Cybersecurity Paper Compilation
FEUP – Fall 2018

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

***Abstract*—This is the second compilation of cybersecurity papers authored by master-level network engineering students at FEUP and developed during the Fall of 2018. This year the compilation has a strong emphasis in malware analysis – on sandboxed environment, from binary and source code, focusing on C2C traffic and other runtime features, targeting IoT, RAT, and PowerShell. Other topics include IDS and SIEM, vulnerability scanning, open source intelligence, and looking at cybersecurity as a game for DoS and vulnerability exploiting and patching.**

I. Topics

Malware is increasingly complex and nefarious and half of the papers in this compilation focus on this. We start this compilation with three papers on sandboxing with Cuckoo – one looking at the evolution of a particular ransomware, the other focusing on sandbox detection avoidance of a banking trojan, and the third on the detection capability of Cuckoo using an open source trojan. Two papers then follow focusing on the Mirai IoT malware, one analyzing its source code and the other its traffic. We close the set of malware-related papers with an exploration of how Suricata can be configured to detect PowerShell Empire command and control traffic.

The second part of this compilation is more diverse. It starts with a paper on the impact of programmed vulnerability scanning on the page load time, then moves to a paper that explores existing IDS and SIEM architectures and to a paper that applies open source intelligence methods to collect information on an organization as a preparation step for penetration testing. The last two papers set examples of how to look at cybersecurity as a game where defender and attacker actions are relatively well established – the first of these papers considers vulnerability patching and exploitation as defense and attack actions, while the second paper considers request timeout threshold and number of attack connections per second on a web server under DoS attack.

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 DoS cybergame paper has not only taken the challenge of modeling a specific DoS attack as a game, defining specific actions on both sides, but also developed web server, attack, and normal clients that can be tuned according to the desired defender and attacker actions and a web page where the attacker and defender can specify and observe the results of their actions. The vulnerability scanning paper is also worth highlighting given the extent of the measurements that were performed, as are the three sandboxing papers given the complexity in setting up the Cuckoo environment and analyzing different malware.

The papers were not edited after submission and can have a significant number of errors – depending on the paper. Typos and formatting issues abound, and while most papers are written in English, some are in Portuguese, and one even has the abstract in English and the body in Portuguese. In any case, I believe they are worth publishing and hopefully other students can pick up where these have left off.

III. Paper List

Malware Analysis

- *Malware Sandboxing – Locky*, Nuno Moreira, Pedro Pinto, Pedro Cova, p.2
- *Sandboxing - Análise do Trojan.Emotet e Deteção de Máquinas Virtuais*, Artur Ribeiro, Hugo Guia, Joao Pedro Araújo, p.6
- *Open Source Remote Access Trojan Analysis – Pupy RAT*, Alejandro Molina Bermúdez, Miguel Sampaio, p.9
- *IoT Malware – Mirai Botnet Source Code Analysis*, Daniel Granhão, Joana Macedo, p.13
- *IoT Malware on the Wild – Mirai and friends*, Joana Catarino, Sara Costa, p.16
- *PowerShell Empire*, Anabela Reigoto, Renato Cruz, p.20

Operations, Intelligence, and Cybergames

- *Impact of WPScan as a Vulnerability scanning tool*, Miguel Ribeiro, Nuno Adrego, p.27
- *IDS & SIEM Architecture Review and Performance Assessment*, André Oliveira, Baltasar Aroso, p.30
- *Open Source Intelligence*, Pedro Silva, Kevin Vieira, Tiago Simões, not published given risk of sensitive content exposure
- *Cybersecurity as a Game: Apache Web Server*, Francisco Fraga, Gonalo Costa, João Pereira, p.34
- *Cyber Game Based on a DoS Attack*, João Aires, Miguel Barros, p.38

Malware Sandboxing: Locky

Nuno Francisco Moreira *, Pedro Soares Pinto† and Pedro Vila Cova‡
Department of Electrical and Computer Engineering
Faculty of Engineering of University of Porto
R. Dr. Roberto Frias, Porto 4200-465
Email: *nuno.francisco@fe.up.pt, †up201406009@fe.up.pt, ‡up201403635@fe.up.pt

***Abstract*—This paper aims to describe the study of malware, more specifically Locky, making use of sandboxing to provide a safer and controlled test environment. The tool utilised for this matter was Cuckoo Sandboxing. Throughout the analysis, six samples from different appearance dates, gathered from different online sources, were tested, as to understand the evolution of such ransomware better, being the same presented in the paper.**

I. Introduction

Malware is currently evolving faster than ever, in fact, in these evolving times, detecting and removing malware artefacts is not enough: it is vitally important to understand how they operate in order to understand the context, the motivations, and the goals of a breach. Since malware can be extremely harmful, it is imperative to select a safe environment to test them. Therefore, it was decided to use a sandbox, more specifically Cuckoo, which allowed to retrieve relevant information about the threat.

Locky is a very harmful ransomware with complex behaviour. It is believed that it was developed in Russia and finds its way to potential targets through emails. It can encrypt a wide variety of files, saving the encryption key in the server side, making it impossible to retrieve the files without the payment of a rescue.

As an analysis tool, Cuckoo allows for as much of a detailed analysis as the user might desire. It produces a wide variety of assessment files, including PCAPs, giving emphasis in the analysis main page to the most relevant files and operations that the malware compromised and the most relevant information about the same.

II. Related work

A. Cuckoo Sandbox

Cuckoo Sandbox is the leading open source automated malware analysis system and provides a detailed report outlining the behaviour of the file when executed inside a realistic, but isolated environment. It is a free software that automated the task of analysing any malicious file under Windows, OS X, Linux, and Android.

By default it is able to:

- Analyse many different malicious files (i.e., executables, office documents, pdf files, emails), as well as malicious websites under Windows, Linux, Mac OS X, and Android, virtualised environments.
- Trace API calls and general behaviour of the file and distils this into high-level information and signatures comprehensible by anyone.
- Dump and analyse network traffic, even when encrypted with SSL/TLS. With native network routing support to drop all traffic or route it through InetSIM, a network interface, or a VPN.
- Perform advanced memory analysis of the infected virtualised system through Volatility as well as on a process memory granularity using YARA [1].

Due to Cuckoo's open source nature and extensive modular design one may customise any aspect of the analysis environment, analysis results processing and reporting stage. Cuckoo provides all the requirements to easily integrate the sandbox into the existing framework and back end in the way the user wants, with the desired format. All without licensing requirements. [2]

B. Installing Cuckoo

Despite being an open source software, Cuckoo is somewhat challenging to install and configure. A significant portion of the time was devoted to configuring the test environment. First, a virtual box running Ubuntu 18.10 was set up and was given 4GB of RAM, 25 GB of disk space and a two core processor. In this machine, Cuckoo was installed following instruction found in [3].

It was set up to run a Windows 7 installation in a Virtual Box, and the installation is configured in order to keep the test environment clean for each test. With this double layer of insulation, both the host machine and the network are protected from propagation. In fact, the Windows 7 installation is only able to communicate with the Ubuntu 18.10 host through a virtual interface set up in host-only mode. After that, iptables was configured in order to allow communication between the Windows 7 test installation and the Internet. The test environment was set up in order to avoid the DHCP mechanism from delaying the start of the analysis with errors in the attribution of IP addresses. The test environment architecture is displayed in figure 1.

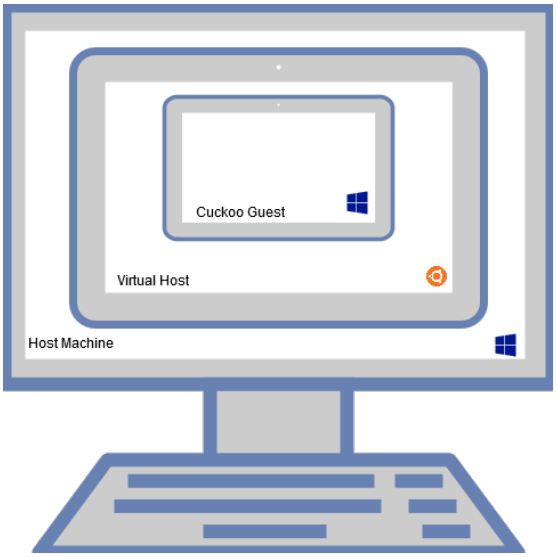


Fig. 1. Architecture

C. *Locky*

Locky is a ransomware malware released in 2016. It has advanced features such as domain generation algorithm, complex spam email campaigns, server-side encryption, and generic PE packers. Since its release, its authors have fine-tuned it and added features to make Locky even more difficult to detect.

The ransomware uses RSA-2048 + AES-128 cypher with ECB mode to encrypt files. Keys are generated on the server side, making manual decryption impossible, and it can encrypt files on all drives (fixed, removable, network and RAM). Payment varies between 0.5 and 1 bitcoin. It is also unfeasible to try to retrieve the files through shadow copies as an administrator, as the malware deletes them when launched as administrator. Some of the messages it displays are observable bellow.

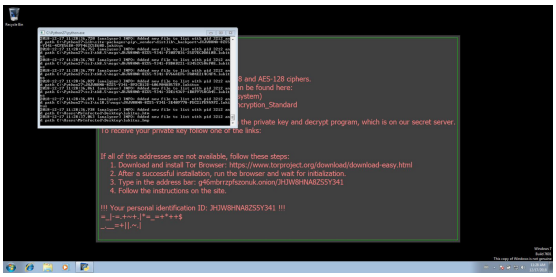


Fig. 2. Locky Ransomware

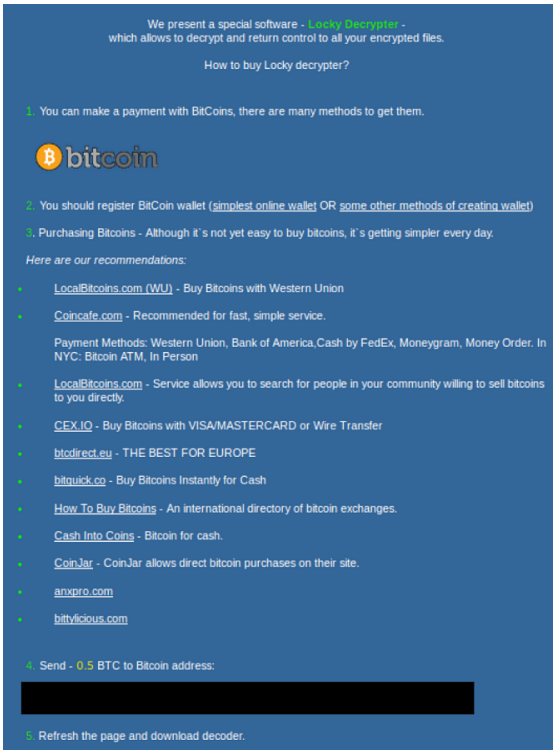


Fig. 3. Locky decrypter

It targets a wide range of file formats used by engineers, designers, developers and testers, being small businesses particularly at risk. In demographic terms, the top 10 countries hit by Locky are: France, Italy, Germany, Spain, USA, Great Britain, Poland, Japan, Czech Republic and Canada.

Spreading through fake emails and infected attachments, including .doc, .xls or .zip files, the user is asked to ”enable macro if data encoding is incorrect”, as the opened documents do not display correctly. This is a social engineering technique used to trick people because once they enable macros, a scrip downloads the Locky binary and executes it.

The infected emails are disguised to look like they are coming from legitimate sources, which makes them very difficult to recognise. Fake messages can have subject lines like ”Upcoming Payment - 1 Month Notice”, and allegedly contain an invoice, which is actually a Microsoft Word document with malicious macros. The infected user will be prompted to pay a ransom in exchange for the recovery key, and the message can be personalised, depending on the PC’s location.

Encrypted files are given a new extension, often named after gods of Norse and Egyptian mythology, such as ”.osiris”, ”.odin”, ”.thor”, ”.aesir” or ”.locky”.

Some believe that this malware is linked to a group of hackers with the name of Dridex, others that it comes from Russia, as many of its servers are there and because the ransomware is programmed to exclude Russian IPs from infection. [4]

The latest versions of the malware now include several detection-avoiding techniques, such as detecting whether it is

running within a virtual or physical machine, and relocation of the binary within the disk.

III. Solution

The system consists of a virtual machine that is running Cuckoo Sandbox, with a virtual machine with Windows 7 inside it, since the analysed malware was developed to it. With this setup, it is possible to perform tests to the threat, retrieving key information safely.

At first, the generated PCAP file was analysed in Wireshark. In the beginning, it was rather difficult to differentiate between the communications from the Guest Operative System and those from the malware activity. In order to overcome this constraint, it was used the Cuckoo automated report and filtered the captured traffic by the listed IP. After this, it was simple to understand and follow the communications to the C&C. From the tested samples it was possible to understand that, in general, the malware will try to communicate to a remote server using an HTTP POST request to "*remote server*/imageload.cgi", making use of encrypted parameters and encrypted values in those parameters. For the samples that did not receive a response to the SYN packet, a server was configured with the same IP address in order to receive the exchanged message.

After this initial communication, where it is supposed that the encryption key and victim information are sent, the malware starts to encrypt the computer files, the file termination is changed, and a rescue message is displayed as the background image. An HTML page is generated with the same rescue message as the background image. The rescue message explains how much bitcoins the victim must pay to receive the unlock code as well as how to proceed to the payment.

IV. Evaluation

During the project six samples of the Locky malware, with different checksums, were tested. The samples analysis is presented in chronological order, based on the date when the file was detected. These binary files were extracted using IDA [5], and it is visible that they are obfuscated using an unknown packer, so it is difficult to analyse the extracted binary. Using the DLL import information, gathered by the IDA disassembler, it is possible to infer the malware activity. Besides the standard Windows 32 DLL files (user32, kernel32, advapi32) the sample imports crypt32, certcli and shlwapi. These DLLs can be used to perform the encryption of the files, creation or deletion of certificates and provide Unicode functionality to legacy Windows editions, respectively. Regarding the detection-avoiding techniques, all of the samples did verify their surroundings. Either by verifying if they were being run on a debugger, by checking "IsDebuggerPresent" function from kernel32.dll, by detecting the presence of "VBoxMRXNP.dll" from VirtualBox and detecting Cuckoo's analyser script, "analyzer.py".

- The sample with the hash b02c55383f11e02e1564ca80386a6bc3 [6] was detected on March 28 of 2016. From the evaluation made we could assess that it tried to contact three different IP addresses (185.75.46.193, 82.118.17.219, 176.107.185.19), that did not respond, and also encrypted a handful of files, around 2300. The file extension used for the encrypted files was ".lukitus".
- File 1d30699c7b9ba6bce19543e3e9bf745a [7] was detected on August 21 of 2017. The only relevant difference from the previous version was that it tried to connect with IP addresses 176.107.185.19, 104.45.184.101 and 2.16.65.146, which were unsuccessful to respond.
- The behaviour of the file ba80256f80ef4339142ea19fa24b756a was very similar to the first one. This file was detected on August 21 of 2017.
- File 600ceb2492686011500da6fa8d5bfb4c [8] was detected on September 6 of 2017. It tried to connect with 193.233.60.199 and 91.230.211.76, of which only 91.230.211.76 was accessible. The payload on the HTTP POST request was different from the previous versions.
- For the file 86927f4d92665747679ab72a9be87b05 [9] two attempts to connect with 104.45.184.101 and 2.16.65.146 were detected, which were unsuccessful. Except for this, the behaviour was also very similar to the first file. This version was found on October 6 of 2017. The file extension used for the encrypted files was ".ykcol".
- Version dda37961870ce079defbf185eeeef905 [10] was detected on October 19 of 2017, and its behaviour was very close to the previous one. This time file extension used for the encrypted files was ".asasin".

All the IP addresses presented above were investigated and almost all were registered on Russia or Ukraine, which can prove the allegations made on the previous chapters about this malware having Russian origins. Relatively to some of this IP addresses, there are many complaints all over the Web about illicit use.

The IP address 176.107.185.19 was used more than once, the address was verified in Shodan.io [11], and it is possible to observe in figure 4 a short part of the report.

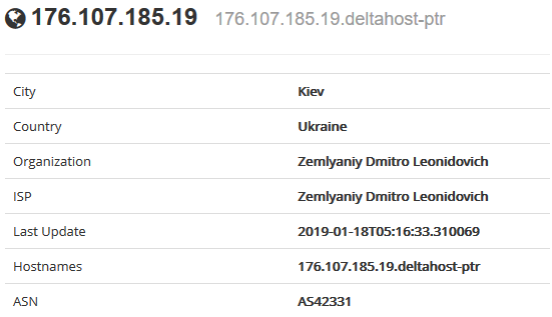


Fig. 4. Shodan Report

Note that the time between the first and second is more than one year. Still, the IP address 176.107.185.19 was used in both versions. That is a long time compared to the other IP addresses which denote a certain importance to the people who developed this malware. This importance can be due to the server cost or performance, or to the fact that the hosting company did not care about the illicit activity running under this server. Was also detected that after encrypting all files every version executes some commands that initiate Internet Explorer with the rescue message.

Even though six different files where evaluated, with a long time span between them, it was not possible to find a relevant difference in the behaviour of the Locky malware. The only detected variations were the IP addresses of C&C and the extension of the encrypted file that could be an aesthetic change or an indication of a different encryption method.

V. Conclusion

With the developed work the group was able to analyse the behaviour of Locky, retrieving essential information from its mode of operation, achieving the desired goal.

To understand how the malware has evolved, different versions of it were analysed, observing the diverse modes of operation, concluding that it has become more conscious about its surroundings and making use of different files, which makes it harder to understand and observe.

Cuckoo Sandbox played a vital role in the process of analysing the malware behaviour in detail, where the expected results were observed. With the PCAPs analysis, it was possible to detect the connections to the foreign servers as well as identifying some of the content being exchanged and from the Cuckoo's analysis main page, the moved and encrypted files and the overall action.

References

[1] VirusTotal. Yara documentation.
[2] Cuckoo Foundation. Automated malware analysis.
[3] Warunika Amali. Cuckoo sandbox installation guide, Jul 2017.
[4] Avast Software. Locky ransomware.
[5] Hex-Rays SA. Ida documentation.
[6] malekal.com. Locky sample 1.
[7] malekal.com. Locky sample 2.
[8] malekal.com. Locky sample 3.
[9] malekal.com. Locky sample 4.
[10] malekal.com. Locky sample 5.
[11] shodan.io. Shodan report.

Sandboxing - Análise do Trojan.Emotet e Deteção de Máquinas Virtuais

Artur Ribeiro
SSRE
FEUP - MIEEC
Porto, Portugal
Email: up201404880@fe.up.pt

Hugo Guia
SSRE
FEUP - MIEEC
Porto, Portugal
Email: up201305075@fe.up.pt

João Pedro Araújo
SSRE
FEUP - MIEEC
Porto, Portugal
Email: up201405315@fe.up.pt

***Resumo*—O *malware* encontra-se em constante evolução, tentando aproveitar novas vulnerabilidades para infetar sistemas e redes. A técnica de *sandboxing* permite analisar o comportamento de *software* malicioso, permitindo uma neutralização e remoção mais eficaz.**

Neste trabalho foi analisado o Trojan.Emotet, bem como capacidades de deteção de máquinas virtuais no geral.

Concluiu-se que o Emotet continua ativo, e que é possível contrariar a deteção de máquina virtual com a não instalação das VMTools na VM de *sandbox*.

I. Introdução e Trabalho Relacionado

A. Malware - Trojan

Malware é, por definição, qualquer *software* cuja intenção é causar dano a um computador, servidor ou rede de computadores. O nome nasceu de *"malicious software"* (*software* malicioso) e engloba diversas classes: vírus, *worms*, Trojans, *ransomware*, *spyware*, entre outras.

Dependendo do tipo de *malware*, existem diversos métodos de infeção.

Neste trabalho vamos abordar e analisar o Trojan.Emotet.

Um Trojan (ou *Trojan horse*) é qualquer tipo de *malware* que engana o utilizador sobre os seus objetivos. Os Trojans são geralmente difundidos através de *"social engineering"* (através de um anexo num e-mail, ou de um anúncio num website). Depois de infetar um computador, um Trojan pode roubar informações e credenciais, e também garantir o acesso não-autorizado ao sistema. Normalmente, os Trojans não tem capacidade de auto-propagação.

O Emotet [1] é um Trojan bancário - o principal objetivo é obter credenciais bancárias dos utilizadores através da interceção de tráfego de rede [2]. Após a infeção, que pode acontecer através de anexos maliciosos ou de auto-propagação numa rede interna, o Emotet tenta contactar um centro de Comando e Controlo (C&C), para onde envia a informação obtida e do qual recebe ordens. Ao contrário da maioria dos Trojans, o Emotet tem capacidade de auto-propagação, através de *brute-force* de palavras-passe, utilizando uma tabela de palavras-passe comuns , e também utilizando as vulnerabilidades EternalBlue/DoublePulsar [1]. Este método tem a agravante de poder bloquear os utilizadores das suas contas, o que aumenta o dano causado. Este *malware* sofreu evolução crescente [3]: primeiro aconteceu a transformação para uma estrutura modular [2]; ao longo do tempo foram acrescentados diferentes módulos com novas funcionalidades.

A versão 2 acrescentou a referida modularidade, incluindo módulos de transferência automática de dinheiro, *malspam* e um módulo bancário com alvo em bancos específicos (na Áustria e Alemanha).

A versão 3 introduziu capacidade de deteção de máquinas virtuais, o que dificulta a análise deste Trojan através de *sandboxing*.

A versão atual traduz uma mudança de paradigma no grupo responsável pelo Emotet (Mealybug), que passa de implementar o seu próprio *malware*, e foca em entregar *malware* de outros grupos. O Emotet infeta máquinas e a sua *payload* varia, o que leva, muitas vezes, a infeções mais rápidas devido à junção de auto-propagação de dois *malwares*.

B. Sandboxing

Para que seja possível analisar um *malware*, sem que se seja alvo de um ataque deste, é a utilização de um mecanismo de segurança de forma a isolar o *software* a testar do resto de programas ou processos que poderão ser vulneráveis a este.

Conhecido por *Sandboxing*, este mecanismo consiste na execução do *software* a testar num ambiente restrito, de forma a controlar os recursos que o *malware* possa utilizar ou tentar aceder, como memória, descritores de ficheiros, sistema de ficheiro, entre outros. Entre as varias implementações de *Sandboxing* existentes, a mais relevante para este trabalho é o *Sandboxing* em Maquina Virtual, consistindo num emulador de sistema operativo que pode ser executado no sistema operativo nativo do sistema utilizado sem que os diferentes sistemas operativos partilhem recursos e assim restringindo os recursos que o sistema operativo que é executado no emulador.

A analise do *malware* pode ser feita por sistemas de analise automática que abrem, executam e monitorizar ficheiros e obtém informação sobre estes.

Existem dois tipos de abordagens de analise, sendo estes o estático e o dinâmico. Estas duas abordagens distinguem se pela interação que o sistema de analise utiliza para analise do software ou ficheiro em questão, sendo a primeira abordagem uma analise binária do ficheiro sem que este seja executado e

a segunda abordagem uma analise em tempo real do ficheiro ou software em execução.

No âmbito de *Sandboxing*, a abordagem dinâmica é claramente utilizada, visto que o seu objetivo é monitorizar as ações do ficheiro analisado.

Um dos sistemas de análise *Sandboxing* mais conhecidos e utilizados é o *Cuckoo Sandbox* [4], um sistema *open source* que é capaz de rastrear chamadas ao sistema e tráfego de rede, detetar *Memory dumps* de processos ou maquinas, *Screenshots* tirados durante a execução do *malware* e criação, eliminação ou *downloads* de ficheiros.

A arquitetura deste sistema de analise consiste num programa que controla uma ou varias maquinas virtuais onde o software a analisar é executado, sem que este tenha acesso a processos da maquina onde este sistema está a ser executado.

C. Deteção de Máquinas Virtuais

1) Deteção: Devido à utilização de *sandboxing* para analisar *malware*, alguns grupos responsáveis pela criação de *software* malicioso introduziram capacidades de deteção de máquinas virtuais (VM). Esta deteção pode ser efetuada de diferentes formas [5] [6], incluindo:

- **Registry check**: entradas de registo criadas ao iniciar uma máquina virtual [5];
- **Memory check**: localização de estruturas de memória (nomeadamente, IDT (Interrupt Descriptor Table)) é diferente numa máquina virtual, devido ao mapeamento direto na memória física [5];
- **Communication Channel check**: utilização de instruções priveligiadas (IN), que geram uma exceção se utilizadas com um utilizador não privilegiado num sistema normal, mas não geram se utilizadas numa máquina virtual [5];
- **Processes and Files check**: processos específicos de VMs e ferramentas instaladas para facilitar o seu uso, que podem ser detetados através de *hashes* ou de procura direta de nomes de processos [5];
- **MAC check**: os primeiros bytes do endereço MAC identificam o fabricante da placa de rede. No caso de VMs, permitem identificar que está a ser corrida uma máquina virtual [5];
- **Hardware check**: os identificadores e outros parâmetros de *hardware* podem denunciar a existência de uma VM [5].

2) Evitar Deteção: Visto que é possível um *malware* detetar se está a ser corrido numa máquina virtual e, possivelmente, estar a ser alvo de *sandboxing*, este pode mudar o seu comportamento [2]. Exemplos disto são a não desencriptação e instalação do *malware/payload*; contacto de centros C&C falsos.

Sendo que esta mudança de comportamentos pode dificultar ou até impossibilitar o estudo de um *software* malicioso, é importante conseguir combater esta deteção. Isto é possível, utilizando diversas formas:

- **Interceção de chamadas à API**: grande parte dos métodos de deteção de VM podem ser contrariados com a interceção de chamadas à API do sistema [5]. Isto pode ser feito através de [7]:
  - Detours: *software* disponibilizado pela Microsoft que permite, de forma simples, intercetar chamadas à API do sistema e retornar uma resposta customizada, através da modificação da função original;
  - Syringe: *software* que permite modificar as respostas de uma função da API do sistema, modificando as *import entries* da função-
- **Hardware check**: é possível contrariar a deteção através de *hardware* ao alterar os nomes e restantes parâmetros dos elementos de *hardware* presentes no sistema da máquina virtual;
- **Processes and Files check**: a não instalação de ferramentas e ficheiros específicos de VM pode impedir a deteção por parte do *malware*.

II. Solução

Ainda que o software Cuckoo, como referido previamente, forneça um relatório detalhado acerca da atividade do *malware* no interior da *sandbox*, a utilização do Cuckoo no nosso caso serviu apenas por uma questão de blindagem e isolamento de forma a poder analisar o comportamento do Emotet sem por em causa a segurança do sistema operativo de *host*.

A nível de hardware, real e virtualizado, a configuração foi a seguinte: no *host* estava instalado Windows 10; utilizando o software VirtualBox foi instalado Ubuntu 18.04. O Cuckoo foi executado em ambiente Ubuntu virtualizado que continha nele instalado uma cópia de Windows 7 instalada em VirtualBox. Foi nesta sub-máquina virtual com Windows 7 onde o Emotet foi executado. Esta configuração pode ser visualizada em 1.

Visto que o intuito era constatar a reação do *malware* em ambiente virtualizado para observar se o mesmo estava a comportar-se de forma diferente, a análise decorreu através do software *Wireshark*, capturando pacotes de rede e analisando os endereços IP's.

Em situações normais, quando infecta um certo host, o *malware* entra em contacto com um centro de Comando e Controlo enviando informações relativas à máquina infetada. Contudo, utilizando mecanismos referidos acima, se detetar que está a ser executado numa máquina virtual, o contacto é feito para endereços IP falsos de forma a enganar quem estiver a tentar analisar o seu processo de funcionamento.

A nossa análise foi baseada nesse conceito: na análise dos contactos feitos antes e após tentar transformar a máquina virtual em máquina real utilizando as técnicas mencionadas previamente. Foram testadas diversas versões do malware que foram executadas durante 10 minutos. Simultaneamente, o Wireshark registava toda a atividade de rede da placa de rede virtual e foi nessa análise que foram encontrados os endereços IP's de C&C. Esses mesmos endereços foram alvo de pesquisa relativamente à sua localização, actividade relacionada com os mesmos e portas abertas.

Figura 1. Configuração utilizada para *sandbox*

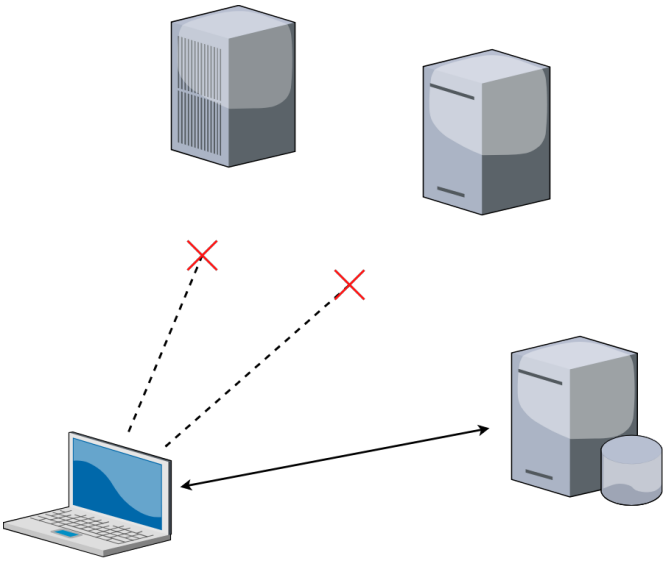


Figura 2. Tentativas de contacto do Emotet para endereços IP falsos

III. Avaliação dos Resultados

O foco foi colocado na determinação das capacidades de deteção de ambiente virtual do Emotet. Desse modo, foram analisadas três versões do Trojan.Emotet, e obtidos os seguintes resultados I, referentes à comunicação com os centros de Comando e Controlo.

Como a versão 1 é antiga, os seus centros C&C estão desativados.

Na versão 2, é possível verificar que são contactados endereços IP diferentes quando as VMTools estão instaladas ou não.

Na versão 3 testada, os endereços IP não estão (ainda) associados a atividade maliciosa.

Para verificar o estado de cada endereço, foi utilizado o Feodo Tracker. Este é um *website* dedicado a divulgar centros C&C associados aos *malwares* Emotet e Dridex.

Após a constatação que portas relativas a HTTP se encontravam abertas nos endereços analisados, foi usado um browser para aceder a esses mesmos endereços e foram encontrados

Tabela I
Endereços IP obtidos e analisados

Versão	IP C&C	Estado	Nota
1	212.83.146.230	Offline	
1	62.210.86.114	Offline	
2	119.59.124.163	C&C Ativo	
2	188.126.72.179	C&C Ativo	
2	192.163.239.60	Online	
2	118.244.214.210	C&C Ativo	Sem VMTools
2	146.185.170.222	C&C Ativo	Sem VMTools
2	187.250.146.185	Online	Ativ. Maliciosa
3	47.205.41.43	Offline	
3	198.199.185.25	Online	

websites de fachada, com informação aleatória referente, por exemplo, a agências de viagens em França.

IV. Conclusão

Conclui-se que o *malware* Trojan.Emotet consegue detetar se se encontra a correr numa máquina virtual, a partir da versão 2. O Emotet realiza esta deteção analisando ficheiros, diretórios e processos relacionados com a execução dentro de uma VM. É de importância notar que um passo simples para evitar que um *malware* detete uma máquina virtual passa pela não instalação das VMTools.

Referências

[1] "Emotet malware – an introduction to the banking trojan." [Online]. Available: https://www.malwarebytes.com/emotet/
[2] A. Shulmin, "The banking trojan emotet: Detailed analysis." [Online]. Available: https://securelist.com/the-banking-trojan-emotet-detailed-analysis/69560/
[3] "The evolution of emotet: From banking trojan to threat distributor." [Online]. Available: https://www.symantec.com/blogs/threat-intelligence/evolution-emotet-trojan-distributor
[4] "Automated malware analysis." [Online]. Available: https://cuckoosandbox.org/
[5] "How malware detects virtualized environment (and its countermeasures)," May 2017. [Online]. Available: https://resources.infosecinstitute.com/how-malware-detects-virtualized-environment-and-its-countermeasures-an-overview/
[6] "Avoid malware's vm detection with antivmdetection," Nov 2016. [Online]. Available: https://www.andreafortuna.org/cybersecurity/avoid-malwares-vm-detection-with-antivmdetection/
[7] S.-W. Kim, "Intercepting system api calls," Mar 2012. [Online]. Available: https://software.intel.com/en-us/articles/intercepting-system-api-calls

Open Source Remote Access Trojan Analysis Pupy RAT

Alejandro Molina Bermúdez
up201802023@fe.up.pt
Master in Electrical and Computer Engineering
Faculty of Engineering of the University of Porto

Miguel Angelo Pinto Sampaio
ee12065@fe.up.pt
Master in Electrical and Computer Engineering
Faculty of Engineering of the University of Porto

***Abstract*—The purpose of this study is to identify the structure and analyze the behaviour of an open source Remote Access Trojan called Pupy. The analysis comprehends the implementation of the RAT in a sandbox environment so statistics can be obtained. The information collected shows which points are vulnerable and can be used to find ways to strengthen the defense against this type of malware. Even though RATs are always improving in finding a way of not being detected, this is a good approach to know how they work and what are their favorites targets to exploit.**

I. Introduction

This paper will resume the work done in the main project of the curricular unit Security for Networks and Systems of the Master in Electrical and Computer Engineering where it was proposed that we analyzed an open source Remote Access Trojan in a sandbox environment with the main goals of analyzing the code and the characteristics of this type of malware and test it in a sandbox environment provided by Cuckoo Sandbox.

Like any other malware, Remote Access Trojans, or simply RATs, are malicious programs that were written with the intent of doing harm to people, their devices or the data present on those devices. They are usually downloaded invisibly when the user tries to make a requests to an infected program or sent as an email attachment. This type of malware runs invisibly on the targeted computer and allows an intruder to have remote access to the infected resources through a backdoor for administrative control which provides unlimited access to infected endpoints and compromises sensitive and personal data [3], [5], [13]. Some of the main functionalities that RATs provide to the intruders are:

- Monitor users through keyloggers and other spywares;
- Access confident information like credit card and social security numbers;
- Use the webcam, take pictures and record videos;
- Take screenshots;
- Distribute other infected programs;
- Establish a botnet;
- Delete, download or alter any type of files.

Advanced Persistent Threat (APT) are attacks in which a person or a group in stealthy computer networks gain unauthorized access to a network and remain undetected for an extended period. The majority of this attacks take advantage of RATs to recognize the system, bypass strong authentications, spread the infection, and access sensitive applications to infiltrate or exfiltrate data. Some of the most known RATs to this date are the DarkComet, Poison Ivy, Back Orifice, Mirage and Bifrost.

The idea of testing this type of malware in a controlled environment is to analyze what malicious files it executes, which connections it establishes, trace API calls and evaluate the behavior of the file to translate this data into high level information comprehensible by anyone. This information can be associated with an analysis to the network traffic to achieve a better understanding of the infected virtualized system.

II. Related work

There are a lot of works that talk about malware, malware analysis and sandboxing. The way to create and use countermeasures like black list and white list techniques against RATs without disrupting business activities is the main focus of the paper *"Development of white list based autonomous evolution of defense system for rat malware"*[11], while the paper *"Evaluation of a brute forcing tool that extracts the rat from a malicious document file"*[8] focus more on how to embed a RAT to a document and in the development of a brute force tool to decode obfuscation and extract the RAT from a malicious document. How to use sandboxing technology to detect malware samples and how to determine their behaviour by examining the data collected by Cuckoo Sandbox are the main focuses of the papers *"Dynamic malware analysis using cuckoo sandbox"*[4] and *"Windows malware detection based on cuckoo sandbox generated report using machine learning algorithm"*[12].

III. Solution

To create the sandbox environment it was used a program called Cuckoo Sandbox[1] which is an extremely modular and open source automated malware analysis system that is able to:

- Analyze different types of malicious files and websites on Windows, Linux, Mac OS X, and Android virtualized environments;
- Trace API calls and characterize the general behavior of the malware;

Figure 1. Example of an email containing phishing.

Figure 2. Example of an holiday greeting cards containing malware.

- Dump and analyze all network traffic generated in the virtualized environment;
- Perform memory analysis of the infected system.

After analyzing the open source RATs available on the web it was made the decision to analyze Pupy RAT[9]. Pupy is an open source multi-platform (Windows, Linux, OSX and Android) post-exploitation tool mainly written in python that is used by advanced persistent threat (APT) groups. It was first reported in mid-2016 in a campaign of espionage attacks called Magic Hound that targeted Middle East organizations connected to the financial, oil, and technology sectors. The alleged attackers of that campaign were an Iranian-government backed APT group called *"Rocket Kitten"*. The attackers used phishing emails containing attached documents with malicious macros that, once enabled, would call Windows PowerShell to retrieve additional tools. This attachments were disguised as communications from Middle East governments, as in figure 1, as well as holiday greeting cards, as in figure 2. [10][14].

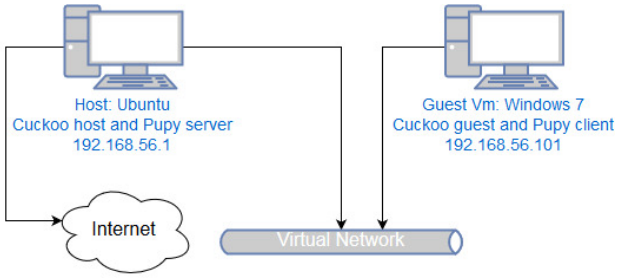


Figure 3. Setup used for the experiments.

To test Pupy and run it in an isolated environment we created a setup using VirtualBox with the host running on Ubuntu. This host is connected through a Virtual Network to a guest virtual machine running on Windows 7, as described on figure 3. The host has the Cuckoo host which connects and analyzes what happens on the guest through a file called agent.py while also being the Pupy server. With the help of the Cuckoo Sandbox documentation [2] and some guides found on the web [6] we created the script present on Listing 1 which allows anyone to install everything that you need to run Cuckoo on Host. On the guest we statically defined the IP address(192.168.56.101), the default gateway(192.168.56.1) and the default DNS servers(8.8.8.8 and 8.8.4.4), we placed the agent.py file into the startup folder so it runs every time the Windows starts and then created a snapshot of this virtual machine so Cuckoo can use the virtual machine and reset it to its previous state. To create the Pupy RAT executable file we need to install Pupy on the Ubuntu host, both the installation process and the RAT payload creation can be found on Listing 2.

Listing 1. Script used on Host for Cuckoo.

```
#!/bin/bash
apt-get update && apt-get upgrade

#install dependencies and Cuckoo
apt-get install virtualbox -y
apt install virtualbox-ext-pack -y
apt-get install python python-pip \
python-dev libffi-dev libssl-dev -y
apt-get install python-virtualenv \
python-setuptools -y
apt-get install libjpeg-dev \
zlib1g-dev swig -y
apt-get install mongodb -y
apt-get install postgresql \
libpq-dev -y
apt-get install qemu-kvm libvirt-bin \
ubuntu-vm-builder bridge-utils \
python-libvirt -y
apt-get install python-pip -y
pip install XenAPI
apt-get install tcpdump -y
setcap cap_net_raw,cap_net_admin=eip \
```

```
/usr/sbin/tcpdump
getcap /usr/sbin/tcpdump
apt-get install swig -y
pip install m2crypto==0.24.0
apt install libguac-client-rdp0 \
libguac-client-vnc0 libguac-client-ssh0\
guacd -y
pip install -U pip setuptools
pip install -U cuckoo
virtualenv venv
. venv/bin/activate
pip install -U pip setuptools
pip install -U cuckoo
cuckoo -d
iptables -t nat -A POSTROUTING -o eth0\
-s 192.168.56.0/24 -j MASQUERADE
iptables -P FORWARD DROP
iptables -A FORWARD -m state --state \
RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -s 192.168.56.0/24\
-j ACCEPT
iptables -A FORWARD -s 192.168.56.0/24\
-d 192.168.56.0/24 -j ACCEPT
iptables -A FORWARD -j LOG
echo 1 | sudo tee -a \
/proc/sys/net/ipv4/ip_forward
sysctl -w net.ipv4.ip_forward=1

#copy the agent file to the shared
#paste
cp ~/.cuckoo/agent/agent.py\
~/desktop/ssre

#run Cuckoo
cuckoo web runserver
cuckoo -d
```

Listing 2. Script used on Host for the malware.

```
#!/bin/bash
#Install dependencies and Pupy
apt-get install git libssl-dev \
libffi-dev python-dev python-pip \
tcpdump python-virtualenv
git clone --recursive \
https://github.com/n1nj4sec/pupy
cd pupy
python create-workspace.py -DG pupyw
export PATH=$PATH:~/.local/bin
git submodule init
git submodule update
pip install -r requirements.txt
wget https://github.com/n1nj4sec/\
pupy/releases/download/latest/\
payload_templates.txz
tar xvf payload_templates.txz
mv payload_templates/* \
pupy/payload_templates/
rm payload_templates.txz
rm -r payload_templates

#Create the RAT for deployment
cd /path/to/pupy/pupy
./pupygen.py -l | less -R
./pupygen.py -f client -O \
windows -A x86 connect \
--host 192.168.56.1:443 -t ssl

#Run the server
./pupysh.py
```

IV. Evaluation

The tests made to Pupy were the following:

- Take a simple screenshot of the targeted screen;
- Execute an established command on a python shell;
- Execute an established command on a shell;
- Run an interactive shell where you can run commands on the server side;
- Run module lazagne, which retrieves all passwords stored on the targeted computer;
- Check if it is running on a virtual machine environment;
- Get the IPs and MACs of all interfaces;
- Get the ppid, pid, user id and netstat;
- Run module beroot, which checks for a privilege escalation path in the system;
- All of the above at the same time.

In all of this tests the targeted computer would connect to the server and then the server would execute the commands needed for the completion of the test and then close the connection. The results obtained for each one of these tests can be seen in figure 4. On the left side is presented a list of the tests and on the right side the score obtained in the Cuckoo web interface, this score should be between 0 and 10 but the scoring system is still under development and should be considered an alpha feature.

By analyzing the results we can divide the tests made in 3 groups, group A which contains the tests with score lower than 4, group B which contains the tests with score between 4 and 8 and group C which contains the tests with score higher than 8.

All the tests in group A either retrieve a very small file or execute few commands that only retrieve a couple of lines and even though they can generate a varying number of processes (1 to 40 processes) and use different amounts of memory, they all have in common the fact that they have seven yellow flags and only one red flag, which is the fact that they look for the Windows Idle Time to determine the up time, this flag is common to all tests made, even in other groups. This type of tests do not try to change any information on the system and do not access any important processes, thus the scores are low.

Group B tests can also generate a varying number of processes (2 to 6) but they either generate more yellow flags (7

Test	Blue flags	Yellow flags	Red flags	Score
Take a screenshot	2	7	1	3.8
Execute a command on a python shell	2	7	1	3.8
Execute a command on a shell	2	10	1	5
Run an interactive shell	2	8	8	8.8
Run module lazagne	2	8	4	6
Check if it is running on a virtual machine	2	7	2	4.4
Get the IPs and MACs of all interfaces	2	7	1	3.8
Get the ppid, pid, user id and netstat	2	7	2	4.4
Run module beroot	2	8	4	6
All of the above on a single connection	2	13	16	15.6

Figure 4. Results of the tests.

to 10) or more red flags than group A tests (from 1 to 4 flags, the higher the score the higher the number of flags). When the malware checks if it is running on a virtual machine more than getting the same flag that all the group A tests got, it also checks the version of the BIOS for anti-virtualization. After checking that, it returns the information that the malware is running on a VirtualBox virtual machine to the server. The test that gets the ppid, pid, user id and netstat got 5 events where the malware expressed interest in specific running process. Those processes where the svchost.exe, which is a generic host process name for services that run from dynamic-link libraries, the lsass.exe which is the Local Security Authority Subsystem Service, the wininit.exe that primarily acts as a launcher for background applications that are running on the system (this is a common process which malwares use to disguise), services.exe, used to manipulate other programs and wmpnetwk.exe which is responsible for network sharing. Running the lazagne module will provoke 4 flags in the Cuckoo analysis being them the up time, the harvest of credentials and other information from local FTP client softwares, instant messaging clients and local email clients. The beroot module will enumerate services, collect information about installed applications and detect VirtualBox through the registry keys to try to achieve a path to root.

For the remaining two, the test where the malware tried to create a dynamic shell got 8 flags with the number of events going from 1 to 28. This flags are mainly related to code injection and memory manipulation hence the higher score. As expected, the test where all other tests were run in a single session was able to achieve the highest score and the highest amount of red flags (16).

Another aspect that is common among all this groups of tests is the fact that the malware tries to establish a connection with more than 450 websites from amazon.fr to MTV or even Wikipedia. The reason to this is either the fact that the malware tries to disguise one connection that we weren't able to identify or the fact that it knows it is being put on a sandbox environment and so it tries to create as much false data as possible. After running all of the websites on a platform for testing website security [7] we didn't obtained any type of malicious software warnings so it is more likely to be just false data.

V. Conclusion

Even though we were not able to deeply analyze the code of the malware because everything used encryption, the flexibility of this RAT allowed us to test its performance in different ways of building the payloads, although the results remained the same. The data collected in this work should help to create patterns and understand the behaviours and target selection of RATs.

In future works it should be taken into account the possibility of creating default modules to test more capabilities of the malware and place the Pupy server on a separate computer to see if it influences the scores obtained. To better understand the data collected, this data should be crossed with data collected from other RATs for better pattern establishment and behaviour understanding.

The work done during this semester helped us to understand much better how malware works, specially RATs, and how nefarious they can be to a system. These knowledge coupled with the ability to create a structured and isolated environment by using VirtualBox and Cuckoo will be really important and of much use if we decide to keep studying and working with cyber-security.

References

[1] Cuckoo. Cuckoo sandbox. https://cuckoosandbox.org/. Last accesses 2019-01-17.

[2] Cuckoo. Cuckoo sandbox book. https://cuckoo.sh/docs/. Last accessed 2019-01-16.

[3] IBM. Remote access trojan (rat). https://www.trusteer.com/en/glossary/remote-access-trojan-rat. Last accesses 2019-01-17.

[4] S. Jamalpur, Y. Navya, P. Raja, G. Tagore, and G. Rao. Dynamic malware analysis using cuckoo sandbox. pages 1056 – 60, Piscataway, NJ, USA, 2018.

[5] K. Lab. Kaspersky lab encyclopedia: Remote access trojan (rat). https://encyclopedia.kaspersky.com/glossary/remote-access-trojan-rat/. Last accessed 2019-01-16.

[6] Medium. Cuckoo sandbox installation guide. https://medium.com/@warunikaamali/cuckoo-sandbox-installation-guide-d7a09bd4ee1f. Last accesses 2019-01-17.

[7] T. MICRO. Site safety center. https://global.sitesafety.trendmicro.com/. Last accesses 2019-01-18.

[8] M. Mimura, Y. Otsubo, and H. Tanaka. Evaluation of a brute forcing tool that extracts the rat from a malicious document file. pages 147 – 54, Los Alamitos, CA, USA, 2016//.

[9] n1nj4sec. Pupy github repository. https://github.com/n1nj4sec/pupy. Last accesses 2019-01-17.

[10] NJCCIC. Pupy. https://www.cyber.nj.gov/threat-profiles/trojan-variants/pupy. Last accesses 2019-01-17.

[11] T. Shigemoto, S. Fujii, I. Kuriama, T. Kito, H. Nakakoji, Y. Fujii, and H. Kikuchi. Development of white list based autonomous evolution of defense system for rat malware. pages 95 – 101, Los Alamitos, CA, USA, 2018//.

[12] S. Shiva Darshan, M. Kumara, and C. Jaidhar. Windows malware detection based on cuckoo sandbox generated report using machine learning algorithm. pages 534 – 9, Piscataway, NJ, USA, 2016.

[13] TechTarget. What is a rat (remote access trojan)? https://searchsecurity.techtarget.com/definition/RAT-remote-access-Trojan. Last accesses 2019-01-17.

[14] UNIT42. Magic hound campaign attacks saudi targets. https://unit42.paloaltonetworks.com/unit42-magic-hound-campaign-attacks-saudi-targets/. Last accesses 2019-01-17.

IoT Malware - Mirai Botnet Source Code Analysis

Daniel Granhão
up201406280@fe.up.pt
Master in Electrical and Computers Engineering
Faculty of Engineering of the University of Porto

Joana Macedo
up201607759@fe.up.pt
Master in Electrical and Computers Engineering
Faculty of Engineering of the University of Porto

***Abstract*—The Mirai botnet started to gain recognition when it overwhelmed several high profile targets with massive distributed denial of service (DDoS) attacks. This malware targets IoT devices running Linux and turns them into "bots" - remote controlled devices that together are able to generate big quantities of traffic directed to a targeted victim.**

In this paper it's provided an analysis of Mirai's source code, more specifically on the vulnerabilities present. Furthermore, some of these vulnerabilities are exploited and arguments are made about the importance of finding them even on "ancient" malware.

I. Introduction

Since the appearance of Mirai the security landscape changed in significant ways as this malware highlighted the potential of IoTs as DDoS attack assets and how vulnerable devices are.

One of the most peculiar aspects of Mirai is the fact that it's source code was made public. The source code became available with instructions included and anyone could raise their own botnet. This created a new threat since attackers had an existing framework that they could modify to add behaviors and improve on work already done.

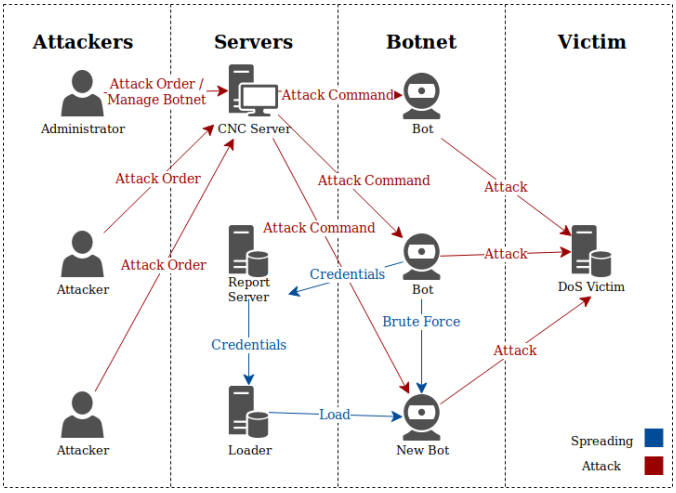


Fig. 1. Mirai's Architecture

Figure 1 represents the architecture of Mirai's operation. It's shown the different constituent nodes as well as the possible interactions between them.

The arrows in red represent the interactions associated with performing an attack. On the left column there are several attackers represented to highlight the possibility of the botnet being shared and even be monetized as a service. The attackers can send attack orders to the CNC server that forwards the same order to the bots and these proceed with the DDoS attack. The attack can be of several types and work on different levels of the OSI model. Some examples of available attacks are: UDP, SYN, TCP Stomp, among others.

The interactions associated with Mirai's spreading to a new bot are represented by blue arrows. All bots from the botnet are actively searching for new targets that offer a visible Telnet service and try to access them using brute force with a predefined set of credentials (default credentials of several IoT devices). When it's possible to access a new device, the bot that discovers the credentials sends them to the Report server which is listening to collect targets. The address and credentials are used by the Loader server that loads and runs the bot executable in the new IoT device. This is done by accessing the device through Telnet and using a tool like TFTP to download the executable.

Initially, the main goal of this work was to analyze Mirai's source code in order to understand and make clear how it works. During the analysis it became clear that Mirai was only designed to function and was developed without having defense in mind. The fact that some of the communications present in figure 1 use the Telnet protocol indicates how little concern has been given to the defense against possible attacks. With that in mind it was decided to explore and exploit some of Mirai's vulnerabilities. The code under observation is the originally made available and can be found in [1].

II. Related Work

There is some work done regarding the analysis of Mirai's source code such as in [2]. This specific paper sheds some light on the inner workings of different parts of the original version of Mirai by explaining, for example, how the CNC server handles attackers requests and how the bot competes for control of an IoT device. Furthermore, it's noted that there's possibly an SQL injection vulnerability in the CNC server API handler code but during our analysis none was found.

In [3] the propagation process is better explained in relatively detailed steps and a comparison is made between Mirai

and Hajime, a competing IoT malware also used to launch DDoS attacks.

III. Analysis

A. Vulnerabilities

To try and find vulnerabilities in Mirai's code the method used consisted on searching for a predefined set of vulnerabilities in each component of Mirai. The vulnerabilities found are detailed in the next subsections.

1) Missing Data Encryption: The communications between Mirai's components are as follows.

- Attacker $\Longleftrightarrow$ CNC server: Telnet;
- CNC server $\Longleftrightarrow$ bot: binary codification;
- Bot $\Longleftrightarrow$ new bot: Telnet;
- Bot $\Longleftrightarrow$ Report server: binary codification and ASCII;
- Report server $\Longleftrightarrow$ Loader: shared file;
- Loader $\Longleftrightarrow$ new bot: Telnet.

Everyone of them is missing any type of encryption and doesn't offer confidentiality or integrity. This leaves Mirai very vulnerable to anyone that could obtain access to some of the traffic. For example, credentials are sent in plaintext between attackers and the CNC server.

2) SQL Injection: No SQL injection vulnerability was found in Mirai. In the context of this malware this vulnerability only makes sense applied to the CNC's database. With the analysis of the corresponding code it was verified that the it was developed in order to prevent the occurrence of this vulnerability through the use placeholders to build the SQL statements.

3) Buffer Overflow: A buffer overflow vulnerability was found as stated in [4]. This article reveals a flaw in a segment of the code that carries out HTTP flood attacks that allows an adversary to stop the attack. No other buffer overflow vulnerabilities were found. Components whose code is written in the Go programming language such as the CNC server, are secure against this vulnerability as this language provides built-in protection against overwriting data in any part of memory and automatically checks that data written is within the boundaries of strings, arrays and slices.

4) Unvalidated Inputs: A bug was found in the interface available to an attacker in which the submission of an incomplete command (the character "@") crashes the CNC server, effectively disabling all of Mirai's operation. This happens due to missing validations that cause the program to try to access a nil object.

5) Application layer DoS: The report server simply listens on port 48101 for incoming raw TCP connections and receives addresses and respective credentials in plaintext. No integrity mechanism is used. This opens the possibility of an application layer DoS attack on this server that will reduce the loader efficiency. A small program that exploits this vulnerability is detailed in section III-B1.

B. Lab Experiment

A test setup was configured in order to observe the dynamic behavior of Mirai. Debug instances of the bot and of the CNC server were run on different machines and DoS attacks were launched on a victim server. An instance of the report server was also executed. Some of the vulnerabilities stated before were exploited.

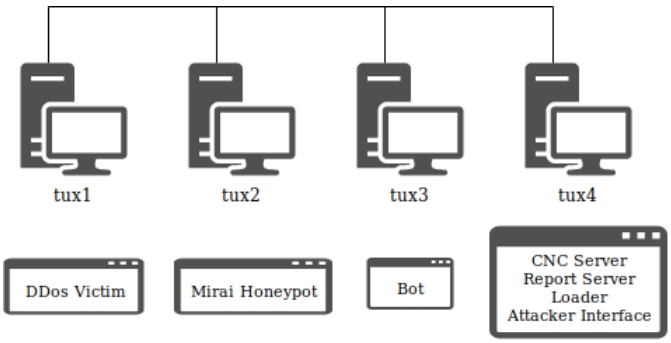


Fig. 2. Lab configuration

1) Application layer DoS: A simple program was coded to continuously send random fake IoT addresses and credentials to the report server. The server stores all reports received. When the loader uses this data, it wastes time trying to access possibly non existing devices with incorrect credentials.

2) Crashing CNC: An experiment was done to try and stop CNC operation from the point of view of an agent exterior to Mirai. Traffic between the attacker and the CNC server was captured using Wireshark in the lab and the attacker credentials were retrieved as they are sent in plaintext. Access to the CNC was accomplished using these credentials and entering the character "@" leads the CNC server to crash, as previously explained.

3) Fake bot: A fake bot was coded to communicate with the CNC server and behave as any other bot from the point of view of the CNC. When it received an attack order, it logged the event and didn't proceed with the attack. This allows the monitoring of the botnet of an adversary.

IV. Conclusion

As presented in this paper Mirai has a surprising number of vulnerabilities for such a notorious malware that was responsible for massive attacks. Although some of the vulnerabilities only decrease Mirai's performance, others have the possibility of completely inhibit it's operation. As Mirai's architecture is centralized (based on a CNC server), it presents a single point of failure. New botnets that were built upon Mirai improve on this aspect and are based on distributed architectures.

As previously stated, this analysis contemplates the original leaked code of the Mirai malware thus it's likely that current versions have been improved and the stated vulnerabilities removed. Nevertheless, it's interesting to realize how a novelty malware that's developed without much care about security can still be highly effective while it stays undiscovered.

References

[1] "Mirai-source-code," https://github.com/jgamblin/Mirai-Source-Code, accessed: 2018-10-12.

[2] G. P. J. D. Roger Hallman, Josiah Bryan and J. Romero-Mariona, "Ioddos the internet of distributed denial of service attacks," *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security (IoTBDS 2017)*, pp. 47–58, 2017.

[3] A. S. Georgios Kambourakis, Constantinos Kolias, "The mirai botnet and the iot zombie armies," *Milcom 2017 Track 3 - Cyber Security and Trusted Computing*, pp. 267–272, 2017.

[4] "Mirai vulnerability disclosed, but exploits may constitute hacking back," https://threatpost.com/mirai-vulnerability-disclosed-but-exploits-may-constitute-hacking-back/121644/, accessed: 2018-11-09.

IoT Malware on the Wild
Mirai and friends

Joana Whiteman Catarino
Security for Networks and Systems
Telecommunications, Eletronics and Computers
Master in Electrical and Computer Engineering
Faculty of Engineering, University of Porto
Email: up201406455@fe.up.pt

Sara Serra Costa
Security for Networks and Systems
Telecommunications, Eletronics and Computers
Master in Electrical and Computer Engineering
Faculty of Engineering, University of Porto
Email: up201402938@fe.up.pt

***Abstract*—The mirai botnet, consisting primarily of IoT devices, performed one of the largest attacks of distributed denial-of-service attack. In this paper, we will talk about the mirai and the attacks in which it was involved, as well as its evolutions, since its source code was made available on the internet. Finally, we will analyze an attack performed by Mirai. Although these attacks are relatively recent, there is already much research in the area, which will continue to be updated by the rapid evolution of malware. We concluded that by its simplicity and rapid growth mirai was able to originate significant attacks compromising even the best-defended targets, so we indicated some preventive measures of protection.**

I. Introduction

This work was developed under the course of Security for Networks and Systems and had the objective of understanding how malware attacks work, more specifically the attack carried out by Mirai.

Taking advantage of IoT device vulnerabilities, Mirai has formed an ”army” of botnets carrying out one of today’s biggest DDoS attacks. In a simple but smart way, Mirai searched for open Telnet ports on the Internet, then trying to access them through default passwords, infecting thousands of IoT devices used during the attack. Since the source code has been made available on the Internet, new threats appear every day, exploiting new vulnerabilities and reaching new targets according to the author’s purpose.

In the following sections, we’ll start by explaining how Mirai came about, describing the attacks it was involved in, and describing other malware attacks that have emerged from it. Finally, we will explain how these malware can be detected, through an example of an attack performed by Mirai present in a pcap obtained in another work carried out also within the scope of this course.

II. Description

Mirai is a malware created by two university students, Paras Jha and Josiah White, that turns vulnerable devices into remotely controlled ”bots” that can be used as part of a botnet in large-scale network attacks. For this, Mirai scanned big blocks of the internet for open Telnet ports and attempted to log in default passwords, thus creating a botnet army.

A botnet is a collection of internet-connected devices that are under remote control from some outside party. The infected devices behave like zombies, i.e. they are inactive until the moment an attack is started, or they can be used to infect other devices. Usually, these devices have been compromised by some outside attacker who controls aspects of their functionality without the owners knowing. With the existence of several bots in different parts of the internet, all under the control of the hacker, it becomes difficult to locate the supercomputer that generates the attack.

Botnets can be used to perform Distributed Denial of Service attack (DDoS attack), which is the Mirai botnet served. The owner can control the botnet using command and control (CNC) software. The attacker can use his botnet army for spambots too. This kind of attack was the first to appear and is still used because the unwanted messages are being sent from many different computers, making them hard to spam filters to block.

A DDoS attack works by drowning a system with requests for data. This could be sending a web server so many requests to serve that page it crashes under the demand, or it could be the database being hit with a high volume of queries.

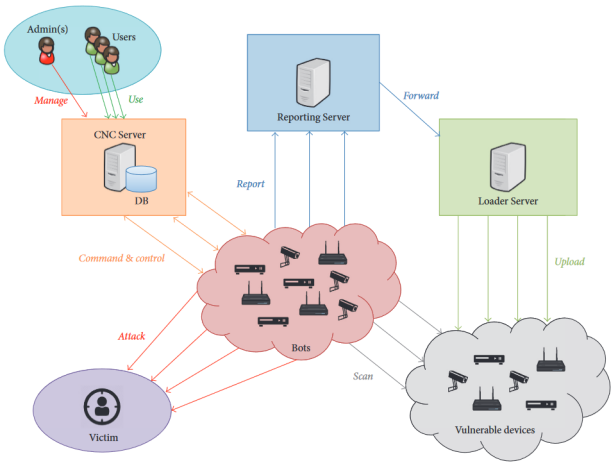


Fig. 1. Mirai Attack

III. Mirai Attacks

Jha Paras becomes interested in how DDoS attacks could be used for profit. He launched a series of minor attacks against his own university's systems, timed to match important events like registration and midterms, all the while trying to convince them to hire him to mitigate those attacks.

He was also a big Minecraft player, and one of the peculiarity of the game is to obtain profit hosting Minecraft game servers. This can lead to disputes between hosts since they launch DDoS attacks against their rivals in order to put the servers offline to attract as many players as they can.

The first Mirai attack occurred on September 19, 2016 and was used against the French host OVH because OVH hosted the popular tool that Minecraft server hosts use to fight against DDoS attacks.

The biggest attack took place on October 12, 2016 and was launched by somebody else against Dyn, an infrastructure company that, among other things, offers DNS services to a lot of big websites. This DDoS attack occurred using Mirai malware installed on a large number of IoT devices, resulting in the inaccessibility of several high-profile websites such as GitHub, Twitter, Reddit, Netflix, Airbnb, Amazon, Spotify and many others.

This happened because Jha Paras posted the code of the Mirai botnet online with the purpose of making it harder to figured out who was the creator, since many hackers have access to the code. However, in December 2016, Jha and his associates pled guilty to crimes related to the Mirai attacks.

As the code was in the wild, other hackers had the opportunity to improve it to be used in new attacks and be more difficult to fight.

IV. Evolution of Mirai

Since the release of the Mirai source code, botnet authors have been using it as a framework for new malware. They add in new exploits and functionalities that decreases drastically the development time for botnets. These botnets are being used not only for DDoS attacks, but for other purposes, such as attacks on cryptocurrency mining clients.

A. Satori Botnet

Is a mirai variant whose main target is software associated with ARC processors, that are used by many IoT devices.

Satori scans vulnerable IoT devices, and if the devices default settings have not been changed, it uses the default credentials to control the device. Once compromised, it searches the network for other vulnerables devices and attempts to repeat the process.

Researchers were able to identify an attack using devices controlled by Satori. In this attack, the devices would scan the Internet for cryptocurrency-mining devices that were exposed. When under control, the attackers would reconfigure the devices to join a mining pool and deposit earned cryptocurrency into their wallet.

B. Masuta

Has two variants, Masuta Botnet that uses the sames techniques as Mirai by attempting to overcome the security of the IoT device using a built-in list of default credentials. And PureMasuta, described as an enhanced version with a built-in exploit against certain vulnerable routers.

Unlike Satori, with Masuta, attackers have the possibility to take direct control of the devices and make other malicious actions, including:

- Trojan Activity - uses embedded code to launch a Trojan virus onto the target IoT devices, this allows the attacker to view the devices activity in real time and take control of the machines at any given time.
- Traffic Redirect Activity - using compromised switches or routers, the attacker can reroute internet traffic through a controlled server enabling complex man-in-the-middle attacks.
- Malware Delivery - utilizes the compromised devices to deliver various types of viruses to other connected devices on the network.

C. Wicked Mirai

Scans multiple ports on network devices, and uses open ports to download copies of various payloads, the type of payload depends on which port is available. Differently from Mirai, which uses brute force to access vulnerable IoT devices, the Wicked Mirai depends on a variety of port related vulnerability exploits to gain access to a device.

Once a device is compromised, Wicked Mirai contacts CNC server from which downloads a payload that meets the attackers objective and nature of the device.

This Mirai variant looks for specific vulnerabilities on a platform that the botnet can exploit. This way, the botnet controllers have a faster compromise time allowing the botnet to come online faster. Additionally, the Wicked Mirai can maintain a presence on infected IoT devices, increasing its power and persistence to launch substantial DDoS attacks.

D. JenX Botnet

Uses the Grand Theft Auto videogame community to infect IoT devices. Unlike other variants, JenX has a centralized infrastructure and uses a central server to scan for new hosts.

The other variants perform distributed scanning and exploiting, each compromised device searches for new victims, which grants a rapid growth but comes at the price of flexibility. JenX, with its centralized structure, provides attackers with great flexibility to improve the functionality of the malware over time.

V. Mirai Detection

In this part of the paper we will describe the analysis that we made to the pcaps of an attack performed by Mirai Malware.

In the figure 2, the constituents of the analyzed attack are represented. This attack was performed by another group in the lab bench, so we will refer to the machines by the name Tuxn, where n is the number of the machine in question.

Describing the constituents, we verified that the attacker was in Tux1, with the IP address 172.16.1.61, the CNC server was in Tux4, with the IP address 172.16.1.64, the Bot was in Tux3, with the IP address 172.16.1.63, and finally, the victim was on Tux2, with IP address 172.16.1.62.

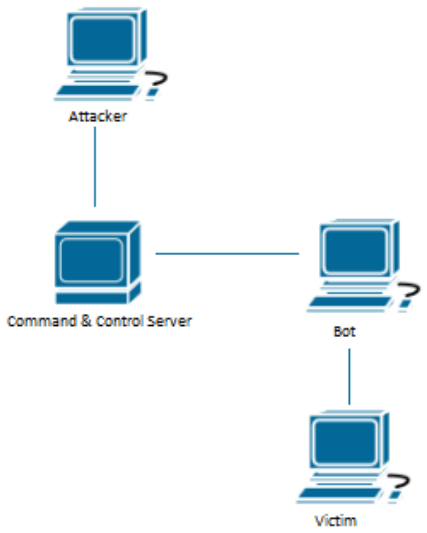


Fig. 2. Constituents of the analyzed Attack

Let's start by analyzing the attacker's login on the CNC server. Both parties communicate through TCP and Telnet protocols.

```
..... ..!.."..'.....#.[?1049h........".. ..... ..... ..... .....
.[34;1m.....[33;3m: .[0m...tteesstt
.
.[34;1m.....[33;3m: .[0mt*e*s*t*
.

.[37;1m..... ..... .[31m-
.[37;1m..... ..... .[31m\
.[37;1m..... ..... .[31m|
.[37;1m..... ..... .[31m/
.[37;1m..... ..... .[31m-
.[37;1m..... ..... .[31m\
.[37;1m..... ..... .[31m|
.[37;1m..... ..... .[31m/
.[37;1m..... ..... .[31m-
.[37;1m..... ..... .[31m\
.[37;1m..... ..... .[31m|
.[37;1m..... ..... .[31m/
.[37;1m..... ..... .[31m-
.[37;1m..... ..... .[31m\
.[37;1m..... ..... .[31m|
.[0m[+] DDOS | Succesfully hijacked connection
[+] DDOS | Masking connection from utmp+wtmp...
[+] DDOS | Hiding from netstat...
[+] DDOS | Removing all traces of LD_PRELOAD...
[+] DDOS | Wiping env libc.poison.so.1
[+] DDOS | Wiping env libc.poison.so.2
[+] DDOS | Wiping env libc.poison.so.3
```

Fig. 3. TCP stream between CNC Server and Attacker

The first data packet sent by the server determines command line text output color,. [? 1049h. The CNC server prompts the user to enter a username, followed by a password, through the telnet protocol.

```
[?1049h [34;1mпользователь [33;3m:  [0m tteesstt

[34;1mпароль [33;3m:  [0mt*e*s*t*
```

Fig. 4. Username and Password

In *red* we have the attacker and in *blue* the CNC server. The server sends the following messages: "I love chicken nuggets", followed by "user", and finally "password". We have managed to verify that the username and password are test and test.

Now let's look at the remaining data in blue. The server repeatedly sends the message "having checked accounts", in order to create a "dynamic" rotation at the end.

In the following messages sent, [+] DDOS and Wiping env libc.poison.so, the CNC Server pretends to perform some masking operations.

```
[37;1mпроверив счета... [31m-
[37;1mпроверив счета... [31m\
[37;1mпроверив счета... [31m|
[37;1mпроверив счета... [31m/
[37;1mпроверив счета... [31m-
[37;1mпроверив счета... [31m\
[37;1mпроверив счета... [31m|
[37;1mпроверив счета... [31m/
[37;1mпроверив счета... [31m-
[37;1mпроверив счета... [31m\
[37;1mпроверив счета... [31m|
[37;1mпроверив счета... [31m/
[37;1mпроверив счета... [31m-
[37;1mпроверив счета... [31m\
[37;1mпроверив счета... [31m|
[0m[+] DDOS | Succesfully hijacked connection
[+] DDOS | Masking connection from utmp+wtmp...
[+] DDOS | Hiding from netstat...
[+] DDOS | Removing all traces of LD_PRELOAD...
[+] DDOS | Wiping env libc.poison.so.1
[+] DDOS | Wiping env libc.poison.so.2
[+] DDOS | Wiping env libc.poison.so.3
[+] DDOS | Wiping env libc.poison.so.4
[+] DDOS | Setting up virtual terminal...
[37;1m[!] Sharing access IS prohibited!
[!] Do NOT share your credentials!
```

Fig. 5. Messages sent by CNC Server

After the connection is established, the server sends the number of connected bots, one in this case, and the attacker's username, every second.

```
[36;1mReady
[32;1mtest@botnet#  [0m ]0;1 Bots Connected | test  ]0;1 Bots Connected | test  ]0;1 Bots Connected | test
```

Fig. 6. Bots connected

Client sent the command help but it return an error since it is not a valid attack. To consult available attack list, the attacker must send a ?.

```
[32;1mtest@botnet#  [0m ]0;1 Bots Connected | test hhelll ]0;1 Bots Connected | test pp

[31;1m [33;1mhelp  [31mis not a valid attack! [0m
[32;1mtest@botnet#  [0m ]0;1 Bots Connected | test  ]0;1 Bots Connected | test  ]0;1 Bots Connected | test  ]0;1 Bots Connected | test ??

[31;1m [37;1mAvailable attack list
[36;1mgreip: GRE IP flood
udpplain: UDP flood with less options. optimized for higher PPS
greeth: GRE Ethernet flood
http: HTTP flood
udp: UDP flood
vse: Valve source engine specific flood
dns: DNS resolver flood using the targets domain, input IP is ignored
syn: SYN flood
ack: ACK flood
stomp: TCP stomp flood
```

Fig. 7. List of Attacks

Here is a brief description of the available attacks.

GRE, Generic Routing Encapsulation, is a tunneling protocol that can encapsulate various network layer protocols inside virtual point-to-point links over an IP network. The encapsulated IP packet header uses the same parameters as the encapsulating IP header. The Transport Layer protocol for the encapsulated IP packet is UDP. Mirai has two types of GRE

attacks, with and without Ethernet encapsulating, IP flood and Ethernet flood. To impact the target, huge payloads are sent thus adding processing overhead of IP defragmentation.

The HTTP flood causes the server to allocate as many resources as possible to the attack through HTTP requests that can be ”GET” or ”POST”. This denies legitimate users access to the server.

The UDP flood sends a large quantity of UDP packages to a remote host. As a result, the target system will send many ICMP Destination Unreachable packets, becoming eventually unreachable to other clients.

The Valve Source Engine flood (VSE) is a UDP attack used to consume available resources of a server. It sends many TSource Engine Query requests, preventing the server to process all of them and creating a denial of the gaming service.

The DNS flood attack attempts to exhaust server assets, memory or CPU, by overwhelming server resources with a flood of UDP requests. This causes the server to not be able to direct legitimate request to zone resources.

The SYN flood sends successive syn requests in order to block the server's available server resources, preventing the response to legitimate requests.

The ACK flood sends a significant amount of scattered ack packets, not related to any open connection. As a result, system resources are redirected to evaluate incoming packets, slowing down performance or causing a complete failure.

Stomp is a simple application layer text-based protocol. The TCP stomp flood sends many fake stomp requests, after using stomp to open an authenticated TCP handshake with the targeted application. This flood leads to network saturation, and also, if the target is programmed to parse stomp requests this exhausts server resources.

We will now describe the attack.

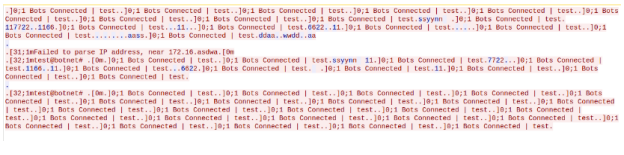


Fig. 8. TCP stream of the Attack Command

We noticed that the first attack attempt failed because of an error entering the command. A new attempt was followed, this one was successful, being the command of attack ”syn 172.16.1.62 1”. We have found that syn, from the list of previously available attacks, means that the attacker wants to trigger a syn flood attack. The IP address of the victim is 172.16.1.62, and the duration of the attack is one second.

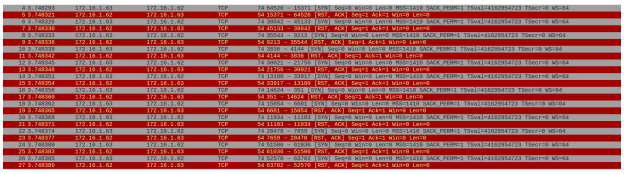


Fig. 9. Excerpt of communication between the Bot and the Victim - Attack

The first syn was received by the victim in 3.748293, and the last in 4.749721, from this we conclude that the syn flood actually happen for a second.

VI. Conclusion

With this work we acquire new knowledge in the area of computer security. We see how we can prevent malware attacks and get a small example of how they can be detected. The latter is extremely difficult because it depends entirely on the type of attack performed.

The existence of vulnerable devices on the Internet makes it possible to continue DDOS attacks in the near future. In addition, Mirai's evolution makes it difficult to eradicate malware so some precautionary measures should be taken, such as eliminating default credentials, so that attackers can not build a credential list that allows them to compromise vulnerable devices, and enforce login , either by imposing a limit on the login flow, preventing brute-force attacks, or using a captcha or proof of work. In this way, it will be possible to reduce the number of attacks carried out because less IoT devices will be vulnerable to infections by these malwares. Still, it is necessary to continue to investigate ways to detect and combat malware attacks as new threats will surely arise.

References

[1] URL https://www.csoonline.com/article/3258748/security/the-mirai-botnet-explained-how-teen-scammers-and-cctv/-cameras-almost-brought-down-the-internet.html.

[2] URL https://developer.ibm.com/articles/iot-anatomy-iot-malware-attack/#how-to-protect-your-iot-devices.

[3] URL https://www.csoonline.com/article/3222095/network-security/ddos-explained-how-denial-of-service-attacks-are-evolving.html.

[4] URL https://en.wikipedia.org/wiki/Mirai_(malware).

[5] URL https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective/-analysis/.

PowerShell Empire

Anabela Reigoto
Faculdade de Engenharia
da Universidade do Porto
4200-465 Porto, Portugal
Email: up201405662@fe.up.pt

Renato Cruz
Faculdade de Engenharia
da Universidade do Porto
4200-465 Porto, Portugal
Email: up201405867@fe.up.pt

***Abstract*—PowerShell Empire is a post-exploitation framework for computers and servers which run on Windows, which allows to perform different attacks on machines, compromising them. This report presents an analysis on the PowerShell Empire, how this enables to perform attacks and how to detect these through Suricata as Network Intrusion Detection System (NIDS). The detection was made assuming that the empire PowerShell code was not modified.**

I. Introduction

PowerShell Empire is a post-exploitation framework for computers and servers which run on Windows. Empire has the ability to run PowerShell agents without the executable, because PowerShell is much more than an executable, powershell.exe is a console application for PowerShell processes in the System.Management.Automation.dll, one of the main components of the Windows operating system, and can not be removed because it is naturally one of the main components. Once established the connection, PowerShell Empire allows to perform different attacks on this machine using different modules, like privilege escalation, and built-in tools, such as download and upload of files.

PowerShell Empire relies on three principal components:

Listeners - Running process that listen to connections from the victim's computer, after which, it will serve as a C&C (Command and Controll).

Stagers - Payload that allows the malicious code to run through on the compromised machine to establish the connection to the Listener.

Agents - Compromised machine with connection established to an Empire C&C.

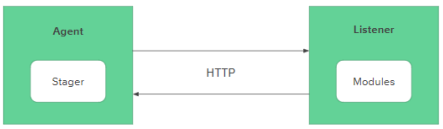


Fig. 1. PowerShell Empire Listener Agent connection

The malicious connection allows the attacker to snoop around the victim's machine and obtain important information about it, such as domain administrator credentials, and can be detected using an **Network Intrusion Detection System (NIDS)** and prevented using **Network Intrusion Prevention System (NIPS)**.

NIDS is a software that monitors network traffic for malicious activity and policy violations, and can work based on two paradigms:

Signature-based - Looks for known attack patterns

Anomaly-based - Looks for deviations on acceptable traffic models

Suricata is an open-source NIDS, that works as both signature-based or anomaly based, because it inspects the traffic of a network and classifies it as a threat through rules that can be tuned to work both ways. Although it also has mechanisms of Network Intrusion Prevention System (NIPS) and Network Security Monitoring (NSM), in this work, it will only serve as a NIDS.

Unlike conventional post-exploitation tools, that open a reverse shell between the victim and the

attacker, which leave traces of it's work, by opening doors and having constant flows between both machines, PowerShell Empire works by having the Agent periodically send keep-alive requests to the attacker, in the form of HTTP GET requests, where the C&C responds with HTTP 200 OK, this responses contain the commands to execute. This is the basis of PowerShell Empire, and it's what makes it hard to detect, there is no constant open connection, it's just normal traffic.

II. Related work

The approach followed in this paper to detect PowerShell Empire consists in using Suricata to search for known patterns, such as fixed headers or sequence of messages that might signal a certain action in progress. With Shodan, it has been shown that this approach effectively narrows down common traffic to Empire traffic by searching for common HTTP headers used by it [2]. With this approach, Suricata will only be used as a signature-based NIDS. To use it as anamoly-based, the study scenario would have to specify what is normal and acceptable behaviour. Although that can be easily done in a corporate environment, where only certain server are visited, and anything that deviates from them will be suspicious, this will not be exploited in this work.

By knowing the process Empire takes to establish a connection, this can exploited with Suricata, to look for certain sequence of messages [1] As seen in Figure 2, there is a fixed sequence of messages that can be exploited, but detecting Empire traffic it's not so trivial because the connection is fully encrypted with a pre-shared staging key. This can be further problematic if the Listener is configured to use SSL certificates, so this paper will only focus on plaintext Empire traffic.

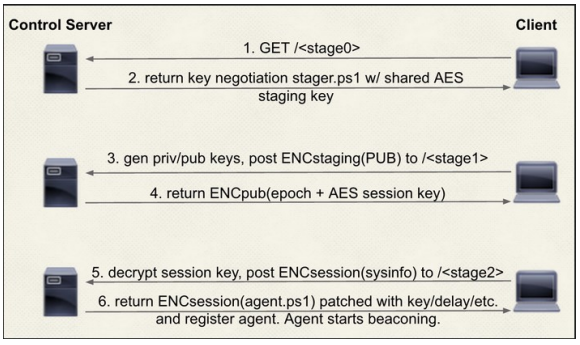


Fig. 2. Empire staging process[1]

III. Solution

A. Study Scenario

This subsection explains the scenario used to test and analyse PowerShell Empire's behaviour. To this end, all antivirus, such as Windows Defender, were disable on the victim's machine as the point of this work was a post exploitation tool, it wasn't really important how the stager gets executed, only that it does. As seen in Figure 3, the topology

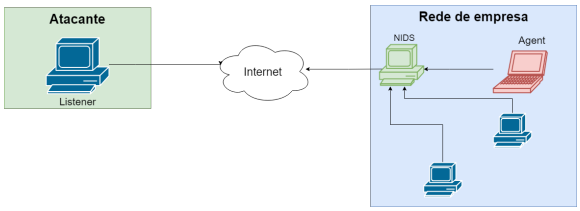


Fig. 3. Topology used

simulated a corporate environment, were a victim is inside a private network and all it's traffic flows through a known gateway with NIDS capabilities. The attacker is in an outside network working as a server waiting for connections.

B. Traffic Analysis

As explained in section II, this work will focus on known packet patterns, by searching for fixed headers.

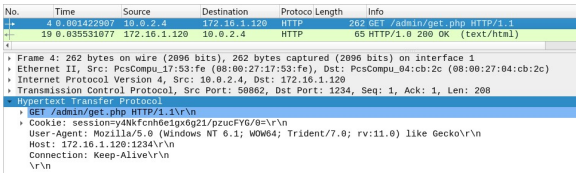


Fig. 4. GET requests - Wireshark

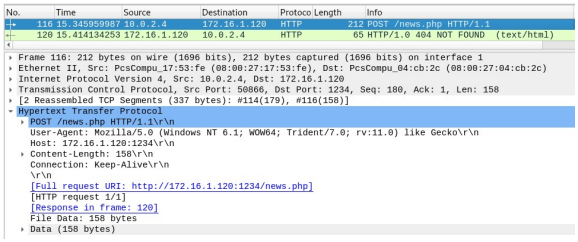


Fig. 5. POST requests - Wireshark

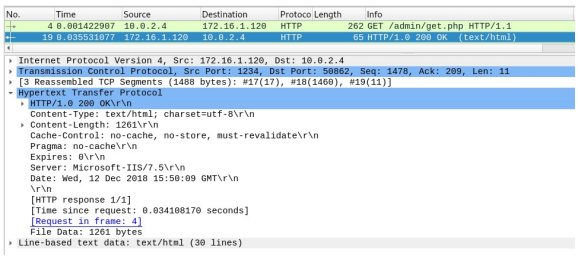


Fig. 6. 200 OK responses - Wireshark

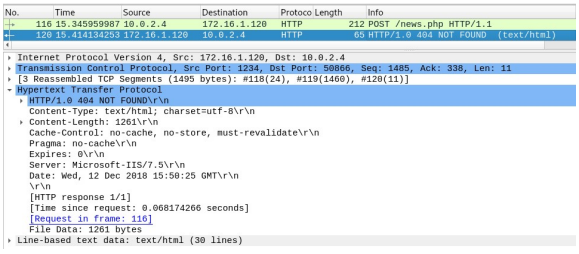


Fig. 7. 404 NOT FOUND responses - Wireshark

From figures 4, 5, 6, 7 the fixed headers that will be exploited can be seen and are laid out in the following table:

TABLE I
COMMON EMPIRE HEADERS
X NOT USED; - USED BUT VARIABLE

Headers	GET	POST	200 OK	404 NOT FOUND
Connection	Keep-Alive	Keep-Alive	X	X
Host	-	-	X	X
User-Agent	-	-	X	X
Cookie	session=	X	X	X
Content-Type	X	X	tetx/html; charset=utf-8	tetx/html; charset=utf-8
Expires	X	X	0	0
Pragma	X	X	no-cache	no-cache
Cache-Control	X	X	no-cache, no-store, must-revalidate	no-cache, no-store, must-revalidate
Date	X	X	-	-
Server	X	X	-	-

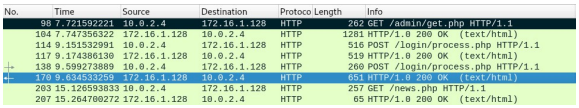


Fig. 8. Empire staging process - Wireshark

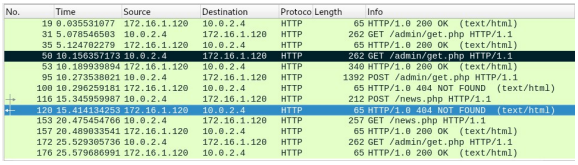


Fig. 9. Empire download process

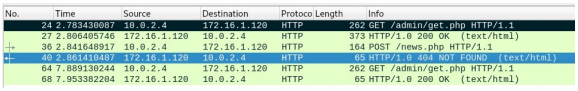


Fig. 10. Empire upload process

From figures 8, 9, 10, the sequences of messages exchanged by Empire to execute the respective actions are shown. They will be later exploited to make Suricata rules. These order of messages aligned with the previous headers, makes Empire predictable in a way, thus making it detectable.

Figure 8 corroborates the sequence of messages found in Figure 2. When there is no more commands to be executed, a default web page is re-

turned. Empire uses python's Flask framework, that uses Werkzeug, in which the 404 NOT FOUD webpage is hardcoded, so following the assumption that it was modified, it's size is fixed at 1261 bytes

C. Suricata Rules

Suricata is a rule-based NIDS, which analyses traffic and takes action upon it, based on a set of pre-established rules. Each of this rule is composed of three main fields[3]:

Action - determines what happens when the signature matches: *pass*, *drop*, *reject*, *alert*

Header - defines the protocol, IP addresses, ports and direction of the rule:
[protocol] [src network] [port] -> [dst network] [port]

Options - defines the specifics of the rule. With these options it's possible to establish warning messages on rules'match, as well as tune the rule to filter specific contents.

With this format established, it's important to refer two crucial options that will be used throughout this section, with which PowerShell Empire protocol messages were detected:

```
threshold:type <threshold|limit|
    both>,track <by_src|by_dst>,
    count <N>,seconds <T>
xbits:<set|unset|isset|toggle>,<
    name>,track <ip_src|ip_dst|
    ip_pair>
```

The *threshold* option allowed to detect periodicity in packets. A rule is matched if it occurred <N> times in <T> seconds, tracking the rule by source or destination address.

The *xbits* option allowed to detect specific sequences of messages. A rule is matched if all other filters apply and if the bit <name> was previously *set*, or if all options in the rule are matched, the <name> bit can be set for future use. In this work, tracking by both source and destination address (ip_pair) will be used to further improve results and detect both the victim and the C&C.

All of the rules presented were made to detect HTTP packets, where *source network* is the *$HOME_NET* from *any* port, with *destination network* is *any* to *any* port:

```
alert http $HOME_NET any -> any any
```

As explained in subsection III-B, all GET, POST, 200 OK, 404 NOT FOUND are detectable because they have common patterns that allows rules to detect them.

1) Detecting GET: Following subsection III-B, fixed headers are first filtered, to help narrow down Empire packets:

```
flow: established,to\_server
content:"GET";http_method
content:"Connection:Keep-Alive"
content:"Host"
content:"User-Agent"
content:"session=";http_cookie
```

Furthermore, these requests are used in the keep-alive messages, using the default periodicity and the *threshold*, *threshold: type threshold, track by_src,count 5, seconds 26*, these keep-alives were successfully detected every 26 seconds.

2) Detectin POST: Headers:

```
flow: established,to_server
content: "POST"; http_method
content: "Connection: Keep-Alive"
content: "Host"
content: "User-Agent"
```

This is a very broad filter, and, by itself, detected many false positives, so this needs to be used together with other options and rules to detect specific actions.

3) Detecting 200 OK: Headers:

```
http_response_line;content:"HTTP
    /1.0 200 OK"
flow: established,to_client;
content:"Server"
content:"Date"
content:"Cache-Control: no-cache,
    no-store, must-revalidate"
content:"Pragma: no-cache"
content:"Expires:0"
http_content_type;content:"text/
    html|3B| charset=utf-8"
```

200 OK without commands:

```
http_content_len;byte_test:0,=,1262,0,
    string,dec
```

200 OK with commands (no html tags found):

```
content:!"<html";content:!"<head>";
    content:!"<body>"
```

4) Detecting 404 NOT FOUND: Headers:

```
flow:established,to_client
content:"Cache-Control: no-cache,
    no-store, must-revalidate"
content:"Pragma: no-cache"
content:"Expires: 0"
http_response_line;content:"HTTP
    /1.0 404 NOT FOUND"
```

As previously stated, Empire uses python's Flask framework with Werkzeug, which in turn uses a hardcoded default 404 NOT FOUND page.

```
http_content_len;byte_test:0,=,1261,0,
    string,dec
```

5) Detecting Empire actions: This section describes how the previous base rules can be used in chains to detect the sequences of messages Empire exchanges to perform certain actions. These patterns were observed using wireshark and are explained in subsection III-B, as such, all rules are based on those assumptions.

Setup: Detecting this part of Empire is important because it allows the defending entities to block the attacker's IP, with a NIPS, before any harm is done to the victim.

From observing the wireshark logs in Figure 2, the time between both POST requests was short in all repetitions observed. With this they could be filtered using *threshold*:

```
threshold: type threshold, track
    by_src,count 2,seconds 1.5
```

This option was added to the base rules that detect POST requests and 200 OK responses with commands, as explained in subsubsection III-C2 and subsubsection III-C3.

Basic commands: GET + 200 OK with commands

Using the rules from subsubsection III-C1 and subsubsection III-C3 combined with *xbits*, by adding

```
xbits:set,command1,track ip_pair <-
    GET
xbits:isset,command1,track ip_pair <-
    200 OK
```

Upload: Command + POST (size:110) + 404 NOT FOUND

POST request with fixed length (110) content, was assumed to be an acknowledgement to the upload

```
http_content_len;byte_test:0,=,110,0,
    string,dec
```

Rule chain to detect upload sequence:

```
xbits:set,upload1,track ip_pair <- 200
     OK
xbits:isset,upload1,track ip_pair;
    xbits:set,upload2,track ip_pair <-
    POST
xbits:isset,upload2,track ip_pair <-
    404 NOT FOUND
```

Download: Using *xbits* Suricata can detect the download sequence as explained in subsection III-B:

```
xbits:set,download1,track ip_pair <-
    200 OK with command
xbits:isset,download1,track ip_pair;
    xbits:set,download2,track ip_pair
    <- POST
xbits:isset,download2,track ip_pair;
    xbits:set,download3,track ip_pair
    <- 404 NOT FOUND
xbits:isset,download3,track ip_pair;
    xbits:set,download4,track ip_pair
    <- POST
xbits:isset,download4,track ip_pair <-
     404 NOT FOUND
```

IV. Evaluation

The tests were executed in a network with the topology represented in the figure 3. The evaluation on the listener-agent connection detection was performed by configuring a HTTP Listener with the Host IP of the attacker's machine in port 1234. After a windowns/laucher_bat stager is generated using the previous HTTP Listener activated.

Fig. 11. PowerShell Empire HTTP Listener.

Fig. 12. PowerShell Empire Stager configuration.

In order to activate the agent, in the victim's machine the stager downloaded, accessing in the browser the attacker IP host followed by the stager file name. As soon as the stager is run, the connection is activated and detected in the compromised machine.

The performed tests to detect upload and download consisted in performing both using files of different sizes and types. The files sizes varied between 400 Bytes to 4 KBytes, being these a *.jpg*, a *.pdf* or *.txt* file. After the operations, it was always verified if the file transferred was totally equal to the original. The victim's machine was always successful in detecting these interactions, right after they were completed.

Another remain possibles intrusions were preformed, as a screenshot of the attacked machine or the set of the wallpaper. These attacks are also detected by the Suricata rules, but as general ones, not being able to specify it.

As expected the developed Suricata rules were able to detect when the malicious connection is set up and when an upload, download or and undifferentiated intrusion is performed, as shown in figure 13 presented in VI .

V. Conclusion

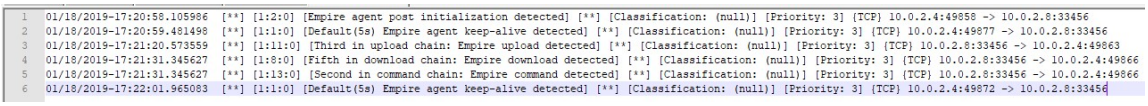
It was developed several Suricata rules to detect PowerShell Empire attacks on a Windows machine. It was possible to develop this rules analysing the interaction between the two machines through Wireshark, noticing the common aspects in the messages exchanged, as the type of messages, their header and the sequence in which they were exchanged.

In this report are presented Suricata rules that allows the NIDS to identify the establishment of a PowerShell Empire connection, an upload, a download or any other attack. It was not possible to detect other specific attacks as a screenshot, due the PowerShell encryption of the messages, that despite the attempts was not possible to decrypt.

References

[1] Offensive and Defensive PowerShell, https://artofpwn.com/offensive-and-defensive-powershell-ii.html.
[2] Identifying Empire Listeners, https://www.tenable.com/blog/identifying-empire-http-listeners
[3] Suricata, Rules format, https://suricata.readthedocs.io/en/suricata-4.1.2/rules/intro.html

VI. ANNEXES



```
01/18/2019-17:20:58.105986  [**] [1:2:0] [Empire agent post initialization detected] [**] [Classification: (null)] [Priority: 3] {TCP} 10.0.2.4:49858 -> 10.0.2.8:33456
01/18/2019-17:20:59.481498  [**] [1:1:0] [Default(5s) Empire agent keep-alive detected] [**] [Classification: (null)] [Priority: 3] {TCP} 10.0.2.4:49877 -> 10.0.2.8:33456
01/18/2019-17:21:20.573559  [**] [1:11:0] [Third in upload chain: Empire upload detected] [**] [Classification: (null)] [Priority: 3] {TCP} 10.0.2.8:33456 -> 10.0.2.4:49863
01/18/2019-17:21:31.345627  [**] [1:8:0] [Fifth in download chain: Empire download detected] [**] [Classification: (null)] [Priority: 3] {TCP} 10.0.2.8:33456 -> 10.0.2.4:49866
01/18/2019-17:21:31.345627  [**] [1:13:0] [Second in command chain: Empire command detected] [**] [Classification: (null)] [Priority: 3] {TCP} 10.0.2.8:33456 -> 10.0.2.4:49866
01/18/2019-17:22:01.965083  [**] [1:1:0] [Default(5s) Empire agent keep-alive detected] [**] [Classification: (null)] [Priority: 3] {TCP} 10.0.2.4:49872 -> 10.0.2.8:33456
```

Fig. 13. Attacks detection through Suricata.

Impact of WPScan as a Vulnerability scanning tool

Miguel Ribeiro
Faculdade de Engenharia da
Universidade do Porto
Master in Electrical
and Computers Engineering
Email: up201405769@fe.up.pt

Nuno Adrego
Faculdade de Engenharia da
Universidade do Porto
Master in Electrical
and Computers Engineering
Email: up201403429@fe.up.pt

***Abstract*—Vulnerability Scanner is the term used for software that is designed to find known weaknesses in computers, networks or applications. These weaknesses are commonly miscondigurations or out-of-date software.**

In the context of the course unit "Security for Systems and Networks", the team was challenged to explore one Vulnerability Scanning tool, WPScan, and observe its impact at targets.

I. Introduction

Wordpress is the most used CMS - Content Management System - on the Internet, running about 30% of all website.

Famous breaches of WordPress website, such as Panama Papers Leak, are mainly due to outdated themes or plugins that open paths for security enthusiasts.

On this article we explore the tool WPScan, a free (for non-commercial use) black box WordPress vulnerability scanner. This tool uses a database which contains 2985 unique vulnerabilities

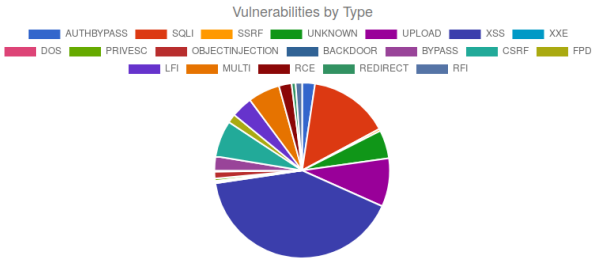


Fig. 1. Types of vulnerabilities present in the database

II. Approach

First, we needed websites running WordPress, in order to WPScan tool have targets. Even though the tests we were planning at this stage were for academic use only, they could also be interpreted as a transgression of some good practices in the field of security. So, in order to tackle this constraint, we decided to look for websites hosted and controlled by someone belonging to our college. The following query was used on Google:

```
site:fe.up.pt inurl:wordpress
```

This is possible because many people unzip WordPress folder and do not move the files to the root of the web server.

So, after running the query on Google, we extracted the links from de *serp* with the Chrome extension Linkclump.

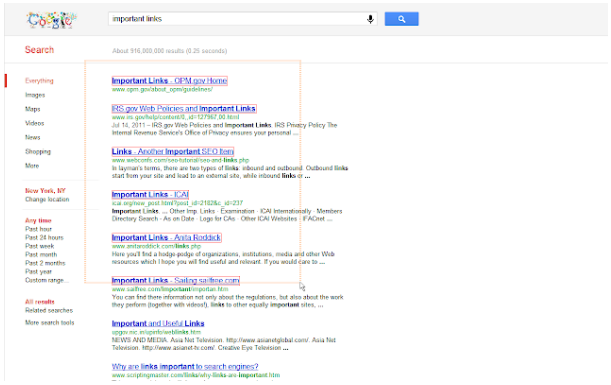


Fig. 2. Linkclump use on a webpage

With this approach, we were allowed to find eighteen unique websites listed bellow. There are certainly more WordPress websites hosted in fe.up.pt servers, but its not trivial to find them because, for both security and SEO reasons, WordPress opts for not indexing pages such as */wp-login.php*.

Once the websites to test were decided we wanted to measure the time a GET request took for each website under different conditions:

1) Normal (100) - without the use of WPScan, one hundred requests;
2) With WPScan (100) - with the use of WPScan, one hundred requests;
3) With WPScan (50/50) - without the use of WPScan, fifty requests and with the use of WPScan, fifty requests;
4) With 4x WPScan (50/50) - without the use of WPScan, fifty requests and with the use of 4 simultaneous WPScans, fifty requests.

III. Solution

To do so, for the 1) condition, we made a python script using the *requests* library which enables a GET request function and then its *elapsed_time* method which returns the time, in seconds, for a given single request.

In order to obtain a more accurate time value for each website we chose to make one hundred requests in a row

and appended the average value to an output file adding a preceding label with the corresponding date/time of request and website url.

Since we needed to measure quite a few time values during the day as to have observation material we created a *cron_tab* job on a virtual machine in a FEUP network labs computer running the python script through all the websites resulting in a output file with the average time values.

Once the output file had enough data we ran it with a parser written by us in C language to create a .csv file. The file would then be imported to excel where we built graphics and observed the results.

For the following 2), 3) and 4) conditions we added threads to run the WPScan in the python script while the requests were being made: one thread for conditions 2) and 3) and four threads for condition 4).

Bellow, in Figure 3, we can observe our first raw results.

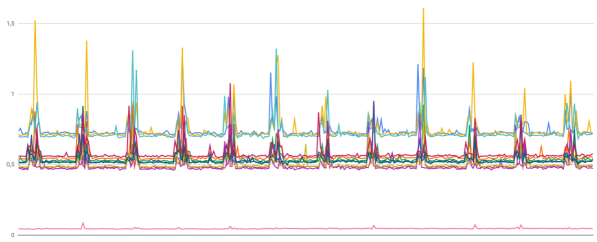


Fig. 3. Results of condition 1) Normal (100)

The peaks we can see in Figure 3 correspond to the night period. With such peaks and variations during the night, we decided to remove these values during the period from 23h00 to 07h59 and study the values from the remaining period 8h00 to 22h59 resulting in what we can see in Figure 4.

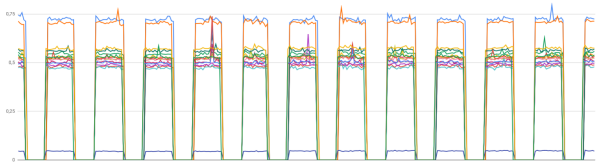


Fig. 4. Results of condition 1) Normal (100) with filter

To synchronize the various values we had to reduce the interval window of the results. For the 1) and 2) conditions we obtain the Figure 5.

IV. Evaluation

As we can notice in Figure 6, the expected increase in time is barely noticeable for the 2) condition.

The difference between 2) and 1) results in an average positive value. Which can show the average delay from WPScan: 0,0074 seconds. However, as we can see its value is to small to have a meaningful impact on the websites load time.

To further check the results from 1) and 2) comparison we observed the 3) condition and the difference between each fifty requests.

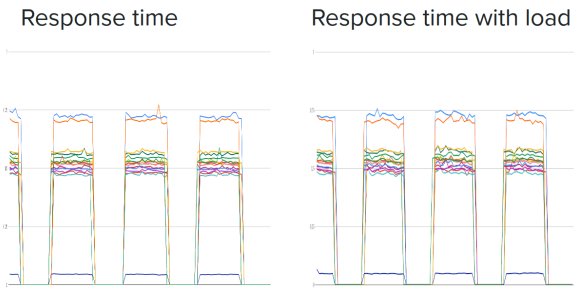


Fig. 5. Condition 1) and 2)

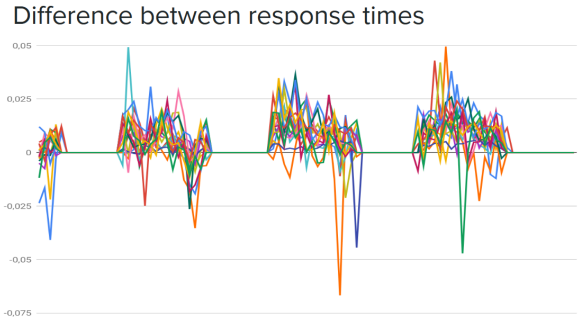


Fig. 6. Results of the difference between condition 1) and 2)

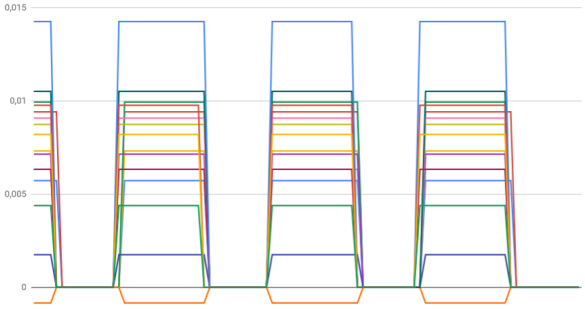


Fig. 7. Results of the difference between condition 1) and 2) on average

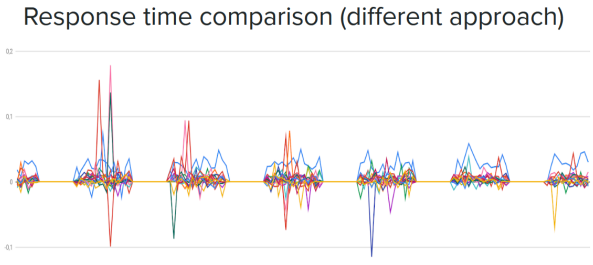


Fig. 8. Results of the difference between each 50 requests from condition 3)

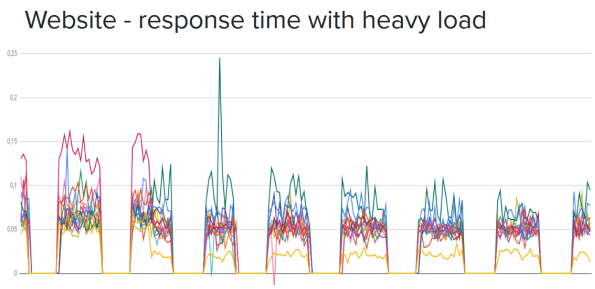


Fig. 9. Results of the difference between each 50 requests from condition 4)

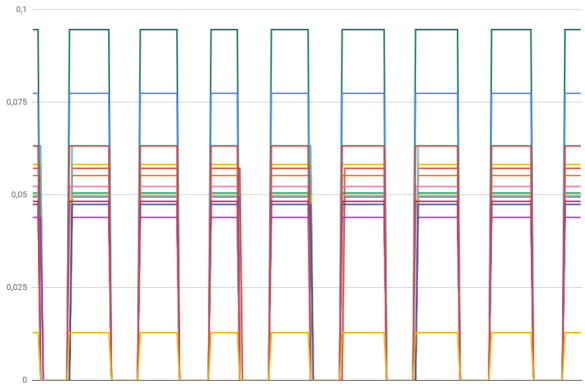


Fig. 10. Results of the difference between each 50 requests from condition 4) average

We have a similar conclusion from the above with an average of 0,0046 seconds.

Lastly we made the 4) condition, a heavy load test.

Now it is clearly visible the impact of WPScan with values around 0,05 seconds and above.

The average for the heavy load is 0,0568 seconds.

V. Conclusion

Doing vulnerability scans regularly and have the results flagged for website administrators is an important must. Nevertheless, the impact on normal users accessing the website can barely be perceived.

A reasonable approach is to run the scans when the are a low percentage of visitors, such as nighttime. While this might work well for small or medium websites, for bigger ones it could mean delays of 0,04s (for a single WordPress installation) for thousands of visitors.

Heres where the balance of QoE vs Security gets highlighted.

IDS & SIEM Architecture Review and Performance Assessment

André Oliveira
MIEEC, FEUP
up201405639@fe.up.pt

Baltasar Aroso
MIEEC, FEUP
up201404125@fe.up.pt

***Abstract*—Nowadays, security in networks is being sought after as cyber attacks become more sophisticated. As such, it is important to be able to identify and take corrective measures to prevent damages to digital property. In this context, this project focuses on exploring possibilities in securing a network through the deployment of an IDS combined with a SIEM. Using a distributed approach composed of a Heavy node capturing traffic and a Master Server analysing it, it is concluded that a network admin can obtain event information with delay in the order of few seconds. It is also observed that the most relevant attempts to breach the network are easily detected and displayed on intuitive graphic platforms, thus allowing a quick assessment of the state of the network.**

I. Introduction

This project aims to study different approaches to get the most important and critical information about cyber security in a specific network environment or host. This sort of goal is seeing an increase in its demand, since information is just a click away and huge amounts of malicious content in the Internet are around the corner, so the risk of being under a cyber attack is at an all time high in a world where computer is an ubiquitous tool.

With the evolution of cyber crime, even small companies and institutions must deploy security measures in their networks. Big companies inject huge amounts of capital to reduce the probability of their data being compromised. So, the importance of this project is reflected on the market as a very important area of study.

In order to detect and prevent some of these well known attacks, there are different IDS (Intrusion Detection System) and IPS (Intrusion Prevention System). Both of this software are usually packed into another one, SIEM (Security Information and Event Management), that will aggregate the relevant data from multiple sources (like different IDS/IPS) and display them in a more detailed and analytic way.

II. Related work

A. IDS and IPS

An **Intrusion Detection System (IDS)** passively monitor and notify vulnerability exploits against a target application or computer. This security software is located out-of-band of the network infrastructure being a listen-only tool that cannot automatically take action to prevent a detected attack or exploit from being able to take over the system.

An **Intrusion Prevention System (IPS)** extends IDS solutions by adding the ability to block threats in addition to detecting them. Contrariwise to the IDS, the active role of an IPS make it be inline with the network infrastructure, namely between the firewall and the rest of it's network. This security software aims to actively catches attackers that firewall or anti-virus may miss.

There are two detection mechanisms [1]:

- **Anomaly-based**: monitor network traffic and compare it against an established baseline, to determine what is considered normal for the network with respect to bandwidth, protocols, ports and other devices. This type of IDS alerts administrators to potentially malicious activity.
- **Signature-based**: monitor all the packets traversing the network and compares them against a database of signatures or attributes of known malicious threats, much like anti-virus software.

There are different types of Intrusion Detection Systems, such as [1]:

- **Network IDS (NIDS)** is deployed at a strategic point or points within the network, where it can monitor inbound and outbound traffic to and from all the devices on the network.
- **Host IDS (HIDS)** runs on all computers or devices and may be able to detect anomalous network packets that originate from inside the local network or malicious traffic that a NIDS has failed to detect.

B. SIEM

Security Information and Event Management (SIEM) consists in a software product that, as the name itself indicates, include Security Information Management (SIM) and Security Event Management (SEM). This system is used to aggregate relevant data from multiple sources, identify deviations from the norm and take appropriate action.

At its core, a SIEM can be based on rules or employ a statistical correlation engine to establish relationships amongst various event log entries.

C. Relevant Software

Table I presents some of the most well known IDSs in the market. Conversely, there are a lot of SIEMs that already implement IDSs and configure them automatically. LogPoint, SIEMonster, AlienVault OSSIM, Splunk and ELK are some

examples of SIEMs researched for this project. However, some of them could not be used in this project due to limitations of their free/lite version.

TABLE I
DIFFERENT IDSS CLASSIFIED PER TYPE AND OPERATING SYSTEMS WHERE THEY CAN BE IMPLEMENTED

IDS/IPS	HIDS/NIDS	Linux	Windows	MacOS
Snort	NIDS	Yes	Yes	No
OSSEC	HIDS	Yes	Yes	Yes
Suricata	NIDS	Yes	Yes	Yes
Bro	NIDS	Yes	No	Yes
AIDE	HIDS	Yes	No	Yes
Open WIPS-WG	NIDS	Yes	No	No
Samhain	HIDS	Yes	No	Yes
Fail2Ban	HIDS	Yes	No	Yes

III. SOLUTION

A. Security Onion

Having analysed various alternatives, Security Onion was elected the best candidate for understanding the power and capabilities of a security prone environment. This Linux distribution's biggest appeals are the ease with which one can set up, deploy and adapt multiple architectures in a running enterprise and/or small network and the fact that it is an open source solution.

Being a fully fledged operating system, Security Onion can either be deployed as a set of virtual machines or actual physical workstations.

It combines multiple security components, such as the Elastic stack, Snort, Suricata, Bro, Sguil, Squert, amongst others, making it a very powerful tool in defending a network. It downloads an updated set of signatures every night using *PulledPork* so as to keep its set of rules ready and robust.

B. Architecture Review

Security Onion allows a user to deploy three different scenarios: distributed, heavy distributed and standalone. These rely on four fundamental node types which follow a *client-server* communication model.

A Master node, acting as a server, runs the *Elastic Stack* [2], which heavily relies on *Elasticsearch* [3] for communication with other nodes through queries using cross-cluster search [4].

Apart from the Master node, there's the possibility of using *forward nodes*, which, as the name implies, forward their data via an *auto-ssh tunnel* to the Master node, or a *storage node* if the Master node has been configured to use one.

There is also the possibility of deploying *Heavy nodes*, which combine forward and storage nodes. I.e., they run a local Elasticsearch instance and create an *auto-ssh tunnel* to the Master node, allowing it to construct a database by querying the Heavy node. One important aspect to note is that communication is based on ssh tunnels, which allows robust firewall rules, allowing a single port (22) to be opened on most nodes (other ports may be opened for visualisation purposes).

1) Distributed: A distributed scenario is the recommended deployment type. It consists of a Master server, one or more forward nodes, and one or more storage nodes.

2) Heavy distributed: This type of deployment consists of a Master server and one or more Heavy nodes. Due to its higher hardware requirements, it is only recommended if a standard distributed deployment is not possible. Figure 1 [5] shows the Elasticsearch based architecture one can find in a heavy distributed scenario.

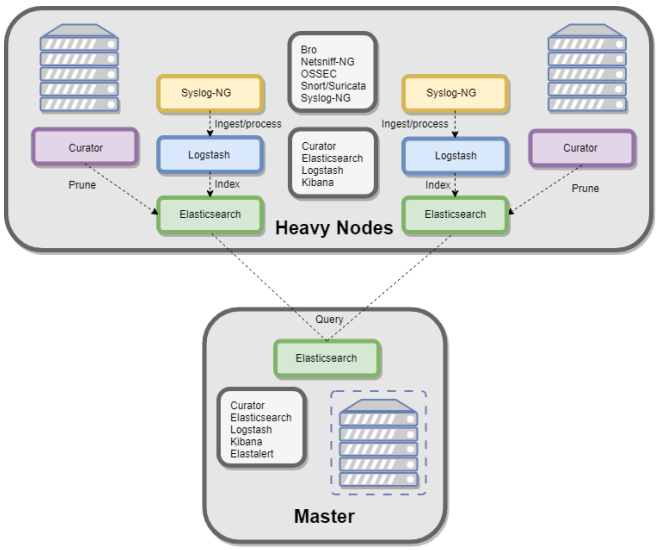


Fig. 1. Heavy distributed architecture

3) Standalone: It consists of a single server running master components, sensor and Elastic stack components. It is not recommended for monitoring high-throughput links (i.e., it's mostly used for very small environments and/or elaborating a proof of concept).

Due to the hardware requirements of Security Onion, and the characteristics of the working environment of the Netlab @ FEUP, a **heavy distributed** scenario was deployed on one of the workbenches. This allowed a distributed implementation while keeping hardware complexity to a minimum, thus allowing to simulate a small production deployment consisting of a Master server and one Heavy node. Details on the placing of this Heavy node, which acts mainly as a *sensor*, and the Master node can be found in subsection III-C.

C. Topology

In order to implement the architecture described in subsection III-B, it was strictly necessary for the Master to have a direct connection with the Heavy node, so that the queries requested by the Master were answered by the sensor with the least possible delay.

For that reason and according to the Netlab network topology, the easiest way to accomplish this requirement was connecting the Master to a Fast Ethernet port of the Switch. This port carries traffic at the nominal rate of 100 Mbit/s,

which is enough for this purpose since the only data that it will transmit through that port is related to the queries made to the Heavy node, unless there are some involuntary connections to it through external agents. Thereby, the Master has a layer 2 connection with the Heavy node that was configured in workstation gnu14 as the sensor that will *sniff* the data that goes through the 23 Fast Ethernet ports belonging to the Switch, using the SPAN (Switch Port ANalyzer) feature. However, the Gigabit Ethernet port used to connect gnu14 to the Switch is justified because the Heavy node should sniff as much data as possible. Ideally, this link would have a bandwidth of $23 \times 100Mbps = 2.3Gbps$.

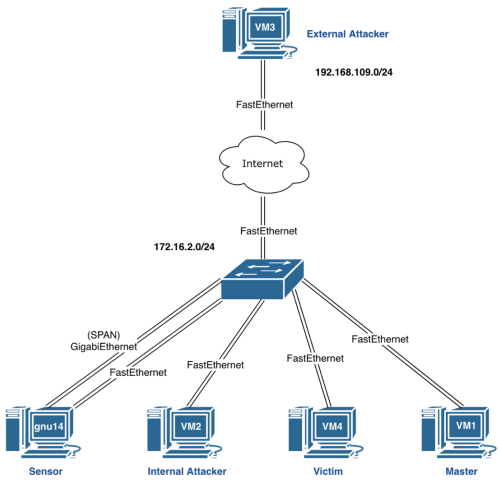


Fig. 2. Implemented network topology

Besides the Master and the Heavy node, the Figure 2 shows four virtual machines (VM) that was configured with the following operating systems: VM1 - Security Onion Master, VM2 and VM3 - Parrot OS/Kali Linux and VM4 - Metasploitable.

Virtual Machines 2, 3 and 4 were used to test the type of events that are detected by the security onion and the ones that are not, showing that there are multiple scenario in which ultimately it is up to the user to not disclose important information, to carefully select which services are strictly necessary and, in the event that they are, to not leave default configurations in, if any, vulnerable equipment.

The Metasploitable VM is a VulnHub, an intentionally vulnerable virtual machine designed for penetration testing, training and general target practice. This virtual machine was used as the target of the incoming attacks of two different workstations, one inside the private network and directly connected to the Switch (VM2), and another working as an external workstation (VM3).

IV. Evaluation

A. Architecture and Topology

Ideally, this deployment would contemplate a completely distributed scenario, one in which Storage nodes help 'alleviate' the load from the Master server, being the ones to store data and logs. Forward nodes, responsible for capturing and forwarding the data to the Master server, would be solely focused on sniffing packets and sending them away, not suffering in performance as much as a Heavy node does when considering this same functionality.

It would also be desirable to have the connection between the Master server and other nodes happen out-of-band, leaving it unaffected by congestion, delay increases and other network problems (which are a natural occurrence in populated networks). Redundancy should also be enforced.

Considering that we are in the presence of a small network, one in which not even all ports on the Switch are expected to be occupied, it is not mandatory to add other sniffing interfaces on the Heavy sensor, or even another Heavy sensor. If the network were to grow, this would undoubtedly be a concern.

If traffic originating in the Local Area Network was not a problem, the sensor could have been placed between the Switch and the Internet, acting as a relay. It should not cause significant overhead or delays since this connection type would originally be Fast Ethernet. This would significantly decrease the traffic load on the Heavy node to a maximum of 100 Mbps, and it should be noted that, so as to not share this connection (and as mentioned before in this subsection), the management connection should be adequately transferred to another place in the topology, preferably out-of-band.

B. Detected Events

Security Onion presents three graphical interfaces: Kibana, Sguil and Squert. The administrator can visualise the details about the logs created by the Heavy node.

After a few tests, the detection of some events, such as an ICMP request/reply (ping) and the access to a web page with filtered content, is automatically and immediately detected by the Heavy node and announced to the Master as it can be seen in Figure 3. The time passed since a malicious event hits the network until it is displayed in one of the Master node's dashboards is insignificant for human perception, but that delay increases as the network becomes more congested. The *timestamp* of the event, however, is preserved even if a an event appears delayed at the Master node. This happens because the response to the query requested by the Master to the Heavy node will suffer from that congestion. This delay can be reduced with the increase of the hardware specs available for Security Onion, like the number of CPU cores, and redundancy in the topology or an off-band approach in order to ensure a connection without congestion between the Master and the Heavy node (it will always depend on the time the Heavy node takes to process the query).

Kibana, a webpage that contains Squert and presents all types of data analysed by the SIEM, displaying it in dashboards, as seen in Figure 4, shows the results of ping floods in the network between two computers that are being monitored by the Security Onion's Sensor.

C. Limitations

1) Hardware: The performance of a Security Onion based approach is entirely dependant on the quality of the available

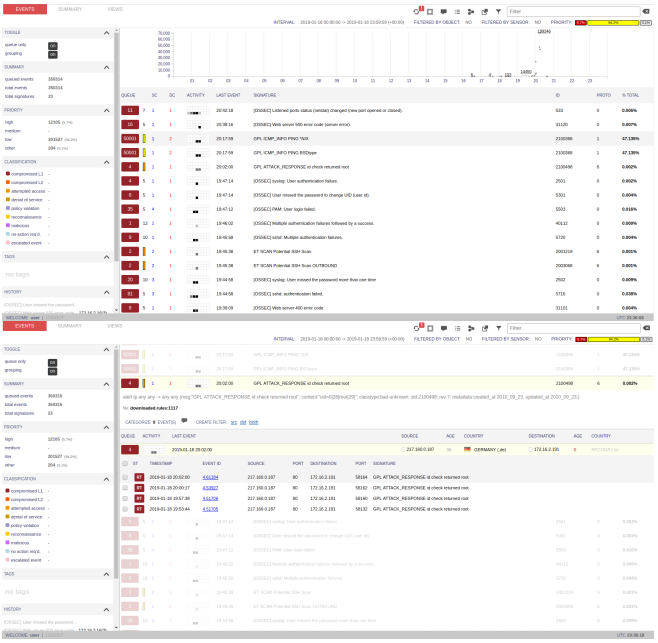


Fig. 3. Squert interface showing the detected events

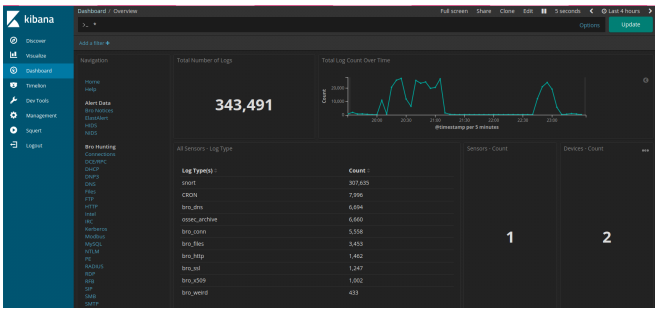


Fig. 4. Kibana interface showing the detected events

hardware. Any node that runs the Elastic stack is recommended to have at least 8 GB of dedicated RAM and 4 CPU cores, not taking into consideration that monitoring more traffic and consuming more logs increases these demands.

A simple calculation can be used to prove that dimensioning the hardware in which the nodes operate is vital to ensure a long term security solution. Considering that the port responsible for sniffing data can obtain a maximum of 1 Gbps, and that a malicious user can try to stress that link. Assuming a disk capacity of 100 GB which, while conservative, is how much was available to the Heavy node, we can say that:

$$\frac{100\text{ GB}}{1\text{ Gbps}} = \frac{100\text{ GB}}{125\text{ MB/s}} = 800\text{ s} = 13\text{ m }20\text{ s} \qquad (1)$$

Following Equation 1, we can see that, given the ability to stress the link of the Heavy node, it becomes easy to fill its entire disk capacity (in a short amount of time), rendering it unusable in the future or forcing it to lose stored data. This could be solved by means of deploying Storage nodes and backing data up regularly. In [5], an extensive list of hardware requirements is introduced.

Equation 2 considers a likely average of 1 Mbps of data rate in the Switch and shows that this implementation enables storage of the events that occur in the past 7 days.

$$\begin{aligned} 1\text{ Mbps} &= 0.125\text{ MB/s} = 10800\text{ MB/day} \\ &= 75.6\text{ GB/week} \end{aligned} \qquad (2)$$

2) Bandwidth: Considering that the link connecting the Heavy node to the network (in this case, a Gigabit Ethernet port on the Switch) only accommodates enough bandwidth to sniff the traffic of 10 stressed out Fast Ethernet links, this node should have at least another Gigabit Ethernet port using the SPAN feature. This requires, however, three Network Interface Controllers (NICs): one for management, and two for traffic analysis. Still, to make sure that all traffic is captured on a 24 Fast Ethernet port Switch with 2 Gigabit Ethernet ports only, **5** NICs are required to capture all traffic if it is being sent at maximum speeds (2 Gigabit Ethernet ports and 2 Fast Ethernet ports for traffic analysis, leaving one port for exchanging data with the Master node).

V. Conclusion

It is possible to conclude that the presented solution, through its distributed nature, can become highly scalable and reliable. Having various IDSs can help in detecting both anomaly and signature based events, as well as having more than one 'perspective' on analysed traffic and events.

In addition to this, the graphical features ported with Security Onion prove to ease its learning curve, providing meaningful, well organised data, coupled with the fact that open source projects usually have better support material, as is the case. It is also noteworthy that Security Onion is relatively easy to deploy, extending its capabilities to people who aren't experts and/or well versed in the area.

Overall, its performance is very satisfactory (can achieve sub-second display of alerts during normal network operation), considering the restraining hardware, which brings about one of the most important aspects in deploying such solutions: as it is highly dependent on hardware, the cost to deploy a strong, robust architecture and topology containing IDSs and a SIEM oftentimes does not justify resorting to such an elaborate solution.

References

[1] S. Cooper, "10 top network intrusion detection tools for 2018," Nov. 2018. [Online]. Available: https://www.comparitech.com/net-admin/network-intrusion-detection-tools/

[2] E. B.V., "ELK," 2019. [Online]. Available: https://www.elastic.co/elk-stack

[3] ——, "Elasticsearch," 2019. [Online]. Available: https://www.elastic.co/products/elasticsearch

[4] ——, "Cross Cluster Search | Elasticsearch Reference [6.5] | Elastic," 2019. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-cross-cluster-search.html

[5] L. Security Onion, "Linux distro for intrusion detection, enterprise security monitoring, and log management: Security-Onion-Solutions/security-onion," Jan. 2019, original-date: 2015-03-24T20:15:23Z. [Online]. Available: https://github.com/Security-Onion-Solutions/security-onion

Cybersecurity as a Game: Apache Web Server

Francisco Fraga
Master in Electrical and Computers Engineering, FEUP
Porto, Portugal
Email: franciscolfraga@gmail.com

Gonçalo Costa
Master in Electrical and Computers Engineering, FEUP
Porto, Portugal
Email: goncalofeup95@gmail.com

João Pereira
Master in Electrical and Computers Engineering, FEUP
Porto, Portugal
Email: up201303799@fe.up.pt

***Abstract*—As cybercriminals, attackers and hacktivist groups are evolving each day, the security industry has to expand itself in order to defend the world security as well as the main interests of companies and product consumers. Gradually, the battle between attackers and defenders has become identical to a game: New types of attacks lead to new defenses to reverse them. Security updates become outdated as soon as attackers find ways around them and a new product version must be developed in order to fix the failure. In the meantime, cyberattackers keep relying on social engineering moves that are hard to defend against. It is the duty of companies to continuously seek for breaches in their products, minimizing the probability of a hacker to find them before they do. That is when the Blue Team / Red Team game comes in hand.**

I. Introduction

When testing the effectiveness of security protocols within a company tool, test attackers must try to break any supposed protections in order to guarantee that they hold up. Considering the same perspective, defenders need to be aware of potential breaches in the system and be ready to take any kind of action to protect systems during and after the testing. Thus, we have the origin of two distinct teams of security professionals who are all working for the good of a certain company, but doing it in opposing ways.

This paper aims to provide theoretical feedback on how cybersecurity as game could be applied to an Apache HTTP Server. More specifically, it addresses the idea of having an automatic and programmed Blue Team / Red Team approach on the proposed tool.

II. Background

The terms red team and blue team are often used to refer to cyberwarfare in contrast to conventional warfare. War games function as a means of testing for the worst case scenarios of coordinated, focused attacks by skilled attackers.

Red Teams are the attackers. This team usually don't have information about the type of security is being used, that knowledge could lead to some attacks being automatically avoided because there is security in place and consequently to vulnerabilities being missed if that security isn't properly configured. Blue teams are the defenders. They continually attempt to harden security around and within the product data systems and networks, even when no testing is going on. They can also act as an active part of the defensive systems when the Red Team is attacking. Blue Teams act independently of the Red Team and therefore they can specialize in defensive operations entirely. Both teams will work together to provide a report of every test that was performed, what succeeded, what didn't, and why.

Some interesting definitions that could help the reader to better understand the paper contents, all taken from external sources referenced in the respective section, are:

- Metta is an information security preparedness tool. The project uses Redis/Celery, Python, and vagrant with virtualbox to do adversarial simulation. This allows you to test (mostly) your host based instrumentation but may also allow you to test any network based detection and controls depending on how you set up your vagrants. The project parses yaml files with actions and uses celery to queue these actions up and run them one at a time without interaction.
- Manual Red Teaming (MRT) is a traditional, human-intensive analysis effort to uncover system vulnerabilities or to find exploitable gaps in military operational concepts. The overall goal is to reduce surprises, improve and ensure the robustness of the Blue ops concepts. In this paper, an Automated Red Teaming (ART) framework is proposed. It is a simulation-based concept that uses Evolutionary Algorithm, Parallel Computing and Modelling And Simulation techniques to complement MRT and enable red teaming to be conducted in a more automated fashion.
- Apache Web Server is designed to create web servers that have the ability to host one or more HTTP-based websites. Notable features include the ability to support multiple programming languages, server-side scripting, an authentication mechanism and database support. The tool can be enhanced by manipulating the code base or adding multiple extensions/add-ons. It is also widely used by web hosting companies for the purpose of providing shared/virtual hosting, as by default, this Server supports and distinguishes between different hosts that reside on the same machine.
- The CVE is a vulnerability database designed to allow

other capabilities to be linked together, and to facilitate the comparison of security tools and services. A CVE listing only contains the standard identifier number with status indicator, a brief description and references to related vulnerability reports and advisories. It may partially contain risk, impact, recommendations or detailed technical information.

III. Solution

A realistic solution was needed in order to have the perfect conditions for a cyberwarfare. The Netlab was a good testing environment where three machines were connected, allowing them to play the game.

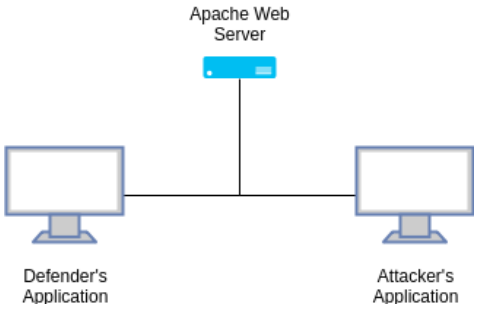


Fig. 1. The game network topology.

A. Blue Team

As mentioned earlier, this team has the duty of maintaining, upgrading, if needed, and fixing eventual errors in the functioning of the Apache Web Server itself. Summarizing, the machine whose is running the Defender Application has all responsibilities that the blue team should have in the traditional cybersecurity game. Taking advantage of automatic responses to certain inputs when the attacker is playing, the defender is running a Python application that continuously scans the server state, searching for eventual changes in order to be able to react. Since the defender is supposed to perform automatically, a set of actions were chosen in order to allow the team to protect the web server.

Label	Action
D0	Find out if anything has changed on the server.
D1	Upgrade to version 2.0.49.
D2	Upgrade to version 2.2.0.
D3	Upgrade to version 2.2.10.
D4	Upgrade to version 2.4.20.
D5	Upgrade to version 2.4.37.

TABLE I
LABEL AND DESCRIPTION OF ALL POSSIBLE DEFENDER ACTIONS.

A few versions were strategically selected to possibly run on the machine: 1.3.9; 2.0.49; 2.2.0; 2.2.10; 2.4.20; 2.4.37. For simplicity, we chose that the defender would only try to fix eventual attacker damages by means of a server software upgrade. Because some vulnerabilities demand specific Operative System versions, Docker was used in order to facilitate the change.

B. Red Team

Similarly to the Blue Team, the attackers' emulator is able to perform several automated actions. In order to do so, it scans the whole network and searches for any machine having an open port tcp/80.

scenario_example.yml 498 Bytes

```
enabled: true
meta:
  author: cg
  created: 2017-10-10
  decorations:
  - Purple Team
  description: Scenario Examples
  link: http://carnal0wnage.attackresearch.com
  mitre_attack_phase: null
  mitre_attack_technique: null
  mitre_link: null
  scenario: True
  scenario_actions:
    1: MITRE/Discovery/discovery_account.yaml
    2: MITRE/Credential_Access/credaccess_win_creddump.yml
    3: MITRE/Execution/execution_regsvr32.yaml
name: Scenario examples
uuid: 7da758ce-7c80-4169-a6ed-27abf3e5978f
```

Fig. 2. Example of an action configuration file.

It takes advantage of Metta, a tool written in Python that parses .yaml files which are automatically chosen depending mostly on the Apache Server version that is currently running on the target machine. As stated priorly, those .yaml files will instruct the attacker to perform any kind of action.

Label	Action
A0	Enumeration performing on the network.
A1	CVE-1999-1053 Metasploit module execution.
A2	CVE-2006-3747 Metasploit module execution.
A3	CVE-2007-6750 Metasploit module execution.
A4	CVE-2011-3368 Metasploit module execution.
A5	CVE-2017-9798 Metasploit module execution.

TABLE II
LABEL AND DESCRIPTION OF ALL POSSIBLE ATTACKER ACTIONS.

As an exhaustive research was conducted about the considered vulnerabilities, we feel that is better to use pre-existing tools in order to exploit them. Regarding the machine Operative System, we chose Kali Linux as it provides the necessary tools to perform the intended exploitations. Since several Apache related modules were found in the CVE website, Metasploit was used in order to perform the attacks undertaken by the Red Team.

IV. The Cybergame

The actual game starts with the blue team installing an initial version of Apache on the server machine. This version is randomly chosen from a list where all have in common the fact of being anyhow exploitable by the attacker. The next steps will be a set of actions taken alternately by both the blue and red team, which leads to a balanced state.

- The balanced state is when the attacker has no more actions left to perform in the current system. Summarizing,

the exchange of actions taken by both parts leads to a game over.

The following table contains the vulnerabilities, ordered by the exploit database, with the corresponding Apache HTTP Server versions the defender can install on the machine. All of them, except for the last released, are exploitable with a Metasploit module.

Versions	CVE	CVSS Score	Vulnerability Type
1.3.9;	1999-1053	7.5	Execute Code
1.3.9; 2.0.49;	2006-3747	7.6	DoS and Execute Code
1.3.9; 2.0.49; 2.2.0; 2.2.10;	2007-6750	5.0	Denial Of Service
1.3.9; 2.0.49; 2.2.0; 2.2.10;	2011-3368	5.0	Null
2.4.20	2017-9798	5.0	Null
2.4.37	Null	Null	Null

TABLE III
APACHE WEB SERVER CONSIDERED VULNERABILITIES WITH CORRESPONDING SOFTWARE VERSIONS.

In order to better understand the implemented code, a state diagram was designed with the set of actions the Blue Team can perform. Summarizing, the code continuously scans for any changes on the server machine and in the case of detecting anything odd, will upgrade it to a higher Apache version.

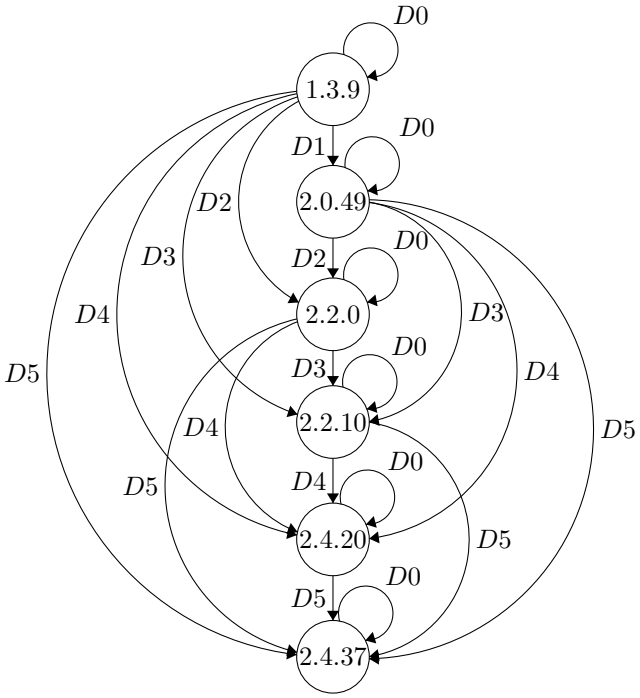


Fig. 3. State Diagram of the Blue Team actions.

In the other hand, as stated before, the red team can exploit several vulnerabilities. As briefly reported on table III some of them are in the range of a set of versions, which implies game states where a multiple choice can occur and more than one action is possible to be made. Naturally, the Metasploit modules we chose take advantage of different vulnerabilities, explained below:

CVE-1999-1053 The Matt Wright guestbook.pl CGI script contains a flaw that may allow arbitrary command execution. The vulnerability requires that HTML posting is enabled in the guestbook.pl script, and that the web server must have the Server-Side Include (SSI) script handler enabled for the '.html' file type. By combining the script weakness with non-default server configuration, it is possible to exploit this vulnerability successfully.

CVE-2006-3747 This module exploits the mod-rewrite LDAP protocol scheme handling flaw discovered by Mark Dowd, which produces an off-by-one overflow. It requires REWRITEPATH to be set accurately. In addition, the target must have 'RewriteEngine on' configured, with a specific 'RewriteRule' condition enabled to allow for exploitation. The flaw affects multiple platforms, however this module currently only supports Windows based installations.

CVE-2007-6750 Slowloris tries to keep many connections to the target web server open and hold them open as long as possible. It accomplishes this by opening connections to the target web server and sending a partial request. Periodically, it will send subsequent HTTP headers, adding to-but never completing-the request. Affected servers will keep these connections open, filling their maximum concurrent connection pool, eventually denying additional connection attempts from clients.

CVE-2011-3368 Scan for poorly configured reverse proxy servers. By default, this module attempts to force the server to make a request with an invalid domain name. Then, if the bypass is successful, the server will look it up and of course fail, then responding with a status code 502. A baseline status code is always established and if that baseline matches your test status code, the injection attempt does not occur. "set VERBOSE true" if you are paranoid and want to catch potential false negatives. Works best against Apache and mod-rewrite.

CVE-2017-9798 This module scans for the Apache optionsbleed vulnerability where the Allow response header returned from an OPTIONS request may bleed memory if the server has a .htaccess file with an invalid Limit method defined.

Due to lack of time, and having simplicity in mind, we chose to end the game when the server reaches the version 2.4.37. Naturally there would be infinite ways to approach the topic, mostly because of its complexity and community interest on hacking.

V. CONCLUSION

The main objective of this project was to automate a game between an attacker and a defender competing for either attack or protect a product. After an exhaustive research about the perfect server type to perform this kind of experiment and with several tools being taken in consideration we chose, in a first stage, a mail server (Dovecot). Due to our natural lack of knowledge regarding the topic in the course beginning and because some technical issues were found in old versions of

Dovecot we struggled to find the right tool for the job. Despite some failed attempts, we were able to match our requirements with the Apache Web Server.

All this barriers led us to a better understanding on the servers exploitations field. Furthermore, we could understand the advantages of performing this type of game on a company scenario. Because sometimes companies need to improve their product and they don't acknowledge a way to do it, nothing is better than an outside opinion in order to find a way to evolve and point out the product weaknesses.

In conclusion, besides all the knowledge we acquired throughout the course, it is important to state that the natural competitivity around the cybersecurity world (specially when automated) can lead us to evolution and failures mitigation with no space to entropy.

REFERENCES

[1] Security Intelligence. (2019). Checkmate: How to Win the Cybersecurity Game. [online] Available at: https://securityintelligence.com/checkmate-how-to-win-the-cybersecurity-game/ [Accessed 15 Jan. 2019].

[2] SecureAuth. (2019). Security in Plain English: What are Red, Blue, and Purple Teams? — SecureAuth. [online] Available at: https://www.secureauth.com/blog/security-plain-english-what-are-red-blue-and-purple-teams-0 [Accessed 15 Jan. 2019].

[3] Armerding, T. (2019). What is CVE, its definition and purpose?. [online] CSO Online. Available at: https://www.csoonline.com/article/3204884/application-security/what-is-cve-its-definition-and-purpose.html [Accessed 16 Jan. 2019].

[4] Cvedetails.com. (2019). Metasploit modules related to Apache Http Server. [online] Available at: https://www.cvedetails.com/metasploit-modules/product-66/Apache-Http-Server.html [Accessed 17 Jan. 2019].

[5] Help.ubuntu.com. (2019). HTTPD - Apache2 Web Server. [online] Available at: https://help.ubuntu.com/lts/serverguide/httpd.html [Accessed 16 Jan. 2019].

[6] GitHub. (2019). Metta, an information security preparedness tool. [online] Available at: https://github.com/uber-common/metta [Accessed 16 Jan. 2019].

Cyber Game Based on a DoS Attack

João Aires
MSc in Electrical and
Computer Engineering FEUP
Faculdade de Engenharia da Universidade do Porto
Email: up201404269@fe.up.pt

Miguel Barros
MSc in Electrical and
Computer Engineering FEUP
Faculdade de Engenharia da Universidade do Porto
Email: up201404278@fe.up.pt

***Abstract*—This paper was developed in the context of the Security in Systems and Networks course. The main objective was to develop a concept of Cyber Security as a game, specifically a DDOS attack and the respective defensive mechanisms. The attack strategy used was based on a Slowloris attack on a web server. The defence strategy was to change a dynamic timeout for the GET requests. The idea was to create a simple game with two entities, one attacker and one defender, that would try to win the game by making the attack unsuccessful without affecting the normal clients, in the case of the Defender, or by taking the web server down without using a high amount of resources, in the case of the Attacker.**

I. Introduction

With the growth of Internet, the number of Web server increased drastically and with them the necessity of implementing defensive strategies to overcome the well known vulnerabilities. The objective of this project was to develop a Cyber game using Slowloris, for the attacker, and a dynamic timeout for requests, for the defender.

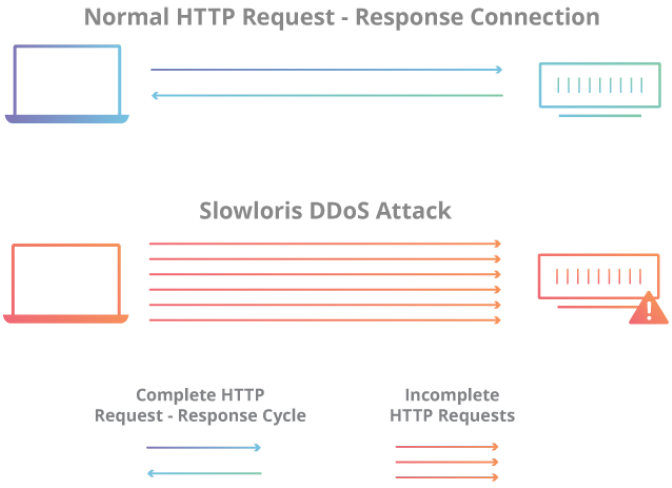
Regarding the defender process, the dynamic timeout was chosen due to the fact that it is simple and it is not a strategy that disable completely DoS attacks enabling the attacker to continue playing even though the timeout changed. The defender has only one parameter that can modify and it is the timeout for client requests. To prevent the defender from stopping the attacker permanently with a small timeout, normal clients will be implemented to measure if the strategy implemented is affecting clients that want to access the website. The server was to receive 100 connections per minute from normal clients that want to access the server resources, which must receive a response.

Regarding the attacker, Slowloris was the DoS tool chosen as it is affected by the timeout set by the defender allowing a game to be played on. Slowloris tests how many connections a server can handle by creating multiple slow connections that will continuously send *Keep Alive* messages to maintain the connection open, we can see an example in the figure 1. The attacker will be able to modify several parameters, among them:

- Number of Connections
- Interval Between Followup data
- Connections per second

The number of connections parameter establishes the number of connections opened during the attack. The interval between followup data is the interval that each connection will wait before sending again new messages. The last parameter defines the maximum number of connections per second.

Fig. 1. Example of a SlowLoris Attack. [1]



II. Related work

In order to understand the Slowloris attack some previous work was studied. Slowloris is a Denial of Service attack based on the vulnerabilities of the HTTP, implementing a slow HTTP DoS attack, this type of attacks makes multiple requests to the server causing the server to run out of the permitted HTTP open connections. [2] Slowloris starts with a legitimate HTTP message and after that continues to send HTTP GET or POST payload at a slow speed. [3]

The SlowLoris attack can be executed using Slowhttptest that implements a generic version of this DoS attack and it was the chosen tool for attacking the web server because of the ease of use and presentation of statistics of each attack performed.[4]

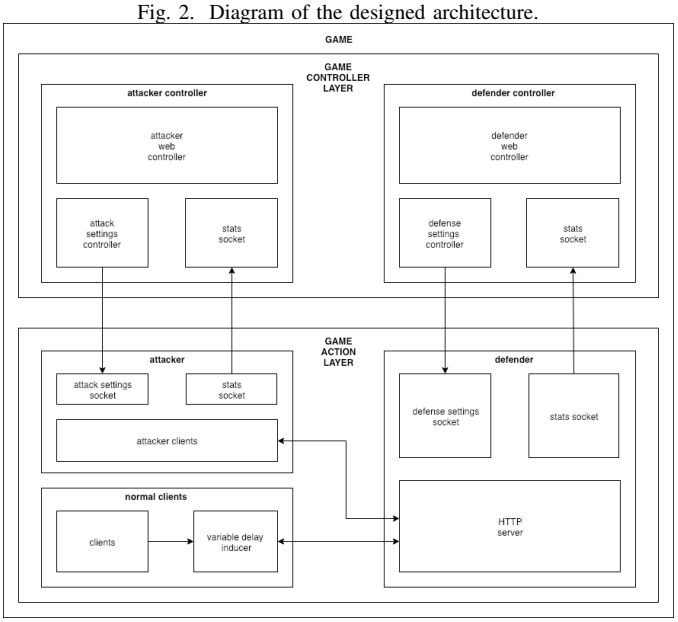
III. Project Description and Architecture

Before any simulations or tests could be performed, a *game model* that could comprehend all the settings and configurations this project intended to include, and simultaneously deliver the results and statistics, not only for the game's *gameplay*, but also for an objective analysis of the various simulations ran in this report, was designed and implemented.

A. Game Architecture

The architecture of the Game, which can be found in Figure III-A, was designed taking all the earlier requirements into consideration. It divides the elements of the game into two main layers: the Game Control Layer, and the Game Action Layer. The former, provides the platform used to allow players to follow the course of the game and adjust the settings according to the performance indicators presented. The latter is the layer in which the game controller's settings are applied and where the game actually takes place, including the attacker's tools and the defender's server and defence tools.

Fig. 2. Diagram of the designed architecture.



The Game Controller Layer was partitioned into two Controllers, functioning in separated environments, each allocated to one of the sides of the game: one for the Attacker and another for the Defender. Each was responsible for setting the corresponding settings, and displaying the metrics for the corresponding game side.

The Game Action Layer was also partitioned into the same categories, Attacker and Defender. The Defender consisted of an HTTP server with the necessary modules to receive the settings provided, namely the mentioned *timeout* attributed to each of the connected clients, and send the values for the required metrics, i.e. the number of normal clients and attacker clients connected. The Attacker consisted of a DoS Slowloris attacker, and, as previously stated, was responsible for the creation of the slow clients of the implemented HTTP server aiming to disrupt the delivery of the service provided by the HTTP server. Additionally, modules which allow the reception and update of the settings provided and the dispatch of the required statistics, are added.

To attribute the game a dimension of consequence which a real server would experience in the situation of a real Slowloris DoS attack, in the design of the game, a simulator of normal clients was included. Normal clients are not guaranteed to have small values of delay in their networks, as not all have the same network conditions, resulting in different values of delay of the clients. If an HTTP server sets a very low timeout value for each client after a TCP connection is successfully established, the connected normal clients in a less capable network might be lost by the server. For this reason, the resulting number of normal clients (good clients) after an attack is a and clear indicator of performance of the setting chosen for the defence mechanism, and will be examined later in Section IV.

B. Implementation

The implementation of the main elements of the architecture is here described and will cover the following aspects:

- Defender
- Defender Controller
- Defender-Controller Communication
- Attacker
- Attacker Controller
- Attacker-Controller Communication
- Normal clients

The Defender, as stated above, was configured as an HTTP server. The chosen language for the implementation was python using the *sockets* module, allowing lower level but higher flexibility configuration of the server, specially useful in the dynamic configuration of the *timeout* setting to be controlled. The concurrent handling of all HTTP requests and TCP connection was possible by the use of threads for each of the connecting clients.

The Defender Controller consists of an HTML page, accessed via a login page, common also to the Attacker Controller, that provided the user the both required functionalities. The statistics for the number of connections to the Defender were plotted in a live updating chart, with a refresh rate of 1s, i.e. upon a reception of a value from the server, and it was implemented using the HTML5 and JavaScript CanvasJS Chart libraries, which provided an easy to use and configure model for a live refreshing Chart. The configuration of the *timeout value* simply used a form to implement a button, which *on click* would call a JavaScript script to send the information to the Defender.

The Defender-Controller Communication was handled by WebSockets implemented in both the Defender's python server and in the HTML page, by JavaScript (JS) scripts. The statistics information was send periodically every 1s by the server. In turn, a JS script, which started on the HTML's *on load* moment, was responsible for listening for the messages from the server and act on them to update the chart as soon as the arrived. The setting, contrary to the statistics, were only sent after the HTML's button form pressed, which activated the JS script responsible for sending the server the current *timeout* value to be used. In turn, the python script would also be listening, via WebSockets for messages form the Controller to, as soon as a message as received, to change the current *timeout* value.

The implementation of the attacker used the *slowhttptest* tool [4], which provided a command line interface to run the intended attacks with the necessary configuration options for its use in this game design. This tool also provided it's own statistics for any given attack it performed, which was later used by the game controller given the quality of the provided statistics.

The Attacker Controller and the Attacker-Controller Communication, was also implemented using an HTML page, but it followed a different approach to presentation of the results and the configuration of the settings of the Attacker, as it used various other technologies, such as Flask and jQuery. It used a form system in which an attack started after a form was completed with the required information to be used and linked to a python script responsible for launching the the Attacker *slowhttptest* tool. The results are then return to teh Attacker Controller via a link to the generated HTML page (by the *slowhttptest* tool).

The Normal Clients followed the implementation of the Defender's HTTP server, and was also implemented in python using sockets, for similar reasons of the ones described in for the HTTP server. The different network conditions regarding the delay were implemented using a Delay Inducer, which in this case corresponded to a simple *sleep()* command with a random number of seconds, between 0 and 20 seconds, after a successful TCP connection to the server. Although simple, it provided the indented results for this application.

Another important aspect of the implementation was the presentation of the live value for both the Attacker and the Defender after an attack. This parameter was simply printed in the HTML pages both the Attacker Controller and the Defender Controller as the aftermath of an attack, and its calculation is explained in Section IV.

IV. Results and Analysis

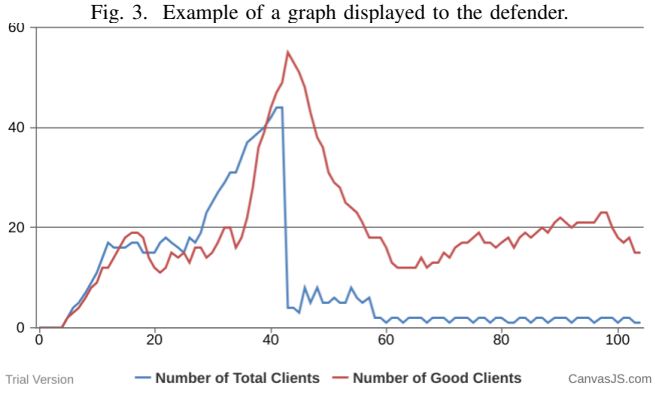
In this section it will be demonstrated the main parts of the game as well as statistics regarding the health of the players in different conditions and health penalisation formulas.

A. Player Statistics

1) Defender: During the game the defender is able to see in real time the number of total connection to the server as well as the number of good clients connected in a time instant. The defender can adjust the timeout value taking into consideration the ratio of Number of total connections vs Number of good client connections. In the case when the client detects a higher amount of total connection when comparing with the number of good clients it can be assumed that the server is under attack and he should change the timeout value in order to reverse the situation. In the figure IV-A1 it is possible to see an example of the real time graph displayed to the defender.

In the graphic IV-A1 we can see that the number of good clients vs total clients is always similar so there is no attack being performed on the network. At the time instant 40 it is possible to identify what happens when the timeout is set to 1 seconds the number of connected clients decline to almost 0

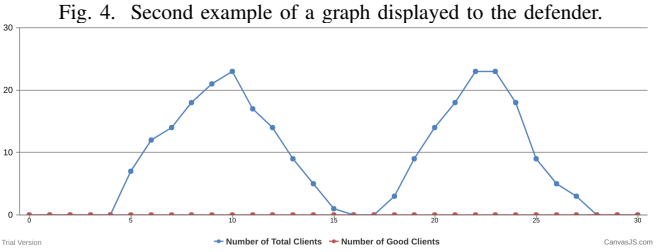
Fig. 3. Example of a graph displayed to the defender.



on the other hand the number of good clients take more time to decline since they wait a higher amount of time before considering the server is down. This situation represents well what happens when the defender sets up a low timeout, it will be able to stop the attack but will not be able to respond to normal requests.

The second graph IV-A1 exemplifies when an attack occurs, it is possible to verify that there are no good client connections but the number of total clients increases. The graph shows two attacks that occur on different time instances. With this information the defender can easily identify an attack and react to it in time.

Fig. 4. Second example of a graph displayed to the defender.



2) Attacker: The attacker can access the results of each attack, on the results there is information about:

- Amount of time that the server was down
- Number of connections accepted
- Number of connections pending
- Number of connections closed

In the graph **??**e can see an example of the results displayed to the attacker.

B. Health

To measure the formula to calculate the health from both attacker and defender it was decided to set a penalisation of 0.01 percent for each parameter but since the attacker has more parameters the game is not fair as we can see from the graph IV-B. With this formula the server would need 10000 dropped connections to reach 0.

To improve the fairness of the game we incremented the penalisation for the defender and with that we got a more closer game. IV-B

Fig. 5. Example of a graph displayed to the atacker.

Test parameters	
Test type	SLOW HEADERS
Number of connections	11
Verb	GET
Content-Length header value	4096
Extra data max length	52
Interval between follow up data	1 seconds
Connections per seconds	1
Timeout for probe connection	3
Target test duration	240 seconds
Using proxy	no proxy

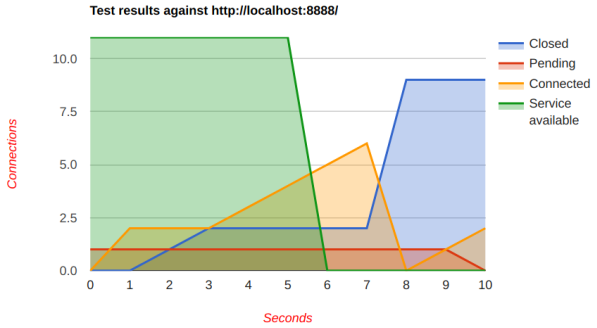


Fig. 6. Defender vs Attacker Health variation.

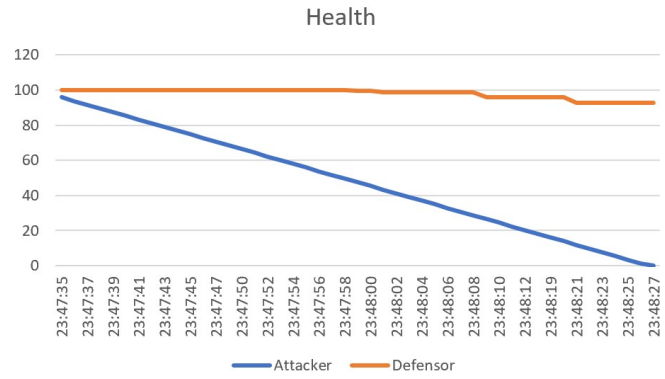
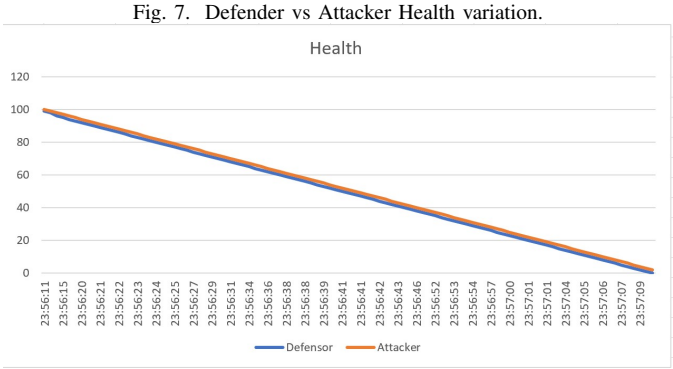


Fig. 7. Defender vs Attacker Health variation.



V. Conclusion

In this paper, a Cyber Game project was described, including the main required background for the understanding of its purpose. It also presented the implementation and the results obtained from a number of simulations that showed that it is possible to create a Cyber game based on a DoS attack. Further investigation is needed to understand deeply how it can be implemented with fairness for both the Attacker and the Defender.

References

[1] Cloud Flare, "Slowloris ddos attack," https://www.cloudflare.com/learning/ddos/ddos-attack-tools/slowloris/.

[2] I. Muscat, "Web vulnerabilities: identifying patterns and remedies," *Network Security*, vol. 2016, no. 2, pp. 5 – 10, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1353485816300162

[3] H. H. Jazi, H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Detecting http-based application layer dos attacks on web servers in the presence of sampling," *Computer Networks*, vol. 121, pp. 25 – 36, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128617301172

[4] shekyan, "Slowhttptest tool," https://github.com/shekyan/slowhttptest, 2013, online; accessed 18 January 2019.