrets for the *RSA* implementation:

```
@moon.strongswan.org @sun.strongswan.org : RSA "/etc/ipsec.d/private/vpn-server-key.pem"
```

Authentication
Architecture review, latency, and scalability

Felipe Bahamonde
Faculdade de Engenharia
da Universidade do Porto
Mestrado Integrado em Engenharia
Electrotcnica e de Computadores
Rua Dr. Roberto Frias, 4200-465 Porto
Email: up201701163@fe.up.pt

Luis Silva
Faculdade de Engenharia
da Universidade do Porto
Mestrado Integrado em Engenharia
Electrotcnica e de Computadores
Rua Dr. Roberto Frias, 4200-465 Porto
Email: ee10095@fe.up.pt

***Abstract*—This paper tries to shed some light in the current state of Single Sign On with the scope being directed to architecture, latency and scalability. To test a SSO system we created one that would simulate a working Authentication service using Gluu Server. We came across some errors that could be mitigated in the future and as such our results are merely a numeric example. We believe that if we'd set up the identity server closer to our target web page and had better hardware as our test bench we would get results that would emulate reality very closely.**

Keywords
Single Sign On, Security, Authentication, Gluu, Segurana em Sistemas e Redes, SSRE

I. Introduction

In the fast world that we live on, performance and security are a must at online ventures. Furthermore, with more and more transactions and exchange of important information being handled online, security is of the utmost importance. As we can see Software as a Service (SaaS) became the online standard for the whole world. Now, instead of building one enormous app, companies are moving towards smaller apps that compliment each others services. In light of this development, companies began to use a SaaS application that, as an example, deals with the payrolls, another that deals with employees/clients information, and so on and so forth. This method prevents, in case of an attack, that the attackers got a hold of all the information of the company. Thus, having a SaaS vendor whose sole purpose is to authenticate users and maintain the security of this information is vital and useful to most companies. To improve security there has been some evolving in the authentication format in the past years. We went from *user-name and password*, to *social authentication*, in recent times to *passwordless*, and now and in the near future *Multifactor Authentication*. In order to know more about this current topics we chose this *Authentication* work project (1).

This work follows the curricular unit of Segurana em Sistemas e Redes (SSRE). It's a class that addresses issues such as encryption, security and security engineering, and so, for this work, we assign this class vision to a single sign on (SSO) authentication system. In order to set up a SSO system we used a Gluu Server as the base software. Gluu Server is an open source identity and access management platform. Being open source, it uses software that was composed by Gluu and software incorporated from other open source projects (2).

In conclusion, for this work we'll be focusing in the security and performance aspect in a single sign-on procedure, testing with different users and different number of users at the same time. Well hope to see how does Gluu perform with different workloads.

II. Related work

There are a few papers about Single Sign On but we think we should talk about one that caught our eyes. It's called *"Adoption of Single Sign On and Multifactor Authentication in organizations - A critical evaluation using TOE framework"*. Their conclusions covered a variety of subjects of which we reaffirm that finding a solution that is both secure and convenient is very difficult and that it is an ongoing task. Other conclusion that they get is that there still isn't a lot of traction to SSO in companies due to the complexity of their existent systems. Changes would affect dramatically the work-flow of companies, although change will be almost inevitable (3).

III. Solution

In order to test and analyze the SSO systems, it was proposed to create a configuration consisting of a basic client, in this case a web application that would allow log in and log out, and an authentication server that would allow authorization and authentication to the application(s) that would work with this SSO system, following a scheme similar to the one shown in image 1 (client server).

To achieve this, an open source server called Gluu Server was used, which allows, among other things, to implement SSO systems without major difficulty, for this, it is decided to use the OpenID protocol, which allows to work easily through RESTFUL APIs, and on the side client side it was decided to create a basic page with authentication, which will be registered in the server in order to be able to enter using

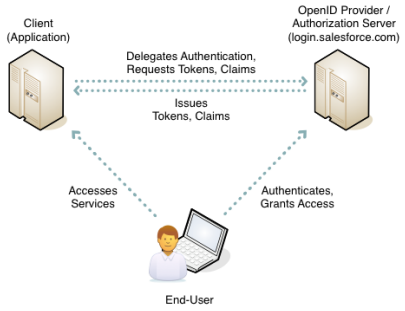

Fig. 1: SSO scheme.

SSO.

With this idea in mind we proceeded to implement the solution.

In the first place, the Gluu Server was installed and configured, when encountering problems during the installation in the computers of the laboratory, we decided to continue in the cloud through Google Cloud, in a server in the Netherlands. Then, we continued with the deployment of the authentication website, for this, a demo page specially developed for SS0 is used. This web application is built on a virtual machine that will run on our computers. To complete the installation, the Gluu plugin "oxd server" is installed in the same virtual machine, which allows the registration of the application (in this case the web page) quickly in the Gluu Server.

With all this installed, you get a configuration similar to the one seen in the image 2.

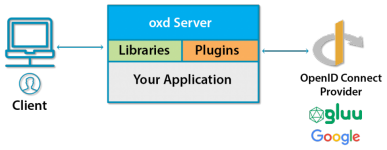

Fig. 2: Main configuration.

To proceed with the registration of the application on our server, we used -as already mentioned- oxd server which allows the dynamic registration of the application, where only the following fields should be modified:

The main parameters here are:

- Host: where the page is, in this case is our localhost.
- Port: The port to connect with the Gluu plugin oxd server.
- Op_host: the URL of the Identity Server, in our case is gluu.server.lab3.
- authorization_redirect_uri: the page where we will be redirected after the authorization on the Identity Server.

*demosite.cfg

File Edit Search Options Help

```
[oxd]
host = localhost
port = 8099
op_host = https://gluu.server.lab3

[client]
authorization_redirect_uri = https://client.example.com:8080/login_callback/
post_logout_redirect_uri = https://client.example.com:8080/post_logout/
client_frontchannel_logout_uris = https://client.example.com:8080/logout_callback/
client_name = oxdpython Flask Example App
```

Fig. 3: oxd configuration.

- post_logout_redirect_uri: Page to do the logout request.
- Client_frontchannel_logout_uris: the page where we will be redirected after the logout.
- Client_name: Easy name to recognize the application in the identity server.

Once the configuration is finished, we proceed to upload the test users in the system, for this, a random user base is created in Excel, in the image 4 it is possible to see an example of the created users.

Username	First Name	Last Name	Email	Password
BVYDKZ	JAMES	SMITH	AYKBDPZ@fe.up.pt	SQGAIB
ABLADS	JOHN	JOHNSON	XKUGPOM@fe.up.pt	JMAAPG
DPLCGW	ROBERT	WILLIAMS	EQRMCUG@fe.up.pt	NNHJTM
ABOSLT	MICHAEL	JONES	IPXFTWI@fe.up.pt	FWFUCQ
TUKDRC	WILLIAM	BROWN	JVQDEMT@fe.up.pt	KXCCWF
SCJBTR	DAVID	DAVIS	NNLTLPZ@fe.up.pt	LLFHHP
UDZOLB	RICHARD	MILLER	PMWVYWI@fe.up.pt	UPJSLJ
CAFAHP	CHARLES	WILSON	QMFSMUW@fe.up.pt	WXMZEC
XLNZQA	JOSEPH	MOORE	SILOEWS@fe.up.pt	GTSXAM
JXRCAK	THOMAS	TAYLOR	YRWWTQC@fe.up.pt	MCSBJI
UYAHGM	CHRISTOPHER	ANDERSON	HPYOEQK@fe.up.pt	UMXOGA
SNMRFL	DANIEL	THOMAS	KEALOKX@fe.up.pt	UQIIJF
MUALLV	PAUL	JACKSON	SKYFLXL@fe.up.pt	DBEYKZ
WNJEQF	MARK	WHITE	PZZVUIB@fe.up.pt	LYNFAP
AHNAJW	DONALD	HARRIS	AWCFXWK@fe.up.pt	KFXUYS
EZDJCJ	GEORGE	MARTIN	DQGZJWO@fe.up.pt	ZLOPKJ
UEESMD	KENNETH	THOMPSON	HTXEZQD@fe.up.pt	QSRUKX
ZKPEZN	STEVEN	GARCIA	AQTDSWP@fe.up.pt	BIOGWX

Fig. 4: Random users.

Then, through the "import User" option in Gluu, users are imported. Here is important to talk about the difficulties encountered in correctly uploading users, since Gluu Server has not yet implemented this tool correctly, so when trying to enter them, users were not activated (disabled to be used), which is why had to manually activate each of the thousand users.

With the user database ready, we proceed to the test stage. Here, 2 alternatives were considered, in the first place a user was simulated by browsing the website and making the log in with the selenium tool. In the following script it is possible to see part of the code. While this first approach managed to automate and test with different users, we could not correctly measure the execution time, this is because with selenium, the code of the HTML page is loaded with images, JavaScript, etc. Which generates a greater delay. So this option was discarded.

```
from selenium.common.exceptions
import NoSuchElementException
from selenium import webdriver
```

```
import datetime
import pandas
from pandas import ExcelFile
import xlrd
from pandas import ExcelWriter
import os
import oxdpython
from flask import Flask, render_template
, redirect, request, make_response
app = Flask(__name__)
this_dir=os.path.dirname(os.path.realpath(__file__))
config = os.path.join(this_dir, 'demosite.cfg')
oxc = oxdpython.Client(config)
xl = pandas.read_excel("userdata.xls")
FORMAT = ['Username', 'First Name', 'Last Name',
'Email','Password','User Status']
useres = xl['Username']
passwords = xl['Password']
driver = webdriver.PhantomJS(service_args=[
'--ignore-ssl-errors=true',
'--ssl-protocol=any'])
driver.set_window_size(1400, 1200)
timevector=[]
for i in xl.index:
    auth_url = oxc.get_authorization_url()
    driver.get(auth_url)
    print ('User no. '+str(i)+': '+ useres[i])
    try:
      driver.find_element_by_id('loginForm:username')
      .send_keys(useres[i])
      driver.find_element_by_id('loginForm:password')
      .send_keys(passwords[i])
      driver.find_element_by_id
       ("loginForm:loginButton").click()
      a = datetime.datetime.now()
      try:
          driver.find_element_by_id
          ("authorizeForm:allowButton").click()
      except NoSuchElementException:
          pass
      b = datetime.datetime.now()
      tem=(b-a)
      timevector.append(int(tem.total_seconds()*1000))
      try:
          continue_link = driver.
          find_element_by_link_text('Logout')
          .click()
      except NoSuchElementException:
          pass
    except NoSuchElementException:
        pass
print('Average time:
'+str(sum(timevector)/float(len(timevector)))+
" Milliseconds")
driver.close()
```

A second approach was simply to make get/post requests and to measure the time between sending and responding using python code and then executing the code in parallel with a bash script. In the following script it is possible to see the code used to make the requests and the added payload (important because it is how OpenID works).

```
import datetime
import pandas
from pandas import ExcelFile
import xlrd
from pandas import ExcelWriter
import os
import oxdpython
from flask import Flask, render_template,
redirect, request, make_response
import requests
import random
app = Flask(__name__)

this_dir=os.path.dirname(os.path.realpath(__file__))
config = os.path.join(this_dir, 'demosite.cfg')
oxc = oxdpython.Client(config)
xl = pandas.read_excel("userdata.xls")
FORMAT = ['Username', 'First Name', 'Last Name'
,'Email','Password','User Status']
useres = xl['Username']
passwords = xl['Password']
timevector=[]

for i in xl.index:
    s = requests.Session()
    auth_url = oxc.get_authorization_url()
    print("\n Auth URL", auth_url)
    print("\n")
    login = s.get(auth_url,verify=False)
    print("\n Login URL", login.url)
    print("\n")
    id=random.randint(0,150)
    payload = {'loginForm':     'loginForm',
    'loginForm:loginButton'     :'Login',
        'loginForm:username' :useres[id],
              'loginForm:password' : passwords[id],
              'javax.faces.ViewState': 'stateless'}
    a = datetime.datetime.now()
    loginpost = s.post(login.url,
    data=payload,verify=False)
    b = datetime.datetime.now()
    temp=(b-a)
    print temporal
    timevector.append(int(temp.total_seconds()*1000))
    print("\nLogin Success\n", loginpost.url)
    print("\n")
    logout=s.get(oxc.get_logout_uri(),verify=False)
    print("\nLogout Success", logout.url)
```

```
    print("\n")
    s.cookies.clear()
print(str(sum(timevector)/float(len(timevector))))
```

IV. Evaluation

To evaluate the performance, it was thought to generate requests in parallel with different number of users, from 1 user to 1000. Unfortunately, this could not be achieved, due to the fact that a bottleneck was generated in our virtual machine due to the limited hardware resources, the VM could not handle that amount of requests at the same time, so the test was performed from 1 to 200 users, which was the maximum supported.

In the following figure it is possible to see the results obtained for: 1 5 10 50 100 200 users. where we have the amount of users at same time and the average time of responds in milliseconds, also we have the standard deviation with the size of the circle .Here it is possible to see that for a user the average time was close to 2 seconds and that for 200 users the average time was 70 seconds. Which is much more than expected.

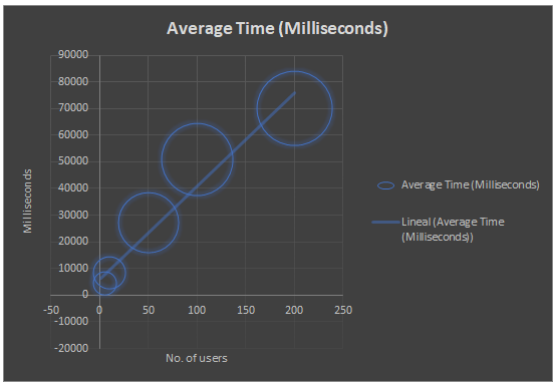

Fig. 5: User test results.

It is also possible to notice a tendency to stability, it is expected that having put more users the trend would have been clearer.

Here it is important to highlight the errors made and the reason for the results: First, placing the Identity server away from the web page was an error, since it adds delay due to distance and nodes in between. A second error is the implementation of the application in a virtual machine, because with that few resources it is not possible to handle such a number of requests at the same time.

V. Conclusion

In conclusion we accomplished what we set out to be our goal. In such we understood how a Single Sign On system is deployed and how it's architecture works. Also, with our mistakes during our implementation, we understood how to evolve and do a better job next time we're facing this challenges. Finally, we acknowledged that a hardware minimum requirements study should have been done prior to starting this project and so we urge all people working on this same subject to implement that same study prior.

References

[1] Ado Kukic, *The Definitive Guide to Single Sign On (SSO)*, Auth0.

[2] Gluu Server Overview `https://www.gluu.org/gluu-server/overview/`

[3] Marise Marie, Michael Lane, *The Adoption of Single Sign-On and Multifactor Authentication in organizations - A critical evaluation using TOE framework*, Issues inf Informing Science and Information Technology.

Architecture of the Signal Protocol

Joana Veiga
Faculdade de Engenharia
da Universidade do Porto
Email: joana.veiga@fe.up.pt

João Francisco
Faculdade de Engenharia
da Universidade do Porto
Email: joao.francisco@fe.up.pt

***Abstract*—O Signal é um protocolo seguro de comunicação, encriptando mensagens, chamadas, vídeochamadas, e outros, entre dois utilizadores ou um grupo. O seu código é livre, podendo a sua segurança ser validada por terceiros. Este trabalho resulta de uma análise à arquitetura do Signal, e tenta traçar o caminho das mensagens trocadas para obter um conhecimento mais aprofundado das características de segurança avançadas deste protocolo. Foi estudado a registo de um dispositivo (Secção IV), o envio de uma mensagem (Secção V), e a receção de uma mensagem (Secção VI).**

I. Introdução

O protocolo Signal implementa encriptação ponta-a-ponta, garantindo uma maior proteção aos seus utilizadores. As comunicações que transitam na rede são protegidas por encriptação não só quando se encontram em trânsito, mas também nos servidores que fornecem o serviço. Desta forma, apenas o destinatário terá novamente acesso à informação em *plain-text*, impedindo a intromissão de qualquer intermediário. A sua sofisticação e aprovação por parte da comunidade de segurança levaram a que, para além da aplicação própria Signal, este protocolo fosse também implementado em aplicações tão populares como o Whatsapp, Facebook Messenger ou Google Allo (estes dois últimos apenas em modo *incognito*).

Fig. 1. Verificação de *fingerprint* de chaves públicas.

O protocolo combina o algoritmo *"Double Ratchet"* [1], *PreKeys* (chaves pré-geradas), e um *extended triple Diffie-Hellman handshake* (X3DH) [2] para garantir as propriedades de *confidentiality, integrity, authentication, participant consistency, destination validation, forward secrecy, backward secrecy/future secrecy, causality preservation, message unlinkability, message repudiation, participation repudiation,* e *asynchronicity* [3]. Para prevenção de ataques *man-in-the-middle*, é possível confirmar a *fingerprint* da chave pública (exemplo na Figura 1) de um determinado dispositivo, comparando-a com a que é obtida no nosso dispositivo. Se forem idênticas, é garantida a identidade do destinatário.

II. Trabalho Relacionado

Uma análise aprofundada ao Signal, focada nas suas valências relacionadas com segurança, foi realizada por Cohn-Gordon, Katriel et al. em *"A Formal Security Analysis of the Signal Messaging Protocol"* [4]. Neste estudo, é evidenciada a falta de escrutínio a que o protocolo foi sujeito, não tendo passado por validação por parte de académicos, o que mesmo assim não impediu a sua implementação em aplicações populares.

Após a verificação do *Key Agreement* e do algoritmo *Double Ratchet* presentes no Signal, os autores concluíram que o protocolo é seguro e não apresenta falhas graves que possam comprometer as promessas de segurança.

III. Desafio

Inicialmente, foi proposto desenvolver uma solução baseada no protocolo Signal, implementando clientes e um servidor no Laboratório de Redes da FEUP (NetLab). No entanto, após uma primeira análise à documentação disponível, verificou-se que esta tarefa seria mais complexa do que inicialmente pensado. Três razões levaram ao abandono desta ideia inicial:

- O código fonte de todos os componentes, embora disponível *online*, não se encontra extensivamente documentado. O principal objetivo com o *open-sourcing* do projeto prende-se principalmente com a possibilidade de verificação do código por parte de terceiros, que podem confirmar que as aplicações que utilizam fazem o que afirmam fazer e que o protocolo é seguro, e não tanto para permitir o desenvolvimento de soluções baseadas no protocolo. Logo, embora possível, encontra-se pouco preparado para esse fim.

- O projeto está pensado para o uso em ambientes móveis (Android e iOS), dependendo de aplicações construídas para

essas plataformas. Usa também um número de telemóvel como identificador do utilizador, razão pela qual as aplicações mais populares não estão disponíveis para equipamentos sem cartão SIM, como *tablets*. Logo, seria necessário um grande trabalho de adaptação do código para permitir a sua instalação nas máquinas do NetLab.

- Para um correto funcionamento de todas as funcionalidades, seria necessário contratar alguns serviços, não acessíveis para um trabalho desta natureza. Esses serviços são Twilio (para envio de mensagens SMS de verificação), Amazon Web Services (para armazenamento de anexos), e contas de desenvolvimento Google e Apple (para envio de notificações *push* aos clientes).

Por essas razões, foi decidido substituir o desenvolvimento prático de uma solução por uma análise teórica ao protocolo, através da análise do código disponível. Foi estudado o código dos projetos "libsignal-service-java", "libsignal-protocol-java", "Signal-Server", e "signal-cli", um cliente Signal para a linha de comandos. Os resultados são apresentados de seguida.

IV. Registo de um Dispositivo

Quando pretendemos registar um novo dispositivo, é necessário criar uma nova identidade. É gerado um *IdentityKeyPair*, com chave pública e privada do tipo Curve25519 (*Elliptic Curve Diffie-Hellman*). De seguida, é obtido um *registrationId* gerado com uma *hash function* SHA-1 geradora de números pseudo-aleatórios.

Após este passo, o dispositivo é registado no servidor, com uma *password* gerada aleatoriamente. É pedida a geração e envio de um código de confirmação por SMS ao servidor, para garantir que o número registado pertence ao utilizador.

Após a verificação do código, são geradas *PreKeys* de uso único, também do tipo Curve25519, sendo uma delas assinada com a *IdentityKey* gerada na criação da identidade. Por fim, a *IdentityKey* pública, as *PreKeys*, e a *PreKey* assinada do cliente são transmitidas para o servidor, onde são armazenadas.

V. Envio de uma Mensagem

No envio de uma mensagem, o processo é iniciado com o pedido das *PreKeys* do destinatário. Depois, são obtidas as *MessageKeys* a partir das chaves de saída das cadeias KDF (*Key Derivation Function*) que formam a *ChainKey* do remetente. É obtida a *RatchetKey* do remetente, que é atualizada a cada nova mensagem recebida, e inserida no *header* da mensagem a enviar. Caso ainda não tenha sido recebida nenhuma *RatchetKey* do destinatário, a inicialização é feita com a *PreKey* assinada do destinatário (disponível no servidor).

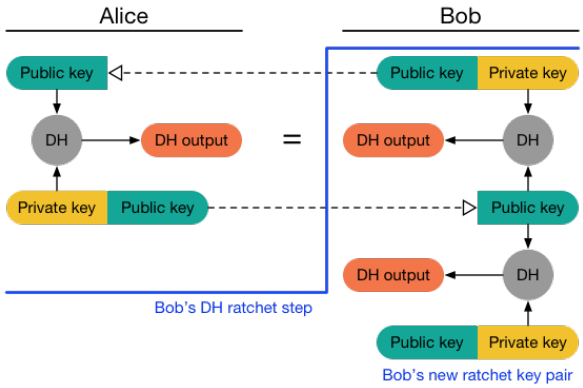

Fig. 2. Atualização da RatchetKey. [1, p. 7]

O conteúdo da mensagem é encriptado, recorrendo às *MessageKeys*, e a mensagem completa é construída. A *ChainKey* do remetente é atualizada, e a mensagem é enviada ao servidor.

VI. Receção de uma Mensagem

Quando o cliente recebe uma mensagem, esta tem de ser desencriptada. A *RatchetKey* e *ChainKey* do remetente são recuperadas, e a partir da *ChainKey* obtidas as *MessageKeys*. Das *MessageKeys* é obtida a chave MAC *(Message Authentication Code)*, que é verificada para confirmar que a mensagem vem do remetente esperado e não foi alterada em trânsito.

Utilizando a *CipherKey*, obtida através das *MessageKeys*, o conteúdo da mensagem é desencriptado e o *plain-text* recuperado, tornando a mensagem legível.

VII. Conclusão

A elaboração deste estudo foi bastante desafiante. Inicialmente, devido às dificuldades mencionadas na Secção III e, depois, devido à complexidade do protocolo. O Signal envolve mais de dez tipos diferentes de chaves e um complexo processo de atualização, que leva a várias "cadeias" de chaves relacionadas [4, p. 2]. Seguir o código fonte implica percorrer um elevado número de ficheiros, com muita informação não comentada.

A documentação disponível no site do projeto Signal é bastante completa, mas também muito complexa e detalhada. A elevada sofisticação dos processos implementados no protocolo garantem segurança e privacidade ao utilizador final, mas torna difícil a sua compreensão completa. Por essa razão, este relatório pode apresentar pouco rigor. No entanto, como projeto para estudo, permitiu alguma familiarização com as boas práticas de segurança de um protocolo avançado, utilizado por milhões de pessoas diariamente.

Conclui-se que seria vantajoso produzir mais documentação

relativa ao protocolo Signal, permitindo a sua mais fácil verificação e validação pelas partes interessadas, e a implementação de soluções próprias, construídas à medida das necessidades de novos grupos de utilizadores. Numa sociedade constantemente ameaçada por novos ataques informáticos, em que as comunicações e outros dados sensíveis que atravessam a Internet se tornam cada vez mais apetecíveis para potenciais atacantes, torna-se essencial garantir que estes se encontram protegidos, fora da vista de terceiros.

References

[1] Moxie Marlinspike and Trevor Perrin. The double ratchet algorithm. https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf, 2016.

[2] Moxie Marlinspike and Trevor Perrin. The x3dh key agreement protocol. https://signal.org/docs/specifications/x3dh/x3dh.pdf, 2016.

[3] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. Sok: secure messaging. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 232–249. IEEE, 2015.

[4] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. *IACR Cryptology ePrint Archive*, 2016:1013, 2016.

Overhead of network security protocols and viability for IoT

Joana Enes
Faculdade de Engenharia
Universidade do Porto
4200-465 Porto, Portugal
Email: up201305054@fe.up.pt

Jose Pintor
Faculdade de Engenharia
Universidade do Porto
4200-465 Porto, Portugal
Email: up201307994@fe.up.pt

***Abstract*—IoT (Internet of Things) is defined by the wireless network of devices embedded with sensors and actuators, connected with each other and to the internet. With the fast growth of IoT, the concern about security increases. However, because of the requirements of their nodes, such as the low energy consume or low processing power, IoT networks require customized solutions. This work explores the security of two protocols designed for IoT and M2M (machine to machine) communication, which are CoAP (Constrained Application Protocol) and MQTT (Message Queue Telemetry Transport). They both implement mechanisms to provide security, by using security communications protocols: in case of CoAP, DTLS (Datagram Transport Layer Security) and in case of MQTT, TLS. For each IoT protocol, there have been made tests with a server and a client for the different security modes they provide, and it was measured the overhead of the security protocols in the messages.**

I. Introduction

With the evolution of technology, we're at a point where every human is connected with each other through devices and the next big step relies on making a connection between everyday objects. That's where the Internet of Things comes in.

A *thing*, in the IoT world, can be anything from a person with a heart monitor implant to an automobile that has built-in sensors. It is anything that can be assigned an IP address and provided with the ability to transfer data over a network. To incorporate this, small and simple devices are developed so they can be placed anywhere.This technology has great potential of becoming the next era of informatics systems embedded into humans day-to-day lives and improve our comfort on every day's struggles. For such big potential, it's important to develop security in these IoT networks, because otherwise the system just wouldn't be reliable and trustworthy.

The idea of embedding small devices into everything requires a great development in resources management in such devices, such as memory, power and communication's bandwidth. So providing security has a cost in performance of such devices.

In this paper it's taken into consideration what's the overhead of providing security, i.e., how much more of information must a system provide and sacrifice for performance so it isn't vulnerable.

This paper is organized as follows: in section II it is given some background on IoT and security issues. Section III presents the two protocols studied on this work, their features and security mechanisms. Section IV illustrates the tests performed in order to analyze security on these protocols. In section V, the results are presented and it is made an analysis of the security mechanisms. At last, in section VI the conclusions taken by this study.

II. Background

Several protocols have been developed for IoT application. They can be implemented in any of the OSI layers, in order to meet the requirements of IoT networks. Below are listed some protocols for each OSI layer. A descryption on these can be seen in [1].

- *Physical/ Data-link:* IEEE 802.15.4, IEEE 802.11ah, WirelessHART, Bluetooth, Zigbee;
- *Network:* RPL, CORPL, CARP, 6LoWPAN, IPv6 over G.9959;
- *Session:* CoAP, MQTT, SMQTT, AMQP, DDS.

In [2] are presented and analyzed some session/ application protocols. In the next sections, we are going to focus on CoAP and MQTT.

Along with the protocols there have to be others which implement security. In [5], it is performed an analysis on security threats and vulnerabilities and are presented some mechanisms that improve security. In this study, it will be aborded DTLS, in case of CoAP, and TLS, in case of MQTT.

III. Session layer protocols

This section presents an overview of session layer protocols CoAP and MQTT.

A. Constrained Application Protocol (CoAP)

CoAP was developed as a web transfer protocol for use in constrained nodes and networks, relying on Representational State Transfer (REST) architecture, which means it is based on a client/server structure. It was designed in order to provide low overhead and multicast support. The main idea was to bring a light adaptation of HTTP optimized to IoT and M2M networks. It runs over UDP, providing its own mechanism of reliability, supporting confirmable messages, retransmissions,

duplicated packets detection and congestion control. It is divided in two sublayers: request/response layer and message layer. The first deals with the request and response codes carried on the messages, in order to prevent out of order, duplicated and lost packets. It contains four methods: GET, POST, PUT, DELETE. The second is responsible for the exchange of messages over UDP and the asynchronous nature of the interactions. It supports four types of messages: CON (confirmable), NON (non-confirmable), ACK (acknowledgement) and RST (reset) (see Fig. 1).

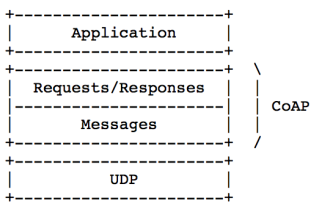

Fig. 1. CoAP layer model

CoAP uses a fixed-size 4 byte header in its messages, which are encoded in a simple binary format. The header has the following fields: Ver (CoAP version), T (Type of the message - CON, NON, ACK or RST), TKL (Token Length), Code (type of the message, e.g, request, success response, client error response or server error response) and Message ID. It is followed by a 0 to 8 bytes token, with the length indicated in the header. The token is used to correlate requests and responses. The token is followed by zero or more options. Options are followed by other options, the end of the message or the payload marker (0xFF) and the payload (see Fig. 2).

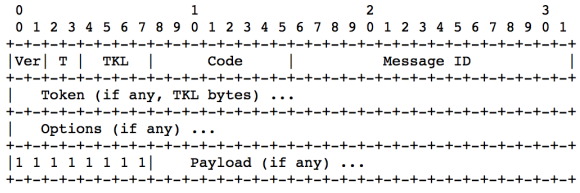

Fig. 2. CoAP messages format

A client can interact with the server by sending a request using of one of four methods: GET, POST, PUT or DELETE. These are similar to the HTTP ones.

1) Security in CoAP: In order to secure messages, CoAP requires an additional encrypted protocol as a cover, usually, DTLS. It supports four modes of security:

- *NoSec:* There is no security provided. Messages are transmitted in plain text.
- *PreSharedKey:* DTLS is enabled. Devices are preprogrammed with symmetric cryptographic keys. Each key includes a list of which nodes can use it.
- *RawPublicKey:* DTLS is enabled. Devices require authentication based on public keys. Each device is preprogrammed with an asymmetric key pair and a list of identities of the nodes it can communicate with.
- *Certificate:* DTLS is enabled. Each device has an asymmetric key pair with a X.509 certificate which is signed by a common trust root. The devices have also a list of root trust anchors that can be used to validate a certificate.

The three latest security modes allow authentication and authorization to proceed with the communication. Furthermore it is needed a cipher suite, in order to ensure confidentiality and integrity of the messages, which can be *TLS-PSK-WITH-AES- 128-CCM-8* for PreSharedKey, *TLS-PSK-WITH-AES- 128-CCM-8* supporting SHA-256 for RawPublicKey, and *TLS-PSK-WITH-AES- 128-CCM-8* which support for X.509 certificates for Certificate mode.

B. Message Queue Telemetry Transport (MQTT)

MQTT is an open source protocol designed for constrained devices and low-bandwidth, high-latency or unreliable networks. It is a publish/subscribe, extremely simple and lightweight messaging protocol, it helps minimize the resource requirements for the device, while also attempting to ensure reliability and some degree of assurance of delivery with grades of service. The MQTT protocol is based on the principle of publishing messages and subscribing to topics. Clients also connect to the broker and publish messages to topics. Many clients may subscribe to the same topics and do with the information as they please. This protocol works by exchanging a series of MQTT Control Packets in a defined way, and always follow this order (Fig.3.):

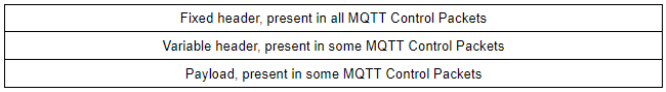

Fig. 3. MQTT Control Packet Structure

Each MQTT Control Packet contains a fixed header (Fig.4.):

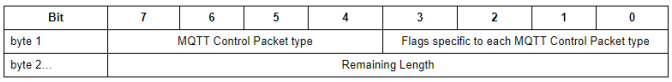

Fig. 4. MQTT Control Packet Fixed Header

The *remaining length* field goes to a maximum of 4 bytes and indicate how many bytes are left of the current packet. So a MQTT fixed header can go from 2 to 5 bytes. Some packets may contain the variable header, which have a 2 byte packet identifier and these effect the *publish* type packets, if the quality of service is greater than 0, and the subscribe type of packets. Finaly the payload is required to connect and subscribe, and it's optional on a publish message.

1) Security in MQTT: As well as in CoAP, MQTT requires an additional encrypted protocol as a cover and it uses TLS. MQTT supports three modes of security:

- *NoSec:* There is no security provided. Messages are transmitted in plain text.

- *Client Identification:* Users must authenticate themselves through username and password and ,unless it's encrypted with TLS,this information is passed in plain text.
- *Certificate:* TLS is enabled. Each device has an asymmetric key pair with a X.509 certificate which is signed by a common trust root. The devices have also a list of root trust anchors that can be used to validate a certificate.

IV. Experiment Setup

The goal of the developed experiments was to study the several modes of security for CoAP and MQTT protocols, analyze how the handshake process goes, and to obtain data, which will allow to take some conclusions about the overhead of the security implementations and the implications of having secure connections in IoT networks.

The main test consist in implement a server and a client, send a POST request, in case of CoAP, or a publish/subscribe, in case of MQTT, and receive the response from the server. This test was repeated for different modes of security for both protocols.

A. CoAP

For implementing CoAP server and client, it was used the Java framework *Californium (Cf)*. For the security mechanisms, an implementation of DTLS named *Scandium (Sc)* was used. It provides a socket-like API which can be integrated with Californium. Because the objective was to obtain the overhead just in number of bytes, both server and client were located in the same machine.

Four security methods were tested: *NoSec*, *RawPublicKey*, *Certificate* without client authentication and *Certificate* with client authentication. For each one, *Wireshark* captures were taken.

B. MQTT

As for the implementation of MQTT, the server and clients were installed in two different machines using *Mosquitto*, an open source broker. As for security, OpenSSL was used to generate the certifacates.In order to receive the published message, the subscriber must be subscribed to the same topic and be listening to the server. This means only one connection is needed to receive multiple messages, but the *Mosquitto* publisher connects the server every time it sends a message.

MQTT also provides different levels of QoS, but only the QoS level 0 was tested.

Three security modes were tested: *NoSec*, *Client Identification* and *Certificate*, and *Wireshark* was the used tool to measure the security overhead. For each experiment it was executed in the following order:

1) Wireshark started capture on the broker;
2) The subscribe request was run;
3) The publish message request was run;
4) Stoped capturing on Wireshark;

V. Experimental Results and Analysis

In this section, there will be presented the results of the tests made and an analysis of the handshake process for each security mode tested will be made.

A. CoAP

It was sent a POST request of a payload with a 5 byte length. The results for the tested security modes are presented below:

1) NoSec: As shown in Fig. 1, there is no handshake previous to the transmission of the POST message. The devices only exchange two packets and the message is sent in plain text.

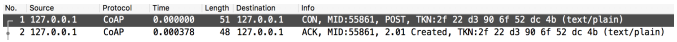

Fig. 5. Wireshark captured CoAP packets (NoSec)

2) RawPublicKey: By using DTLS, it is established a connection before the exchange of CoAP messages. The handshake process is described next: first, the client sends a *Client Hello*, to which the server responds with a *Hello Verify Request*. This packet contains a cookie, which will be returned by the client in the next packet *Client Hello*. This mechanism is used to prevent DoS attacks. Accepted the *Client Hello*, the server sends a set of messages containing the configured security parameters, a session ID, the public key for authentication and the parameters needed for EC Diffie-Hellman encryption. The client responds with its own EC Diffie-Hellman parameters and sends a "Change Cipher Spec" to notify the server that the negotiated parameters will be used from now on. Finally, handshake ends and the messages start to be sent encrypted (see Fig. 6).

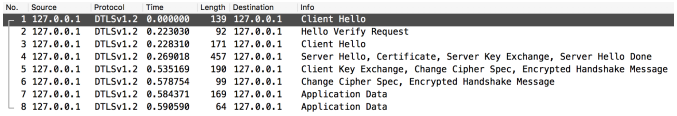

Fig. 6. Wireshark captured CoAP packets (Raw Public Key)

3) Certificate without client authentication: In this security mode, the handshake process is similar to the one described before. The main difference is in the message *Certificate*, sent by the server, which contains, not only a public key, but a full certificate, making this packet much longer.

4) Certificate with client authentication: In order to perform client authentication, the server has to request a certificate from the client. This add two extra packets: a *Certificate Request* (from the server) and a *Certificate* (from the client), consequently increasing the length of the handshake (see Fig. 7).

The table in Fig. 8 summarizes the information about the exchanged messages and their lengths (in bytes) for the tested security modes. Please, keep in mind that we used certificates of 1024 bytes in our tests.

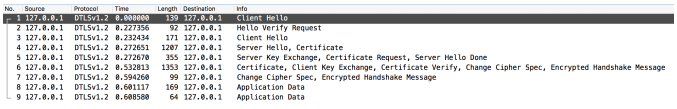

Fig. 7. Wireshark captured CoAP packets (Certificate with auth)

	NoSec	RPK	CertwihoutAuth	CertwithAuth
Client Hello		139	139	139
Hello Verify Request		92	92	92
Client Hello		171	171	171
Server Hello, Certificate		457	1207	1207
Server Key Exchange, Server Hello Done			225	355
Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message		190	190	1353 (+ certificate)
Change Cipher Spec, Encrypted Handshake Message		99	99	99
Post Request	51	169	169	169
Ack	48	64	64	64
Total	99	1381	2217	3510

Fig. 8. Lengths of the exchanged packets)

B. MQTT

Each published message was 16 bytes long and were contained in the topic '/test'.

These are the results:

1) NoSec: In this mode, the process is just a simple distribution of the message. As it can be seen a connection request packet has a 100 byte length packet (see Fig. 9).

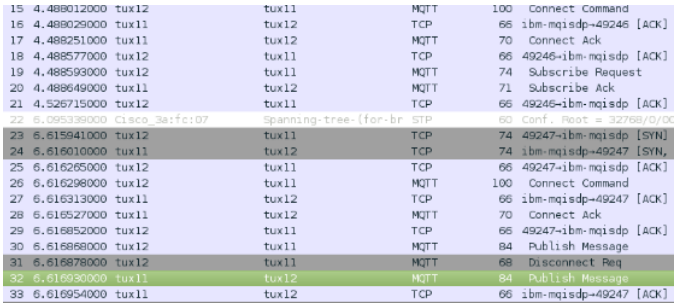

Fig. 9. Wireshark captured MQTT packets with the No Security Mode

2) Client Identification: The difference from this mode to the previous one is that it requires username and password upon connecting to the server. So there's an increment of 9 bytes for the username "jose" and password "1" (see Fig.10).

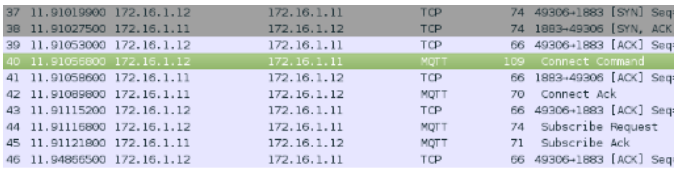

Fig. 10. Wireshark captured MQTT packets with Client Identification Mode

3) Certificate: The other security mode MQTT provides is just like CoAP's security mode: Certificate with out client authentication. As it can be observed, it's logically a lot more over headed (see Fig.11).

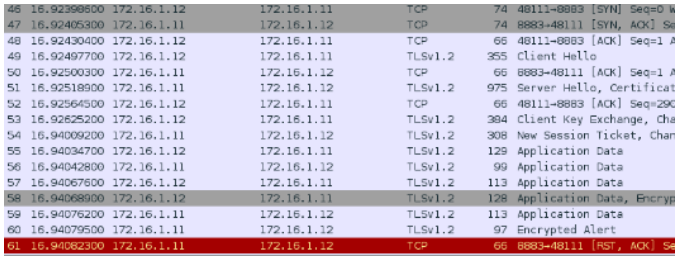

Fig. 11. Wireshark captured MQTT packets with TLS

VI. Conclusion

After analyzing the messages exchanges in both CoAP and MQTT protocols, we can conclude that the introduction of a security layer creates a substantial overhead in the network. These security features don't seem to live up to what the protocols were initially designed for. These protocols were designed for a lightweight and fast communication between small and restrained devices.

However, because this kind of networks is starting to have a huge part in our lives , security is becoming a priority in IoT. While some networks may be "harmless", others may attract the interest of attackers. In these cases, it may be good investment to sacrifice a bit of the performance in order to improve security.

References

[1] Salman T., "Networking Protocols and Standards for Internet of Things", 2015 https://www.cse.wustl.edu/ jain/cse570-15/ftp/iot_prot/

[2] Karagiannis V., Chatzimisios P., Vazquez-Gallego F., Alonso-Zarate J., "A Survey on Application Layer Protocols for the Internet of Things", *Transaction on IoT and Clo ud Computing*, vol. 3, no. 1, pp. 11-17, 2015,

[3] Thangavel D., Ma X. Valera A., Tan H., "Performance Evaluation of MQTT and CoAP via a Common Middleware", *IEEE Ninth International Conference on Intelligent Sensors Sensor Networks and Information Processing (ISSNIP)"*, pp. 1-6, 2014

[4] Shelby Z., Hartke K., Bormann C., "The Constrained Application Protocol (CoAP)" RFC 7252, 2014

[5] S. Jucker, "Securing the constrained application protocol" Masters thesis, Department of Computer Science, ETH Zurich, Switzerland, 2012.

[6] Rahman R. A., Shah B. "Security analysis of IoT protocols: A focus in CoAP", *3rd MEC International Conference on Big Data and Smart City*, 2016

[7] Bhattacharyya A., Bose T., Bandyopadhyay S., Ukil A., Pal A., "LESS: Lightweight Establishment of Secure Session: A Cross-Layer Approach Using CoAP and DTLS-PSK Channel Encryption", *29th International Conference on Advanced Information Networking and Applications Workshops*, 2015

[8] Oasis.MQTT Oasis Standard Specification. Online: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (Accessed on 2017-11-21).

[9] Tang, Konglong, Yong Wang, Hao Liu, Yanxiu Sheng, Xi Wang, and Zhiqiang Wei. "Design and implementation of push notification system based on the MQTT protocol." In International Conference on Information Science and Computer Applications (ISCA 2013), pp. 116-119. 2013.

[10] Wireshark.org, "Wireshark", https://www.wireshark.org, 2017

SDN/NFV review and performance of network security functions

Pedro Gomes
Faculdade de Engenharia da
Universidade do Porto
Porto, Portugal
Email: up201307839@fe.up.pt

Erick Caetano
Faculdade de Engenharia da
Universidade do Porto
Porto, Portugal
Email: up201702302@fe.up.pt

***Abstract*—With the expansion of network infrastructure came the challenges of maintaining this infrastructures and of keep evolving them performance and security-wise, on the last decade this challenge has presented a wall to the fast growing networks and which Software Defined Networks and Network Function Virtualization tries to help solve with new ways of network management with centralized hight level management and more disaggregated software solutions for otherwise commercial dependent network solutions and which enabled the emergence of Open Source projects and software-defined standards. On this paper we discuss the possibilities and problems found while trying to setup a simple Authentication server using this kind of technologies. We used the open SDN platform Ryu for this finality and provide some proof of concept of the possible approach to accomplish the setup of a simple network management application.**

I. Introduction

The work presented hereafter was developed from the solution proposed to solve a problem proposed on a systems and networks security class, the problem came from by after having as work topic the present paper title: "SDN/NFV review and performance of network security functions", the challenge was to using the related technologies to setup a Authentication server which would bar the access of all external users to what could be an internal network, host or simple virtual machine/space on one or more hosts. At this project starting point there was no basic knowledge of what were the SDN and NFV technologies and what possibilities were tied with them, so, as initial step we worked on understanding what they present as functionality and the kind of concepts that we should be aware to develop our project using them.

II. Related work

The first step on the development was on, as already surmised, understanding how SDN and NFV technologies could be employed to solve the problem previously presented, while doing research about the topic we understood that the ability presented by Software Defined Network platforms allowed the creation of an application capable of a centralized control of the network, this was a meaningful functionality presented that at this point seemed to allow the concoction of a solution. From this point we invested our time on research the possible SDN frameworks and how they worked with the underlying network and what interfaces were provided to develop under them. At this point, 2017, there are already major frameworks for SDN development that are well established and Open Source and that are already being used on major industry projects or from which commercial versions have diverged taking important roles on the management of great big network infrastructures. Examples of major SDN projects at this point are the OpenDaylight and the ONOs platform, both are huge Open Source projects with large development communities which have the participation of major network related industries and from which major vendor solutions used as building block, integrating or embedding this platforms code to construct more complex and service specific solutions. After finding this platforms we understood how big the development on SDN solutions as accomplished on little more than 10 years, at this point we had the choice of using this platforms to develop our solution or to use a more simple platform like Ryu. Contrary to the ONOs and OpenDaylight which are big platforms and that already have hight level network management functions that would make the solution have a major focus on using the platform and doing hight level configurations, Ryu which was a much simpler platform where we had much more contact with low level things like OpenFlow which was a major step on the technological side and that made possible the uprising of things like SDN. As we wanted to focus on better understanding the underlying technologies we opted to use a simpler platform that provided only a basic interface to construct the necessary solution as this would enable us to have a better insight into the internal works of SDN platforms.

After choosing the SDN platform to work with, it was time to start the development but for that we found that it was necessary to create the suitable development environment which should enable us to have a better development workflow. Within this environment we should be able to easily setup the necessary network with the necessary hardware and software which would include the necessary hosts with the ability to run the necessary services and/or programs independently and the necessary switch with OpenFlow compatibility, which will allow our SDN platform to control our network. For this purpose we found that Mininet was a very good solution since it had as it's core the objective of being a fast virtual test bed for SDN development. After adopting as solution to

the development environment the Mininet software we started by downloading and configuring a Ubuntu Virtual Machine with the Mininet software package and with the creation of a network topology in which our solution would be established. This was achieved by creating a network topology script that would establish the necessary links and hardware as well as the initialization configuration. This topology consisted of a relatively simple configuration where we have several hosts on a intra-net with a switch as the interface to the outside Internet and where one of the internal hosts would be the administration host where the network administration should stand as well as the Authentication server. With this we proceeded to the development fase of our project.

III. Solution

The first step to the solution development after establishing a suitable environment was to create the normal flow control tables to do the normal work of a switch and deliver the packets as expected of a switch layer 2 functions, this permitted us to better understand how the flow control is made and how to work with the Ryu platform. On the second step we selected the necessary rule-set to make the necessary re-routing so that the Authentication would be a necessary step to access any host represented by an IP address in the intra-net. After being able to use the rule-set to make the right flow control to get the expected results we implemented the management of the control over the Ryu platform within a PHP script so that this control could be made any time a request to the server was made and so that a administration page could be established. When enabled the authentication on the administration page all addresses not known that request a page will be redirected to an authentication page on the admin host and when a successfully authentication is made the access to that authenticated user is granted to that user addresses list. We also implemented a simple admin page that should only be accessible when the user Authenticated has admin rights and which provides simple manual control of the network, it provides functionality like the drop of specific incoming communications to or from specific addresses or the re-routing to different hosts/addresses.

IV. Evaluation

Through the controller, it was possible to create tables and rules to route packets, assign IP and ports, and control the switch. This has resulted in better network security and management because you can control all traffic.

After creating the switches and the routing tables, we developed a PHP script where a micro-server exists for the hosts to connect to the network, if the administrator is to log in, he has access to the switches and can change the IPs, the view the tables. When creating the login server, we chose to do the verification through the IP, so if the host had an IP already known by the controller table it would have access to the intranet, but when we chose this form of verification, it made us think about network security. So we analyzed that this verification could have been done through MAC or cookies.

V. Conclusion

With the conclusion of this work we were able to understand that it's possible to implement network security using technologies like SDN with the use of flow control functionality to inspect and filter different properties of the packets flowing throw the network. We can allie the possibility of having part of the control being done on the switch with having the possibility of re-routing the flows to the controller where something like a network intrusion prevention system like Snort could be employed to have a more powerful security mechanism doing the security inspection on previously selected packages. This type of applications could give use to the centralized network control provided by the SDN to make a better judgement on the network actual state and to have a better control and ability to filter the different communications that flow throw the network having access to do the control on network switches and to establish rules on a more network wide range.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

Side Channel Attacks

Eduardo Rodrigues
Master in Electrical and Computers Engineering
Faculty of Engineering of the University of Porto
Email: up201304756@fe.up.pt

Hélio Puga
Master in Electrical and Computers Engineering
Faculty of Engineering of the University of Porto
Email: up201305586@fe.up.pt

***Abstract*—Encryption is often proposed as a method to guarantee the privacy of users. However, that methodology is not able to hide all the information about the data in the Web. This work focused on a side-channel attack to assess the ability to obtain information only by capturing and analyzing encrypted traffic. SSL traffic from several news websites were captured in order to build graphical representations of the evolution of the received information as a function of time. Two algorithms were developed to analyze the gathered data and disclose which website or page a victim would be using. Both of them were built using the technique Dynamic Time Warp. The results show a hit rate of 86% when trying to identify a web site but only 65% when trying to identify the page the user was accessing.**

I. Introduction

Side-channel attacks are a passive kind of attack. This means that they try to gather information about the victim without attacking it directly. Unlike active attacks, which explore flaws of the encryption algorithms or try to decrypt victims' data, this kind of attack is typically non-invasive. Although there are many types of side-channel attacks like monitoring power consumption, electromagnetic radiation, sound or timing information, this project assessed the ability to obtain information only by the analysis of the encrypted traffic. The main goals of this project were:

- Try to predict which website the victim could be accessing.
- After knowing which website was being accessed, try to identify which page of that website the victim was using.

In order to do that, two algorithms were proposed:

- The average curve method (section IV-A)
- The closest curve method (section IV-B)

Both of them being based on Dynamic Time Warp (DTW) (section III), but with different approaches.

II. Data characterization

To detect the page or website that was being accessed by the victim it was necessary to build a dataset which was going to be used to train the algorithms. Four news websites were analyzed:

- Jornal de notícias [1]
- Diário de Notícias [2]
- TSF [3]

[1]www.jn.pt
[2]www.dn.pt
[3]www.tsf.pt

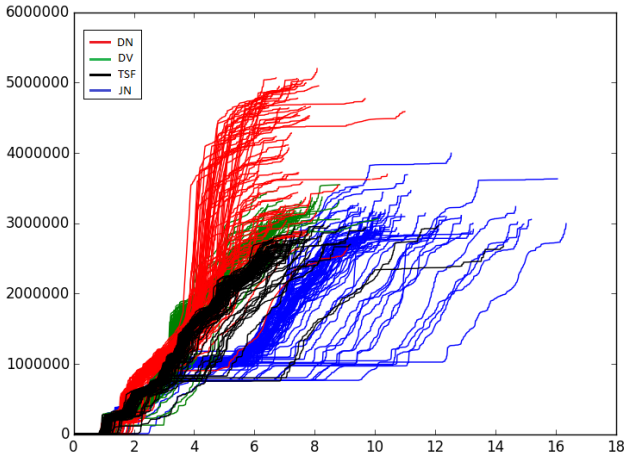

Fig. 1. Distribution of all accesses over time to each of the pages.

- Dinheiro Vivo [4]

The choice of these websites was made due to the fact that all of them were designed and deployed by the same company[5], so their layout and engine was the same, resulting in very similar traffic patterns, impossible to visually distinguish. In each of these websites, 3 different pages were chosen. Each one of them had 25 captured accesses to the training dataset and 15 others to the testing dataset. The automation of these captures was made using selenium[6] and tshark. Figure 1 shows the amount of encrypted information received in function of time to each of the 300 captured accesses. Each website is represented in a different color. By the analysis of the figure one can conclude that it is not possible to visually detect the website that is being used by the victim. Except for DN (red), all of them had similar maximum amount of received bytes. However, even some of DN's accesses had curves overlapping with other websites, which suggests that it's visual detection would not be trivial either. The only big difference was observed to the JN accesses that had a longer duration when compared with all the others.

[4]www.dinheirovivo.pt
[5]www.globalmediagroup.pt
[6]http://www.seleniumhq.org/

III. Dynamic Time Warp

DTW is an algorithm that measures the similarity between two time-series that can differ in velocity. The algorithm analyze both of the distributions and deforms them, by stretching or compressing, in order to approximate them as much as possible (Figure 2). The choice of this algorithm was made due to the fact that every captured access for a specific website had a similar pattern, only diverging in time. In this way, the curves were compared by its evolution pattern and not by the amount of bytes at each time-point.

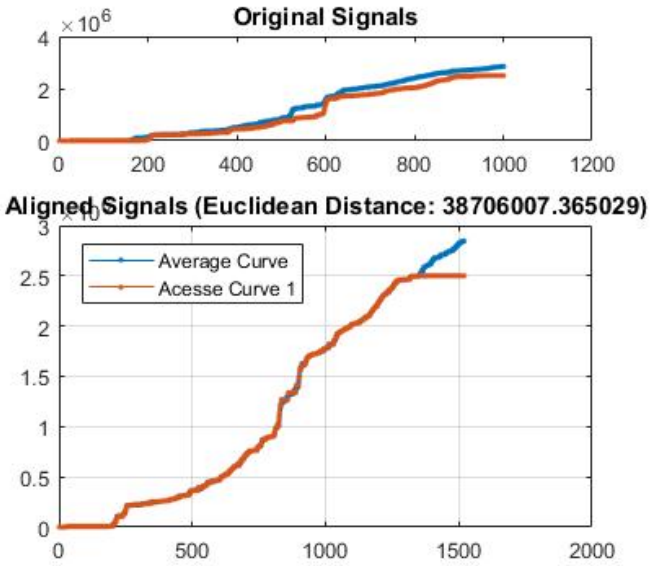

Fig. 2. DTW execution

IV. Methodologies

A. Average curve method

This method consisted in calculating the average curve to each page of each website. Since the number of packets and total time between accesses varies considerably, it was necessary to make an interpolation of each curve, so that all curves had a fixed number of points. In this way, it was possible to retrieve the average value at each time-point, resulting in an average curve of all curves. After calculating the average curve for all the pages, the algorithm compared the in-test accesses to each one of the average curves using DTW. Finally, the algorithm classified the testing curve with the same label as the average curve which yield a lower distance.

B. Closest curve method

This method aimed to discover the page which the user is accessing by comparing its encrypted traffic to a reference curve of each website. To elect this reference curve, the distance of each curve to all the others of the same website was calculated. The one which yielded a lower sum of distances was elected as the closest curve to all others, *i.e.*, the reference curve. The distance between accesses was calculated using DTW. Table I shows an example of the algorithm's choosing process. "access1" was the curve with the lowest sum, thus being elected as the closest curve. After choosing the reference curve of each page, each access of the testing set was compared to each one of those reference curves. The algorithm classified a testing curve with the same label as the reference curve to which the distance was lower.

	access1	access2	access3	sum
access1	0	1000	2000	**3000**
access2	1000	0	2500	3500
access3	2000	2500	0	4500

TABLE I
EXAMPLE OF THE CLOSEST CURVE METHOD CHOOSING PROCESS. ACCESS1 WAS CHOSEN AS THE CLOSEST CURVE.

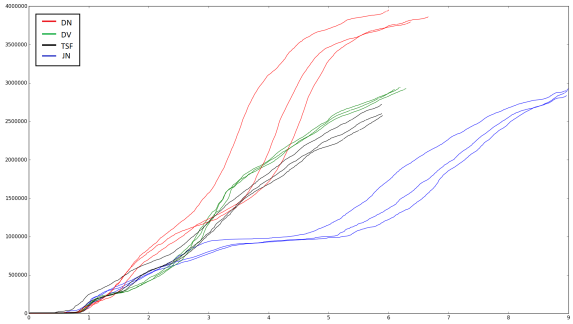

Fig. 3. Average curves of the 12 pages.

V. Results

A. Experience 1

The first experience consisted in finding the website being accessed by the victim. First, an average curve method-based approach was made. The average curve of each page is shown in Figure 3. Each curve represents a page and each color represents a website. It is possible to observe that inside the same website there is a pattern between the three average curves, which might suggest that differentiating a page inside a website might be a difficult task. On the other hand, it is clearly possible to distinguish the four websites. The testing set of each page was submitted to the algorithm and the results are shown in Table A. It can be observed that the hit rate inside a page shows a significant variation, between 0 and 93%, but the value is typically low (the average value is around 30%). This result was expected due to the high similarity between curves inside a specific website. However, the hit rate of the website revealed higher values, typically higher than 60%, with a average of 70% of matches.

Posteriorly, an approach to the same problem was made using the closest curve method. First, it was necessary to train the algorithm to find the closest curve of each page, as described in section IV-B. These curves are shown in Figure 4. In this algorithm, as well as in the previously described one, the curves referring to JN stood out by their longer duration. However, the curves with higher amount of total bytes were the ones referring to DV, unlike the previous algorithm where

Testset	Hit rate
jn1	53,3%
jn2	60%
jn3	20%
dn1	100%
dn2	100%
dn3	100%
dv1	60%
dv2	66,7%
dv3	66,7%
tsf1	100%
tsf2	46,6%
tsf3	100%

TABLE II
AVERAGE CURVE METHOD RESULTS.

Testset	Hit rate
jn1	100%
jn2	100%
jn3	100%
dn1	100%
dn2	86,7%
dn3	93,3%
dv1	40%
dv2	66,7%
dv3	60%
tsf1	100%
tsf2	80%
tsf3	100%

TABLE III
CLOSEST CURVE METHOD RESULTS.

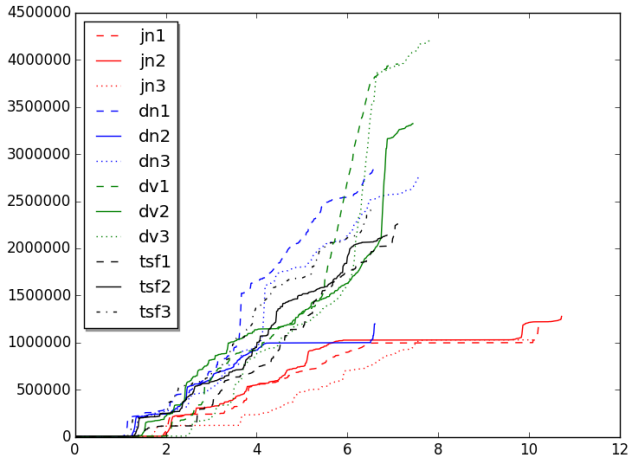

Fig. 4. Closest curve of the 12 pages obtained by the closest curve method.

DN stood out. In this scenario, there wasn't a clear distinction between different curves of different websites, contrasting with what had been observed for the previous algorithm (Figure 3). Each one of the testing accesses of each page was submitted to the algorithm. It classified the access with the label of the reference curve which yielded a lower distance. Table B shows the results obtained using this method. The hit rates for the page were, once again, very variable, with values ranging between 0 and 100%. However, the hit rate for the website yielded higher values, with values between 40% and 100%, with an average value of 86%.

B. Experience 2

The abovementioned experience showed that it is possible to find which website a user is accessing, only by the analysis of it's encrypted traffic. On other hand, the accuracy of the page detection yields random values, suggesting that the page identification using these algorithms could be unfeasible. However, this second experience aimed to disclose an answer to the possibility of detecting the page a user was accessing, after knowing which website he was in. To do that, for each testing set of each page, the datasets were reduced to the

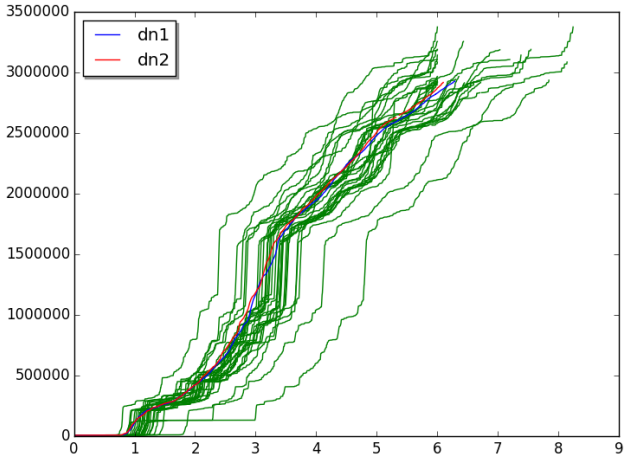

Fig. 5. Teste curva média para o DN.

	dn1	dn2	Taxa
dn1	3	12	20%
dn2	2	13	86,7%

TABLE IV
COMPARISON BETWEEN DN'S PAGES.

	jn1	jn2	Taxa
jn1	14	1	93,3%
jn2	10	5	33,3%

TABLE V
COMPARISON BETWEEN JN'S PAGES.

accesses of the pages of the respective website. For instance, after detecting that the user was browsing DN, the test curves were only compared to the average or references curves of that website (DN1 and DN2). In this experience we only tried to detect the page between two possibilities only. The first approach was made using the average curve method. Figure 5 shows the test example of DN. The green curves are the testset of DN1 and the red and blue curves are the average curves of DN1 and DN2, respectively. One can verify that the test curves display a pattern in their graphical representation. The average curves of DN1 and DN2 are almost overlapped which suggests the impossibility of effectively detect which page the testing set represents. The results in respect to DN are presented in IV. As expected due to the overlapping curves, the distances calculated by DTW were almost the same, yielding a non-deterministic result. The page hit rate of this website and the following others (shown in Tables V, VII, VI) highly suggests that the detection of the page using this algorithm is not possible.

The second approach was made using the closest curve method. The references curves calculated in the previous experience (Figure 4) were used in this experience as well.

	dv1	dv2	Taxa
dv1	6	9	40%
dv2	7	8	53,3%

TABLE VI
COMPARISON BETWEEN DV'S PAGES.

	tsf1	tsf2	Taxa
tsf1	3	12	20%
tsf2	0	15	100%

TABLE VII
COMPARISON BETWEEN TSF'S PAGES.

	dn1	dn2	Taxa
dn1	6	9	40%
dn2	5	10	66,7%

TABLE VIII
COMPARISON BETWEEN DN'S PAGES.

	jn1	jn2	Taxa
jn1	10	5	66,7%
jn2	7	8	53,3%

TABLE IX
COMPARISON BETWEEN JN'S PAGES.

	dv1	dv2	Taxa
dv1	9	6	60%
dv2	4	11	73,3%

TABLE X
COMPARISON BETWEEN DV'S PAGES.

	tsf1	tsf2	Taxa
tsf1	7	8	46,6%
tsf2	3	12	80%

TABLE XI
COMPARISON BETWEEN TSF'S PAGES.

Once again, the tests were made admitting that the website the user was browsing was already known. The obtained results to each page are shown in Table VIII, IX, X, and XI. These results show that hit rate increased in comparison to the previous approach, but they are still around 50%, suggesting the impossibility to predict the page being tested.

C. Experience 3

The last experience was made to analyze the algorithms' performance in their training phase. The training of the average curve method consists in interpolating all of the curves and calculate the average of the total bytes received at each instant. In what concerns the closest curve method, the training consists in measuring the distance of from one curve to all the others and find the one with the lowest sum of distances. Table XII shows the amount of time it takes for each algorithm to complete the training phase. The closest curve method had a training time 30 times larger than the average curve method. This result is easily explained by the fact that, in order to find the reference curve of the closest curve method, it is necessary to make $N^2 - N$ instances of the DTW, being N the number of curves in the training set. An improvement to the closest curve algorithm was made in order to reduce the number of times the DTW had to be done. When the cumulative sum of a certain curve surpassed the lowest sum stored, the algorithm would ignore the remaining comparisons and move to the next curve. However, this update only improved the training time in approximately 100 seconds.

VI. DISCUSSION

The results presented in Experience 1 showed a hit rate of 86%, which suggests it might be possible to detect the website only by the analysis of encrypted traffic. However, the detection is only possible if the dataset is regularly updated. This is due to the fact that the variable with the most impact on traffic is advertising, which is constantly changing. We repeated the experience with a testing set with a difference of one week from the training set and the produced results were much worse.

Average curve (s)	**Closest curve(s)**
20	600

TABLE XII
TRAINING TIME TO EACH ALGORITHM IN SECONDS.

The worse results produced by the average curve method are caused by the fact that all curves have the same weight when calculating the average values. With a dataset of only 25 accesses, a single curve with an anomalous behaviour will have a significant negative impact when calculating the average. A possible mitigation could be the utilization of a larger dataset, or evaluating the curves, assigning different weights to each one of them before calculating the average.

The second experience had negative results because different pages within a specific website have almost the same contents, diverting only on the text and some images. The traffic produced by these variable elements is reduced when comparing it to the amount of traffic generated by the advertisement, that is common to all the website's pages. A possible solution to this problem could be to analyze the sequence of objects downloaded by the browser and verify whether that sequence is constant or not. In the former case, the next step would be to detect in which of the curves' sections were present the said variable elements, and to compare the testing set only using that section of the distribution.

VII. CONCLUSION

In this work we present a methodology of website identification by analysis of encrypted traffic. We did so by capturing traffic generated by different websites in order to build a dataset that can be employed to calculate a reference curve of each page. The identification was made using the DTW technique which calculates distances between two time series. The results suggest that the website identification is possible, but the the page identification within a specific website is not. These results imply that the the user's behaviour can be monitored even without decrypting the information, which is a method used by several intrusion detection systems. However, the user's privacy is maintained because it seems impossible to detect which page one might be browsing. Although detecting news websites might not look critical, other types of websites, such as banking ones, might be more interesting for an attacker to sniff. In this sense, users' privacy becomes a critical subject.

In this work we could retrieve better results using the closest curve method, however it's training time is 30 times larger than the average curve method. This trade-off might not be worth due to the mandatory need of re-training caused by advertisement changes.

Future work may involve improvement of these experiments, as suggested in section VI, or application of this methodologies to other websites whose information acquisition by an attacker would pose a more problematic scenario.

APPENDIX

A. Confusion matrix for Experience 1 using average curve method

testset	jn1	jn2	jn3	dn1	dn2	dn3	dv1	dv2	dv3	tsf1	tsf2	tsf3	page rate	site rate
jn1	3	1	4	0	2	4	1	0	0	0	0	0	20 %	53,3%
jn2	0	6	3	1	0	3	1	0	0	0	1	0	40%	60%
jn3	0	1	2	0	1	5	0	0	0	0	6	0	13,3%	20%
dn1	0	0	0	3	2	10	0	0	0	0	0	0	20%	100%
dn2	0	0	0	4	2	9	0	0	0	0	0	0	13,3%	100%
dn3	0	0	0	0	1	14	0	0	0	0	0	0	93,3%	100%
dv1	0	0	0	0	0	2	0	0	9	0	1	3	0%	60%
dv2	0	1	1	0	2	2	4	0	4	1	0	0	0%	66,7%
dv3	0	0	1	0	0	3	2	0	8	1	0	0	53,3%	66,7%
tsf1	0	0	0	0	0	0	0	0	0	0	10	5	0%	100%
tsf2	0	0	2	2	0	4	0	0	0	0	5	2	33,3%	46,6%
tsf3	0	0	0	0	0	0	0	0	0	3	0	12	80%	100%

B. Confusion matrix for Experience 1 using closest curve method

testset	jn1	jn2	jn3	dn1	dn2	dn3	dv1	dv2	dv3	tsf1	tsf2	tsf3	page rate	site rate
jn1	15	0	0	0	0	0	0	0	0	0	0	0	100 %	100%
jn2	7	8	0	0	0	0	0	0	0	0	0	0	53%	100%
jn3	6	6	3	0	0	0	0	0	0	0	0	0	20%	100%
dn1	0	0	0	1	2	12	0	0	0	0	0	0	6,7%	100%
dn2	0	0	0	1	3	9	0	0	0	1	0	1	20%	86,7%
dn3	0	0	0	2	2	10	0	0	0	0	0	1	66,7%	93,3%
dv1	0	0	0	3	0	2	0	2	4	0	1	3	0%	40%
dv2	0	0	0	2	0	3	0	2	8	0	0	0	13,3%	66,7%
dv3	0	1	0	2	1	1	3	0	6	0	0	1	40%	60%
tsf1	0	0	0	0	0	0	0	0	0	1	0	14	6,6%	100%
tsf2	0	1	2	0	0	0	0	0	0	8	0	4	0%	80%
tsf3	0	0	0	0	0	0	0	0	0	1	2	12	80%	100%

DDoS attack and defence strategies

João Magalhães
Faculdade de Engenharia da Universidade do Porto
Email: up201305379@fe.up.pt

Luís Zilhão
Faculdade de Engenharia da Universidade do Porto
Email: ee12150@fe.up.pt

***Abstract*—This paper was developed in the context of Security in Systems and Networks, more specifically DDOS attacks on systems and Networks. In recent times, DDoS has been an ever present threat to existing services, with possibilities of large scale implications in the event of a successful one, such as the Dyn Cyberattack of 2016. Our goal is to explore a possible measure to diminish the effect of a DDOS attack on legitimate users. The idea is to attribute priority to users with valid sessions, this users will have access to 75% of resources were as the rest will have access to the remaining 25%, attackers will fall in this 25%. Several tests were carried out to characterise the solution and the results will be presented.**

I. Introduction

In a Denial of Service attack the aim is to drain the resources of a targeted system, thus making it slower or stopping it completely. For a Distributed Denial of Service attack, the goal is the same but the origin comes from several machines, instead of a single one. There are several types of attacks, ranging from all of the levels of the network layers [1], such as the **UDP Flood** or **SYN Flood** for transport layer attacks, or the **HTTP GET Flood** for the application layer [2]. The latter was chosen for this work. The procedure is relatively straight forward, as it consists of flooding the target machine with requests, therefore impairing its ability to handle requests from legitimate users, as portrayed in figure 1.

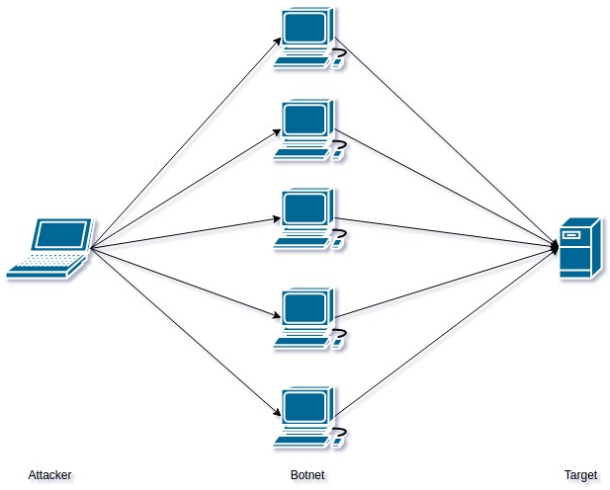

Fig. 1. Network diagram for a DDoS attack

Certain headers are added to each request in order to increase the difficulty of filtering them, in an attempt to make the requests appear that they don't possess a pattern, such as different user agents, and also to increase resource utilisation by the target machines using the No-Cache header. Typically a collection of infected machines, or zombies, will form the botnet which will carry out the attack. These types of networks are capable of generating an immense amount of traffic. The ways in which machines are infected and the prevention method are not covered in this paper, as the main goal is to provide a defence solution and characterise its efficiency in the event of such an attack, and not to prevent it. This type of attack forms valid requests and aims to force the target application to use the maximum resources as possible with each request, therefore each request is a valid one. The solution that we will propose uses a load balancer to distribute the load in accordance to certain rules. These are machines which are placed in front of the servers hosting the website and that decide where each request must go at any given moment. Several rules might be used to determine the destination, such as the available bandwidth on a certain link, or CPU load across all machines. Another advantage that they provide is the ability to mask the servers from the users, acting as a reverse proxy, because they are not able to access the servers directly, without going through the load balancer first.

II. Related work

There has been extensive work in this area when it comes to defence strategies. In [3] an attempt was made to differentiate traffic with recourse to TCP packet header filtering. This is in order to mitigate TCP level attacks, in which the flood will consist of several TCP packets, with spoofed source IPs so as to mask the origin of the flood. Using traffic analysis to determine the IP flow, it is possible to identify if it is a legitimate flow, as DDoS attacks typically are very aggressive and stand out in this regard. However, in this paper the attacks are on an application layer level and packets are not spoofed, as it is not possible to do so with HTTP requests, as that would impede the three way handshake required for each one.

III. Solution

In order to diminish the impact of an attack, the first step was to attempt to distribute the load between several servers, as opposed to having a single machine handling all the requests. Of course, this implies an addition of resources as opposed to providing a solution that requires no additional investment, but nevertheless it is a common setup as no highly used website has one single server for the entire infrastructure. This is the basic setup that most services use. In addition to this, we

proposed a solution that prioritises users which are already authenticated, so that only new users that haven't logged in are affected. This is done with resort to cookies, and results in the network setup in figure 2.

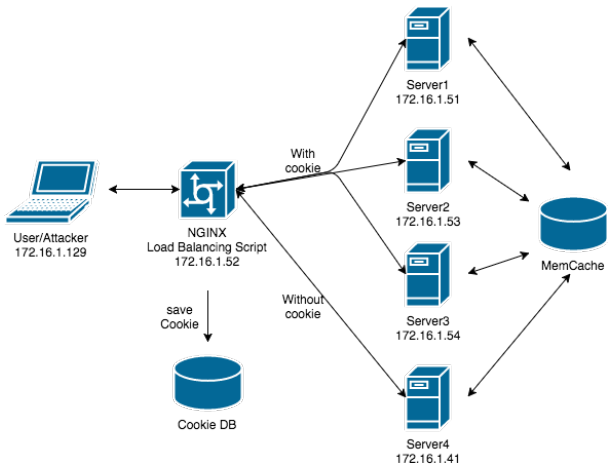

Fig. 2. Network diagram for the setup

There are 4 identical servers that share created sessions though a memcache. Any request from a user is directed at the load balancer and the load balancer will then forward the request to one of the four identical servers.

A python script was developed to implement a load balancer that gives priority to users with a valid session. Let's say that a user tries to access a server for the first time, that request arrives at the load balancer with no cookie (no session). Because there is no cookie in the request, the load balancer will forward the request to server 4. Server 4 creates a session for the request, saves the session in the shared memcache so all servers have access in future requests and sends the response back to the reverse proxy. At this point the reverse proxy will analyse the response and search for cookies. If one is indeed found said cookie will be stored in a data base. When the same user sends a new request it will carry a cookie, the reverse proxy identifies the cookie in the request and confirms its validity in the cookie data base. When an authentic cookie is found in the request the reverse proxy forwards the request to one of the three first servers in a round robin fashion. This setup attributes 0.75 of server capacity to users with a valid session and 0.25 to the rest.

IV. Evaluation

In the following Figures each coloured line represents a cdf of the latency of the service under attack, carried out using a different numbers of threads to simulate different degrees of attacks

In order to evaluate the results, first it was best to establish a baseline for comparison, which meant that we needed an existing professional and proved load balancing tool. **NGINX** [1] was chosen for this role, as it is widely used in the market and easily implementable. For this performance comparison NGINX was working in its most basic function, distributing the load through 3 equivalent servers.

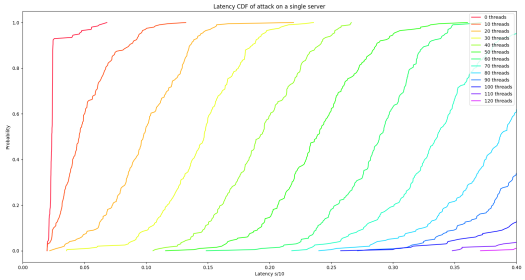

Fig. 3. Latency CDF during attack on a single server

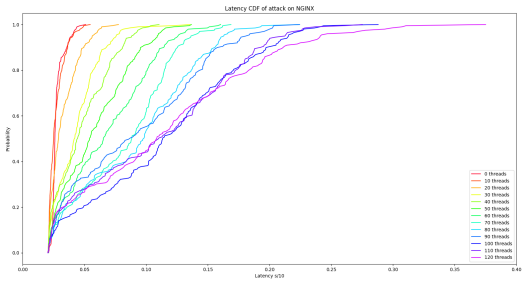

Fig. 4. Latency CDF during attack on NGINX balancing between 3 servers

In figure 3 and figure 4 we can observe the expected result, using a reverse proxy distributing the load through 3 equivalent servers the latency of the service is improved by more than 3 times when compared to the single server.

Now that we know the sort of performance improvement one achieves by using a reverse proxy to distribute the server load, we still need to assess whether our implementation is viable. To go about doing so NGINX was again used for comparison and and with both reverse proxys on their basic functions, simply receiving requests and forwarding them to one of three equivalent servers in round robin.

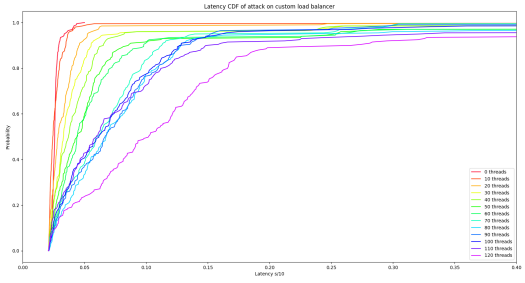

Fig. 5. Latency CDF during attack on custom load balancer

[1] https://www.nginx.com/

The graphs in figure 4 and figure 5 are the results of the proposed comparison. The analysis of the results will be divided in two parts.

For about 80% of the requests, it is visible that the developed load balancer actually performs better than NGINX, having lower latency for the requests for each number of threads.

Now looking at the 20% with higher latency we can see the advantage of using optimised tools such as NGINX, as the slowest requests will never be too far of the average as they were when using the custom reverse proxy, where the slowest requests might take an abnormally long amount of time.

We have now assessed the the advantages of using a reverse proxy and we have established that the developed reverse proxy is a viable solution when it comes to simple load balancing. As this is the case, the next step is to evaluate the performance of the cookie based load balancer, which will not only distribute the load across the available servers, but also check whether they are from users with existing sessions and thus choose the server accordingly.

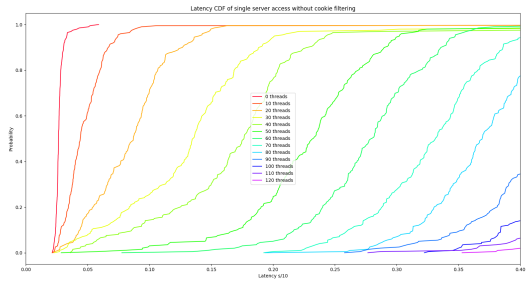

Fig. 6. Latency CDF during attack without cookie filtering

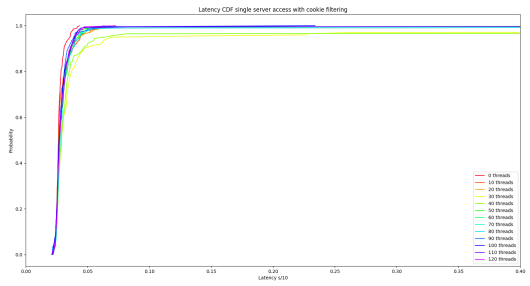

Fig. 7. Latency CDF during attack with cookie filtering

The latency for the access to the website where there is no filtering can be viewed in figure 6. It is visible that the user experience will be significantly worse when compared to figure 7, as the latency is superior and keeps increasing along with the number of threads being used in the attack. Since all the requests originated by the attack are considered by the reverse proxy as non authenticated users, they will therefore have access to only 25% of the server capacity, as opposed to the remaining 75% for users with a valid session.

Figure 8 shows how the cpu's running the different nodes in the network evolve as the attack is being carried out. As time progresses the level of the attack is incremented, the number of threads carrying out the attacks increases and therefore the load increases.

The green trace represents the load in the cpu running the reverse proxy, the orange trace if the load of a cpu running a server for legitimate users and the blue is the cpu of where server 4 (Figure 2) is running, which corresponds to the server servicing non authenticated users.

As we can see the only node affected by the attack is a node only used to attend not authenticated users, meaning that the developed reverse proxy was able to shield authenticated users from an attack, which was the main goal of this work.

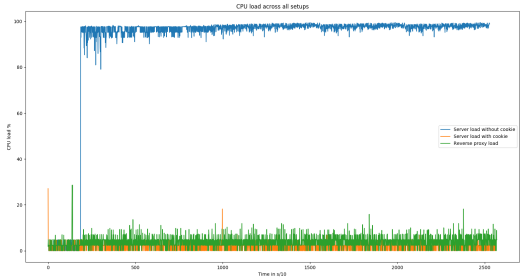

Fig. 8. CPU load during an attack

V. Conclusion

The main goal of the developed work was to mitigate the effect of a DDOS attack on legitimate users, and considering the results presented before it is safe to conclude that the main goal was indeed achieved.

However, it would be trivial for an attacker to understand how the reverse proxy attributes priorities and to find a way around it. To make the idea behind this work more viable in a real situation, the reverse proxy would have to take into account a lot more parameters and additional security measures than simply whether a request has a valid session.

Other things to take into account would be for example the number of requests a user makes in a certain period of time, and block the access if that number is considered excessive, the ip origin, etc. Regardless of this, it can still be a useful tool when the source of the denial of service is legitimate traffic. That is to say, the servers are overloaded, not because of malicious intent, but merely from heavy usage in a certain period of time. In this case, this would provide a guaranteed quality of service to existing users which is to be expected from certain websites.

References

[1] Osi model - wikipedia. https://en.wikipedia.org/wiki/OSI_model. (Accessed on 12/10/2017).

[2] Ddos attack types & mitigation methods — incapsula. https://www.incapsula.com/ddos/ddos-attacks/. (Accessed on 12/21/2017).

[3] Z. Ye, W. Shi, and D. Ye. DDoS Defense Using TCP_IP Header Analysis and Proactive Tests. In *2009 International Conference on Information Technology and Computer Science*, volume 2, pages 548–552, July 2009.

SSRE CTF: Capture The Flag Competition Organization

Eduardo Sobrinho
up201305172@fe.up.pt

Rodrigo Vaz-Pires
up201303277@fe.up.pt

***Abstract*—Regarding the Security for Systems and Networks course it was proposed organizing a beginners Capture The Flag competition as the final project. The objective of this report is to show how this competition was organized and how CTFs work in general. We will cover not only our approach to this project, but all the details about a general CTF competition.**

I. Introduction

Capture the Flag (CTF) is a special kind of information security competitions.

They can generally be included in one of two types: Jeopardy-Style and Attack and Defense Style.

The Jeopardy-Style consist on various challenges divided in different categories. Teams win points by solving these challenges capturing flags. At the end, the winning team it is the one with more points.

In the Attack and Defense Style each team owns a computer network. In a first stage, teams must fix eventual vulnerable services and develop exploits. On the second stage, all the networks are connected and the competition begins. Teams should attack opponents networks while protecting their own.

In some cases, we can have mixed competitions which joins the two types in one.

II. Motivation

Organizing a competition like a CTF it is a huge advantage for our knowledge in this area. It not only makes us work our skills in the Cyber Security but makes us get deeper and better technical skills compared with the ones we would get on an average computer science program. In addition, it gives us the idea of possible security breaches that can happen during this kind of competitions.

For students like us, that are making contact with Cyber Security for the first time in our degree, it is a good way to make ourselves into this matter and to motivate other student's interest in this area.

III. Solution

Since it is the first time we are dealing with most of these topics our main goal was to develop something simple and appealing to newcomers like ourselves. We decided to create a Jeopardy-Style CTF since it's easier and, in this case, a more interesting choice.

There are different types of challenges to be implemented that are usually divided into six main categories:

- Cryptography: Flags are hidden in encrypted files or text. The main objective is to explore vulnerabilities in different encryption schemes;
- Stenography: The art of hiding information in files such as text, images or musics;
- Web: Involves attacking commonly known website vulnerabilities such as SQL Injections on databases;
- Digital Forensics: Like the name suggests, these challenges consist on examining and process hidden pieces of information out of static data files;
- Reverse Engineering: After receiving a compiled binary file, the main objective decompile these executables to get the assembler instructions for solving the problem;
- Packet Analysis: After receiving a packet capture, the objective is to analyze the content with tools such as Wireshark and discover the flag.

All the proposed challenges in this CTF were entirely developed by us. This creates the ability to adjust the contents, difficulty and, at the same time, give us the insight of developing challenges in this environment.

IV. Other CTF competitions

Logically, our first steps was to discover other CTF competitions and how they were organized. We looked into CSAW CTF and iCTF [3] which are two known CTF competitions that are organized yearly and can reach out to more than one thousand participants. In addition, we browsed multiple online platforms such as CTFtime.org to get us into this whole idea of developing challenges and how to solve them.

V. Competition Organization

The competition will occur between teams of two elements and will be restricted to the faculty students. The competition itself it is going to consist in two main phases:

1) Pre-competition (starts 18 Dec 2018): where the platform will be open for team registration and some easier challenges will be available for solving. This will give teams the ability to start straight away and get used to the whole platform and CTF environment, since it is, in most of the cases, the first time making contact with this kind of competitions;
2) Competition Day (middle Jan 2018): this part will last four hours, from 2PM until 6PM, where all the teams must gather in the faculty Networking Lab to solve the remaining challenges. These ones will consist on more

difficult problems. After this part, the winning team will be announced and the competition ended.

VI. Competition Rules

Generally there are a set of rules applied in all CTF competitions which consist in:

1) Teams must be 100% independent. Sharing keys or giving any source of help to other teams is considered cheating;
2) Attacking the competition infrastructure such as exploiting possible platform breaches, performing Denial of Service or brute-forcing flag inputs it is forbidden and will be considered as cheating as well;
3) Sabotaging other teams in any way it is forbidden;
4) In case of breaking any of these rules, the team will be immediately disqualified.

In different competitions, variants or other rules can be applied depending a lot on the competition scheme itself.

VII. Challenges Scoring

Teams earn points by solving the challenges created but because some challenges can be more easily solved than others, a scoring system was used. The platform we used provides us with a feature for giving a set of points for each challenge so we divided the challenges in three categories according to the difficulty, easy, medium and hard.

For the easy challenges we chose values between 100 and 200 points where we introduce easy concepts according to the type, for the medium ones, we set the bar between 200 and 300 points and are introduced more concepts, still in a more basic level and finally for the hard ones the points are over 400 for each with advanced concepts where students will take more time and will need to do more research/work to solve them.

VIII. Hardware and Software

We decided to use the host application suggested by the teacher. Facebook CTF (https://github.com/facebook/fbctf) was created by Facebook Inc. and it is an open source platform, which in our case, was good because we can adapt it to our own needs. The whole platform is hosted in a local server in the Networking Laboratory at our faculty. The whole setup and maintenance was done by the laboratory technician.

The server that hosts the whole platform it is on the laboratory network and it has a public IP address which is accessible from everywhere. It is running Ubuntu 16.04 as the operating system, has a dual core processor virtualized out of a Xenon microprocessor and 4GB of DDR3 RAM memory. Furthermore, the server is backing up the whole virtual machine image weekly to make sure all the content is safe from any kind of major failure.

This specifications must be, in our opinion, more than enough to host a competition of this dimension without any resource problem. Even so, we plan to monitor closely the whole activity to make sure the competition is running smooth and without any kind of problem.

IX. Social Component

This component it is very important for competitions like this and in our case specially.

Since we started our 5 year degree there weren't many competitions or events regarding cyber security. We find that organizing a CTF can be very helpful concerning that there is, in our opinion, a growth of interest from students in this field.

A CTF competition like this, where students will try and discover new concepts, can be the major key to motivate newcomers to this branch of our degree. The whole social part where students will interact with each other can originate new projects and ideas that can be developed in the future.

X. Competition Sponsorship

Since it was the first time that a project like this was beeing developed in the recent history of our faculty, we found that sponsors from external identities were not necessary or justifiable in this case.

Despite this fact, we believe that in future editions of this competition sponsorships can be important.

Firstly, to motivate more students to join initiatives like this, awards can be given to the best teams.

Secondly, to improve the social component. Competition advertisement, coffee breaks or other sort of social activities can represent a huge difference in this kind of competitions.

Lastly, since we are in the last year of our Master degree, companies that work in cyber security field can show presence and interest in this competitions to understand students capabilities and maybe recruit future employees to their firms.

XI. Aftermath

This report was written before the whole competition taking place so, there is not much that can be written at this point. Even so, we suggested to the course responsible teacher to write, when we had the time to, some aftermath facts once the competition was done. This will make us discuss things such as what went wrong during the competition, implemented mitigations, participants feedback and suggestions, general opinion on the whole project and how beneficial it was for this course syllabus.

XII. Conclusion

Cyber security is one of the computer science fields with more growth in the last few years. If we look closely, there is not many tools and information that are not somewhat connected to the internet. More than ever, people are worried and interested in cyber security to keep the privacy of their information.

Developing the challenges was the most time demanding part of this project since it made us study and understand a vast number of different concepts that we never studied before this course. In contrast with all the other suggested projects, that were focused in a more narrow subjects, a CTF can be very broad, starting, for example, by all the different challenges types.

The chance of organizing something like a CTF competition and having it as the course final project is, in our opinion, something very positive. It gave us the ability to learn most of the basic security concepts and the insight of organizing a CTF competition. Although it was targeted to people with barely no knowledge in this area, we found that, by doing this project, we understood the complexity and dimension of this area and its importance nowadays.

We strongly believe that keep this competition on next editions of this course will be a big plus to the students knowledge and interest in this area.

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] Ford, Vitaly & Siraj, Ambareen & Haynes, Ada & Brown, Eric. (2017). Capture the Flag Unplugged: an Offline Cyber Competition. 225-230. 10.1145/3017680.3017783.

[3] L. McDaniel, E. Talvi and B. Hay, "Capture the Flag as Cyber Security Introduction," 2016 49th Hawaii International Conference on System Sciences (HICSS), Koloa, HI, 2016, pp. 5479-5486.

[4] M. Sanchez Rubio, G. Lopez Civera and J. J. Martinez Herraiz, "Automatic Generation Of Virtual Machines For Security Training," in IEEE Latin America Transactions, vol. 14, no. 6, pp. 2795-2800, June 2016.