NetwIOC: a framework for the automated generation of network-based IOCs for malware information sharing and defence

Submitted in fulfilment
of the requirements of the degree of

Master of Science

of Rhodes University

Lauren Lynne Rudman

Grahamstown, South Africa
30 November 2016

Abstract

With the substantial number of new malware variants found each day, it is useful to have an efficient way to retrieve Indicators of Compromise (IOCs) from the malware in a format suitable for sharing and detection. In the past, these indicators were manually created after inspection of binary samples and network traffic. The Cuckoo Sandbox, is an existing dynamic malware analysis system which meets the requirements for the proposed framework and was extended by adding a few custom modules.

This research explored a way to automate the generation of detailed network-based IOCs in a popular format which can be used for sharing. This was done through careful filtering and analysis of the PCAP file generated by the sandbox, and placing these values into the correct type of STIX objects using Python.

Through several evaluations, analysis of what type of network traffic can be expected for the creation of IOCs was conducted, including a brief case study that examined the effect of analysis time on the number of IOCs created. Using the automatically generated IOCs to create defence and detection mechanisms for the network was evaluated and proved successful. A proof of concept sharing platform developed for the STIX IOCs is showcased at the end of the research.

Acknowledgements

I would especially like to thank my supervisor, Prof Barry Irwin, for his guidance and support for the past two years.

I wish to extend my thanks to my family, pets and friends for keeping me going throughout the writing process of this document.

I would like to thank VirusTotal for the use of a private API key, which enabled me to gather malware samples for analysis during the course of this research.

This work was undertaken in the Distributed Multimedia CoE at Rhodes University, with financial support from Telkom SA, Tellabs/CORIANT, Easttel, Bright Ideas 39, THRIP and NRF SA (UID 90243). The authors acknowledge that opinions, findings and conclusions or recommendations expressed here are those of the author(s) and that none of the above mentioned sponsors accept liability whatsoever in this regard.

Contents

List of Code Listings

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

Many security breaches or intrusions on computer systems are not reported, never made public or even detected (Johnson *et al.*, 2014). This allows for attackers to potentially have free rein of victim's computers and networks, which would have a negative effect on organisations. When an organisation does find out about a compromised system or threat and responds accordingly, the information gathered may be valuable to others who experience a similar threat. This makes the sharing of information about an attack on an organisation, related to the detection and identification of threats, an important step in dealing with cyber-attacks (Zheng and Lewis, 2015). The more information that is shared about a threat, the easier it is to understand, track and counter.

As the number of new malware discovered increases every year, it is critical to have a scalable analysis and reporting solution to combat these numerous threats. PandaLabs found around 205,000 new malware per day in 2014, which was double the amount they found in 2013 (Panda Security, 2015). Kaspersky Labs detected around 200,000 new files per day in 2012, 315,000 in 2013, and 325,000 in 2014. The number decreased in 2015, with 310,000 new files per day (Kaspersky Lab, 2015). According to the AV-TEST Institute (AV-TEST, 2016) the number of new malware found in 2016 so far (August 2016), is at about 60 percent of the 2015 total amount. This shows the vast scale of the problem that is malware and how the number of new samples is not slowing down.

In Hun-Ya Lock (2013), it was discussed that there is currently a large amount of information on malware analysis and the tools and techniques that are used to perform

it. However there is a lack of research and information regarding the reporting of the results from the analysis. Hun-Ya Lock (2013) states that when combating malware, the reporting of results is just as important as the results themselves.

Dynamic malware analysis usually focuses on the behaviour of the malware on the system and has comprehensive reports generated about registry keys, files changed, DDLs and more, but there is limited information in reports regarding the activities on the network level. If anything, they give IP addresses contacted, which can include benign IPs, and domain names. There is a need for more comprehensive, human readable reports from dynamic analysis systems that give a focus on information about the malicious traffic malware generated. The information in these reports should be used detect and mitigate malware on a network level and be easily shareable.

A framework that can generate results in a standardised format for sharing and that can be machine and human readable, would be a useful tool. However, there is currently no accepted standard format for sharing detailed network indicators, but there are several formats getting accepted within industry.

1.2 Research Goals

1. The primary goal of this research was to automatically generate comprehensive network indicators from a given malware binary that can be used to mitigate and detect malware on the network. In order to achieve this goal, research had to be conducted on malware analysis, Indicators of Compromise (IOCs) and the types of formats available to represent them. The next part included finding a scalable way to generate indicators through the use of a malware analysis and the packet captures by analysis and correctly filtering certain packets. An IOC format had to be chosen and a framework developed that incorporates analysis and generation of indicators.

2. To run malware such as Dridex and numerous ransomware samples through the working framework to showcase the types of data that can be used in the network indicators and what the actual indicators in the chosen format look like.

3. Evaluate the indicator files for their use in the creation of IDS and firewall rules, as this could aid in the efficient scalable creation of rules. The first part of the process was to create a proof on concept sharing platform with the functionality to download information in IDS and firewall rule formats. The next part was to

evaluate the automatically generated Intrusion Detection System or Firewall rules to see if they are successful in detecting or mitigating malware. This was to show if the data stored in the indicators were useful or not.

1.3 Research Methodology

The research methods are based exploratory analysis using four of case studies which are all related to the automated generation of network IOCs from malware samples. Two of the case studies used in this work will be used to prove if the automated generation of IOCs is possible and will use exploratory analysis to explore what types of data will be present in the results created by NetwIOC. This will be achieved by creating the NetwIOC framework, and gathering malware samples from VirusTotal and then running them through the system. The results will be analysed in an exploratory manner.

The third case study will set out to determine the most efficient time that a malware sample should be run in order to generate network traffic which is sufficient in creating network IOCs. This will be achieved by executing the same samples for different duration's of time and analysing the exact amount of traffic created for certain network protocols at the different execution times.

The last case study will attempt to determine if the network IOCs created by NetwIOC can be turned into firewall rules that can stop the execution of the malware samples. This will be achieved by generating firewall rules from the network IOCs of 20 ransomware samples and determining if they are able to execute successfully after the firewall rules have been implemented.

Finally, a simple sharing platform for IOCs that was developed is showcased as a proof of concept. This will be to demonstrate that the IOCs generated by NetwIOC can be easily shared and be used in the real world.

1.4 Research Scope

This investigation will use the IPv4 address space, as it is more widely used for malicious traffic. The results produced for this research are a proof of concept as such. Network

indicators are created for the HTTP, DNS, FTP and SSH protocols, in addition to ICMP, TCP and UDP.

The operating systems that were chosen were Windows XP, 7 and 8.1. Windows 10 was not chosen as at the time of the experiments, it was not widely used.

The indicators are produced from dynamic malware analysis and not static, and are created in one type of IOC format, STIX[1]. One type of firewall and one type of IDS system was used for the analysis of rules created from the IOCs for defence.

1.5 Document Structure

This research thesis begins with a literature review in Chapter 2. This review includes a look at malware analysis in Section 2.1, dynamic analysis systems in Section 2.2 and traffic analysis techniques post dynamic analysis. The rest of the literature reviews is on ransomware in Section 2.4, Indicators of Compromise in Section 2.5, sharing cyber threat information in Section 2.7 and using IOCs for network defence mechanisms in Section 2.8.

Chapter 3 is about the setup of NetwIOC, which consists of Section 3.1 on the sandbox creation, Section 3.2 on the Virtual Machine creation and Section 3.3 on the design of a simple sharing platform.

The following chapter, Chapter 4, gives details on the logic behind NetwIOC regarding the IOC generation and POC sharing platform. It starts off with Section 4.1, which gives a comparison on two cyber observable formats, CybOX and OpenIOC. Section 4.5 describes how the IOCs were generated and Section 4.3 is about the STIX and CybOX objects used in the framework. Finally, Section 4.6 discusses the generation of IDS and firewall rules from IOCs.

Chapter 5 has three sections, with Section 5.1 discussing the datasets used in the chapter. Section 5.2 on a Dridex case study, Section 5.3 with a ransomware case study and lastly Section 5.4 with an analysis time case study.

Chapter 6 explores a network defence case study in Section 6.1 and considers the sharing platform in Section 6.2. Finally Chapter 7 concludes the thesis.

[1] https://stixproject.github.io/

Chapter 2

Literature Review

This chapter presents a review of the available literature on a number of topics relating to malware and indicators of compromise. It starts off with Section 2.1 on malware analysis, including the difference between dynamic and static analysis techniques. It next discusses the types of dynamic analysis systems such as TTAnalyse (Bayer *et al.*, 2006), CWSandbox (Willems *et al.*, 2007), DRAKVUF and Cuckoo and the analysis of traffic after dynamic analysis in Section 2.2.

Ransomware is explored, in Section 2.4, with a brief history, overview of how it works, its distribution around the world and the targets. Indicators of Compromise are explored in detail, in Section 2.5, beginning with an overview, and a discussion of the different formats they can be represented as. The automated generation of IOCs after traffic analysis is presented, in Section 2.6, with an overview of the types of sharing platforms for cyber threat information following. Finally an introduction to IDS and firewall rules is given, in Section 2.8, ending with an exploration of the conversion from IOCs to IDS and firewall rules.

2.1 Malware Analysis

Malware is defined as any malicious software that is designed to harm computer systems or networks (Santos *et al.*, 2009). There are a few primary sub classes of malware depending on what behaviour they have. These include Logic bombs, Trojan horses, Back doors, Virus, Adware, Droppers, Remote Access Trojans (RAT) (Aycock, 2006). New types of malware are being developed everyday and pose a security threat to everyone. To find a

way to detect and defend against newly discovered malware, security teams need to keep up with the new strains and perform analysis to learn more about the code (Santos *et al.*, 2009). As much information as possible needs to be known so antivirus and anti malware creators can create malware signatures or develop alternative ways to detect the malicious code.

Malware analysis is an essential part of combating new malware and is performed to learn more about a sample of malware and gather information on the damage done to the system. It is also performed to discover the Indicators of Compromise (IOC) (discussed in Section 2.5), to find the exploited vulnerability and to potentially identify the source of the code (Kendall and McMillan, 2007). It also helps to reveal emerging techniques used by malware developers and give insight on how to create appropriate countermeasures (Rossow *et al.*, 2011). The two types of malware analysis, static and dynamic, are discussed below.

2.1.1 Static Analysis

Static analysis entails analysing the code of the binary file, by reverse engineering it, without executing the file. It is considered safer than dynamic analysis because the code does not execute, so the malware can not call home, delete or encrypt files or steal information (Kendall and McMillan, 2007). One of the methods used in static analysis is file fingerprinting, which entails computing a hash of the sample using a common hash function used by researchers. Other static analysis methods include scanning the file with multiple anti-viruses, uploading the file onto sites such as Virustotal[1] or Metascan[2] which may already have information on the sample, packer detection, analysis of strings found in the source code, gathering metadata from the executable like the date and time of compilation and functions imported and exported.

A popular technique is reverse engineering, which consists of disassembling a binary and ending up with assembly code that can be understood by humans (Kendall and McMillan, 2007). Interpreting the assembly code can show an analyst exactly how a binary interacts with a system.

There are limitations of static analysis which are discussed by Raber and Laspe (2007), Sharif *et al.* (2008) and Gadhiya and Bhavsar (2013). One of these limitations include

[1]https://www.virustotal.com/
[2]https://www.metascan-online.com/

not having access to the original source code (typically written in a high level language which is easy to read) which limits what can be learned from a sample as a fair amount of malware use dynamic variables such as date and time or other system variables. Network communication logic is difficult to obtain if a sample is not executed while connected to a network, preferably the Internet. There are sometimes hard coded IP addresses and initial network connection information in the code, but if the malware needs instructions from the Internet to run, then static analysis can not provide insight. Static analysis can also prove to be hard if a malware author has used a range of binary obfuscation techniques. These techniques can obscure the meanings of API calls, function calls and mess with the control flow to confuse analysts (Sharif *et al.*, 2008). However there are some deobfuscation techniques, to solve this problem, that can be performed prior to static analysis such as Eureka presented by Sharif *et al.* (2008), one presented by Raber and Laspe (2007) and another technique presented by (Saidi *et al.*, 2010).

2.1.2 Dynamic Analysis

Dynamic analysis, on the other hand, is all about executing the malware and observing its behaviour on a system and network at run time (Sharif *et al.*, 2008). Since malware is created with malicious intent, it is generally a bad idea to perform dynamic analysis on someone's personal computer (Kendall and McMillan, 2007). Thus dynamic analysis is usually performed in a sandbox environment. This is a precaution taken in case the malware potentially deletes or encrypts files, changes the registry or even steals information.

A sandbox is a restricted execution environment that is run on a system that allows the safe execution of malware without effecting the host system (Oktavianto and Muhardianto, 2013). Some popular sandboxes to run malware in are Anubis, CWSandbox and Cuckoo Sandbox, which are discussed in Section 2.2. Sometimes a bunch of physical computers are connected on their own isolated network and malware is executed on the computers, but this is not as efficient as using a sandbox, because each time malware is run, the computer's Operating System (OS) would have to be restored.

Dynamic analysis with Virtual Machines (VM) can be automated, which means analysis can be done on a large scale (Gadhiya and Bhavsar, 2013). This method is faster than using physical computers (Kendall and McMillan, 2007) and as reported by Sharif *et al.* (2008); Rossow *et al.* (2011) dynamic analysis has had a better track record in the binary analysis sector and is a well established and effective tool.

The Windows API is used to access system resources of a sandbox like files, processes, the registry, network information and more. A process called API hooking is used, where a sample in the sandbox calls a function, it is rerouted to customised code (the hook) which then performs an operation and finally transfers control back to the original API call. Another method to observe system behaviour is Dynamic Link Library (DLL) injection, where the hook functions are copied to the samples address space so that the target can call them Willems *et al.* (2007).

As stated in Rossow *et al.* (2011), malware's behaviour observed greatly depends on the duration of the analysis. If the malware's network behaviour is of interest, then the dynamic analysis must be run for longer than the bootstrapping process, which is usually a few minutes (Rossow *et al.*, 2011). In a study on a system called Sandnet (Rossow *et al.*, 2011) that dynamically analyses malware, the samples were run for 5 minutes and at least an hour and found that only 6.1% of all network flows started in the first five minutes.

Obfuscation also does not usually effect the dynamic analysis process (as much as static analysis) because the binary is executing on the system so it should show its intended behaviour (Sharif *et al.*, 2008). Some sophisticated malware contains anti-analysis techniques such as virtual machine detection, where the malware will not function normally or stop working according to Saidi *et al.* 2010. This malware can detect if it is being executed in a controlled environment and will be discussed in Section 2.1.3 (Gadhiya and Bhavsar, 2013). This may be avoided by running malware on a physical computer attached to a virtual Internet, booting into Linux and analysing the Windows partition. However, the behaviour recorded on the network may not be sufficient and the method does not scale very well (Gadhiya and Bhavsar, 2013). This method also does not get to see and track the malware as it runs, it just sees a snapshot.

2.1.3 Malware Anti-debugging and Anti-virtualization

Dynamic malware analysis gives rich information about malware run time behaviours (Chen *et al.*, 2008). This is why some malware authors use evasion techniques to avoid accurate analysis (Branco *et al.*, 2012). These techniques include, code to detect VMs or debuggers or even detect disassembly attempts (Branco *et al.*, 2012). VMs and debuggers are fingerprinted by checking OS objects or looking at Central Processing Unit (CPU) instruction execution time. If the malware detects that it is in a hostile environment it can stop working or change its behaviour to avoid revealing its intended goal.

Some anti-debugging techniques suggested by Chen *et al.* (2016), include using the Windows API, checking flags in the Process Environment Block (PEB) structure and using certain instructions to trigger characteristic debugger behaviour. Anti-VM techniques include checking interactions such as mouse movements and mouse clicks and finding certain artefacts in service lists or registry keys. Some malware is even configured to execute after a given time or after a few sleep cycles, because VM analysis is usually a few minutes.

In a paper by Chen *et al.* (2016), that investigated the use of anti-VM and anti-debugging techniques in modern malware, it was found that there is a decrease over time of anti-VM techniques used in malware such as Advanced Persistent Threats (APT). APTs usually target companies in critical sectors and governmental institutions (Chen *et al.*, 2016).

According to the twitter page of DMA Locker[3] (a type of ransomware), many companies work with virtual machines and therefore they contain important files. There would be no point of this malware strain not executing in a VM. They also implied that some malware does not use anti-VM techniques because skilled reverse engineers can bypass them in a few hours[4].

2.1.4 Network access

Analysis can be performed with Internet access or without. Some of the possible disadvantages of Internet access suggested by Kendall and McMillan (2007) are that control of the virtual machine may be lost if an attacker gains access to it or if an attacker sees a connection from a machine they did not hack, they may act differently or the malware may participate in a DDoS attack. Inoue *et al.* (2008) states that analysis with Internet access could cause unwanted distribution of self propagating malware. If a proxy or VPN are not used then the attacker may learn the external IP address of your machine. However, denying a sandbox Internet access usually results in incomplete behavioural analysis of the malware (Egele *et al.*, 2012).

In terms of collecting information, an advantage of allowing Internet access is to potentially have access to files malware downloads. It can be absolutely necessary for malware to have Internet access to receive commands, update or perform other actions (Katsamakis, 2014). A lot of malware is modular, which means that it needs to download additional components, updates or other data before functioning maliciously (Egele *et al.*, 2012).

[3]https://twitter.com/dma_locker
[4]https://twitter.com/dma_locker/status/726008067129806848

The behaviour of the secondary files can also be observed to give greater insight into how the malware works as a whole.

Two common solutions to the Internet access problem are giving the sandbox access to a simulated network or allowing filtered Internet access. Simulating network services can be done with programs like INetSim[5] or FakeNet[6] and may trick the malware into exhibiting malicious behaviour, but does not allow malware to update or allow for bots to receive commands (Egele *et al.*, 2012). Tightly monitored limited Internet access would allow malware to update and receive commands, but with a limited bandwidth, a DDoS attack would not be significant. This would be a sensible option when focusing on the network behaviour.

2.2 Dynamic Analysis Systems

Since dynamic analysis has been a crucial part of malware analysis, a number of systems have been developed in order to tackle the problem. Most of the systems consist of a Sandbox environment and a tool that can record behaviour from the sandbox.

There are a number of online malware analysis systems, such as Anubis, DeepViz[7], ThreatExpert[8], Hybrid analysis[9], reverse.it[10] and Metascan Online[11], but these do not scale as they sometimes limit submission speed and the time results are given. Some of the web applications limit the size of the binary being submitted (Qiao *et al.*, 2014). With TTAnalyze, Anubis and CWSandbox, the source code or local packages are not available. The Cuckoo Sandbox however, is open source, which can be customised at will. DRAKVUF is in early development and does not provide comprehensive reports for analysis and generation of IOCs.

2.2.1 TTAnalyze

This is a prototype system that was created by Ulrich Bayer in 2005 for his Master's thesis and allows an analyst to get a brief understanding of a malware's behaviour by

[5] http://www.inetsim.org/
[6] http://practicalmalwareanalysis.com/fakenet/
[7] https://sandbox.deepviz.com/
[8] http://www.threatexpert.com/submit.aspx
[9] https://www.hybrid-analysis.com/
[10] https://www.reverse.it/
[11] https://live.metascan-online.com/

automating the process of analysing malware (Bayer *et al.*, 2006). The final product is a report that gives information about the malware's actions and can be used to infer the purpose and functionality of a sample. TTAnalyze uses an Operating System (OS) emulator to run samples and not a virtual machine. OS simulators process instructions with software, unlike virtual machines which run directly on the underlying hardware.

Unlike many other dynamic analysis frameworks, TTAnalyse does not use a virtual machine, API hooking functions or debuggers. Using an emulator makes it more difficult for a sample to detect that it is not being run on a physical computer. Although, if malware is sophisticated enough, it may realise that the processing speed is slower than it should be as CPU emulators are not as quick as a physical CPU. According to Bayer *et al.* (2006), the presence of TTAnalyze is practically invisible to a sample. The system can monitor Windows API calls and native kernel functions along with the parameters passed to them. Instead of using hook functions, TTAnalyze can perform function call injection and alter the execution of a sample. This system was later updated and dubbed "Anubis", which will be discussed in Section 2.2.2.

2.2.2 Anubis

Anubis[12] is an online malware analysis site that accepts Windows executables, Android APK files and suspicious URLs (Bayer *et al.*, 2009). It was developed from TTAnalyze and has evolved to be user friendly with the web application submissions and a generated report being sent via email. It is not open source, but it is free to use. The operating system that samples are run on is Windows XP on an emulator called QEMU [13]. In general it is an improved user friendly version of TTAnalyze (Bayer *et al.*, 2009).

2.2.3 CWSandbox

This analysis tool first came about in 2007 and is a system that executes a malware sample in a sandbox environment and monitors system calls and eventually creates a detailed report. In the first paper written by Willems *et al.* (2007), the CWSandbox could perform analysis on more than 500 binaries a day, which according to them is at least an order of magnitude faster than human analysis. Real time analysis in the sandbox is performed by doing API hooking as discussed in Section 2.1.2. The system has three parts, initialisation

[12]https://anubis.iseclab.org/
[13]http://wiki.qemu.org/Main_Page

where the system sets up the malware process and installs API hooks, execution where if everything has initialised correctly the malware is executed, and analysis where the sandbox analyses the collected data from the hook functions and generates an XML report (Willems *et al.*, 2007). The sandbox was successful in automating the analysis of Win32 malware.

In 2013 CWSandbox became "ThreatAnalyzer"[14] and now supports analysis across multiple Operating Systems, patch levels and system configurations. It also has a wider array of generated reports, including PDF, HTML, XML, JSON and PCAP. However it is not open source so for research purposes this is not a good option.

2.2.4 DRAKVUF

DRAKVUF[15] is a open source dynamic malware analysis system built with an emphasis on stealth (Lengyel *et al.*, 2014). It is still in an early development stage. It uses hardware visualisation extensions found in specifically Intel CPUs with virtualisation support (VT-x) and with Extended Page Tables (EPT)[16]. It hides the analysis monitoring environment by using the effectiveness of visualisation extensions (Lengyel *et al.*, 2014). Like some of the previous systems it uses virtual machines. DRAKVUF also has a tamper resistant monitoring engine that can provide insight into user and kernel-mode malware and rootkits. It uses an active Virtual Machine Introspection (VMI) and other techniques to hijack a random process in from the VM to initiate the start of the malware sample. By doing this DRAKVUF does not introduce new code into the analysis VM, which makes it stealthier.

2.2.5 Cuckoo Sandbox

Cuckoo[17] is an open source automated malware analysis system that provides fast and complete analysis results (Provataki and Katos, 2013). The setup of Cuckoo used for the rest of this thesis are discussed in Section 3.1. Cuckoo has an online web service called Malwr[18] were samples can be submitted, but there is no say in which operating system is being used or networking options. Thankfully Cuckoo is open source and can

[14] http://www.threattracksecurity.com/enterprise-security/malware-analysis-sandbox-tools.aspx
[15] http://drakvuf.com/
[16] http://www.xenproject.org/developers/teams/hypervisor.html
[17] https://cuckoosandbox.org
[18] https://malwr.com/

be set up locally on a server and a custom configuration can be chosen. Files that can be input into Cuckoo are Windows executables, DLL files, PDF documents, Microsoft Office Documents, URLs and PHP scripts. After analysis, Cuckoo generates a PCAP file of captured packets and a report which includes screenshots, static analysis results, dropped files, DNS and HTTP requests and a behaviour summary. Cuckoo can deploy multiple Virtual Machines and run them simultaneously, which guarantees efficiency (Qiao *et al.*, 2014). It has a feature called 'trampoline' which was implemented to make the analysis environment harder to pick up for the malware. Cuckoo injects a DLL file into the memory of the sample which enables it to hook the original API calls for tracking (Qiao *et al.*, 2014) and has less hooks than CWSandbox.

Cuckoo consists of modules, including a processing[19] module that can perform static malware analysis on the binary (Cuckoo Foundation, 2015a). Static analysis of a binary can give interdependent insights to dynamic analyses (Sharif *et al.*, 2008).

2.3 Network Traffic Analysis

Once dynamic analysis is completed on a sample, there is usually a report on the behaviour of a sample with registry keys added, files added, the hash of the sample, hosts contacted and more. However this information can only be useful if it is shared or used to create defence mechanisms against the malware. The first step in doing this is to analyse the results of the analysis to find useful information. Below are a number of studies or systems that focus on the analysis of the results of dynamic analysis with regards to network traffic.

2.3.1 Multiple executions

In a study conducted by Provataki and Katos (2013), where a single malware sample was executed multiple times, an extension was added to the Cuckoo Sandbox in order to correlate and investigate the Cuckoo result reports of the multiple executions. The extension ran after dynamic analysis. This framework mainly focuses on the system behaviour of the malware but briefly looks at the network activity. Samples were run a certain number of times with Internet access and the same number without Internet access. This was done with the intention of observing any behavioural changes that may occur.

[19] http://docs.cuckoosandbox.org/en/latest/customization/processing/

In terms of the network results, the Cuckoo extension is able to display a certain connection and show which execution of the samples generated it.

2.3.2 Sandnet

In a paper by Rossow *et al.* (2011), a system called Sandnet was developed to dynamically run malware in a sandbox and record the network traffic. It was found that the duration of the analysis effects the amount of traffic as some malware does not communicate with the network immediately. After the network traffic of multiple samples were recorded, the authors performed an analysis on the HTTP and DNS traffic. The traffic capture files were generated from more than 100,000 malware samples with the analysis taking around 12 months to complete. Each sample was run through the dynamic analysis system for around an hour. The virtual machines used for the sandbox had Windows XP SP3 and used VirtualBox as a hypervisor.

The system had a NATed Internet connection and used a preconfigured local DNS resolver. To limit the amount of potential damage of giving the sandboxes Internet access, certain traffic is redirected to local sink holes or honey pots. The network bandwidth, concurrent connections to each sandbox and packet rate is limited in case of a DDoS attack. Each PCAP file generated is converted into a UDP/TCP flow. A TCP flow is usually made up of a tuple including the layer 4 protocol, source IP, destination IP, source port and destination port. A UDP flow is a steam of packets ending by a five minute inactivity period.

Out of 104,345 samples tested, 43.8% of them exhibited network activity. From the 43.8% more than 70million flows were generated. By looking at the HTTP and DNS protocols, 92.3% used DNS, 58.6% used HTTP, 8% used IRC and 3.8% used SMTP. A in depth analysis of the use of the DNS and HTTP protocols are presented, which shows that some malware does not use the preconfigured resolver for domain resolution but may have a hard coded one. It was also found that for the DNS resolution, 10% of the resolved domains have their TTL value set to five minutes or below, which could indicate a fast flux domain. More results include 65% of samples generating more than 5 HTTP requests and 16.3% made one HTTP request and then stopped. Also 29.9% specified the wrong operating system or Windows versions in their HTTP User-Agent headers.

This study shows that dynamically analysing malware with careful Internet access can provide useful network communication information. Since there is useful information generated, it can be used to generate IOCs.

2.3.3 DNS Analysis

In a Doctoral research paper written by Binsalleeh (2014), a framework was developed for the analysis of reports from dynamic malware analysis tools. After dynamic analysis, all the DNS, HTTP, FTP, Simple Mail Transfer Protocol (SMTP) and IRC data were extracted from the traffic captures.

One of the main focuses of the thesis is the analysis of DNS traffic and uses domain names given by name servers to extract indicators of malicious behaviour. The indicators are to indicate the maliciousness of domain names and are intended to be used in a severity rating system for name servers. The thesis does not focus on the local system events of the sandbox while the malware is executing, but the network activity only.

2.4 Ransomware

Ransomware is a variation of malware that restricts a user from accessing part or the whole of their system. If users want to access their system or files again, they have to pay a fee, usually using an online payment method. In a news article by Trend Micro (2014), it stated that new strains are emerging every day, which makes it tricky to keep up to date with the latest samples. There are two major types or ransomware, one that encrypts files (crypto ransomware) and one that locks a computer (locker ransomware).

With locker ransomware, a ransom letter is displayed on the screen explaining that the only way to gain access to the computer or files is to pay the ransom fee. It usually leaves the underlying system and files untouched, unlike crypto ransomware (Bhardwaj *et al.*, 2016). This malware can usually be removed using anti-malware software, with the computer potentially being restored to its normal state (Savage *et al.*, 2015).

On the other hand, crypto ransomware finds and encrypts, what it considers to be valuable data found on a computer system. The ransom note claims that the only way to decrypt the files is by paying the ransom money and getting the decryption key. In some cases, decryption keys have been found by researchers in the malware binary itself (Doniec, 2016), or they have found other ways to combat the encryption and developed tools. Examples are Talos TeslaCrypt Decryption Tool[20], an online decryption tool for Petya[21]

[20] http://www.talosintel.com/teslacrypt_tool/
[21] https://petya-pay-no-ransom.herokuapp.com/

and NoRansom[22] created by Kaspersky. Crypto ransomware does not usually target critical system files but encrypts user files so the computer functions normally, except for the user not being able to access files such as documents, pictures and more.

Some ransomware uses Tor[23] to hide their Command and Control (C2) communications and make a user download and go to a certain URL for further instructions and to pay the ransom fee (Trend Micro, 2016). Locker ransomware is found to prefer payment voucher systems and crypto ransomware is found to prefer cryptocurrencies as the preferred payment method (Savage *et al.*, 2015). During the first few years ransomware targeted a few file types such as .DOC, .DLL, .EXE and a few more. In recent years the list is considerably longer (Trend Micro, 2016).

Ransomware is most effective on people with low computer literacy skills and who do not know about ransomware. Businesses are also good targets because they usually contain critical data needed for it to function.

2.4.1 Brief history

In 2005, the first cases of ransomware started occurring, but they were not as sophisticated as modern variants. In less serious cases starting in 2005, malware would be in the form of a misleading application that poses as a spyware removal tool or performance enhancer tool that exaggerates the bad state of the computer and promises to resolve the issues if the user pays a fee. In May 2005, crypto ransomware that used weak custom encryption techniques appeared in the form of GDP Coder (Savage *et al.*, 2015).

In 2006, there were incidents in Russia of CrypZip, where the ransomware zipped certain files and locked them with a password. It would also leave a notepad file with instructions on how to unlock the files for a fee of $300 (Trend Micro, 2016). Also in 2006, was Archiveus, which was similar to CrypZip, but did not ask for money in turn for the password. Instead the user was prompted to buy medication from online pharmacy URLs, which allowed the attacker to gain commission for the sale. The victim would have to submit the order ID to get the password.

From 2008 to 2009 most ransomware was in the form of fake antivirus software, which looked almost identical to some modern antiviruses. It would perform fake scans and give

[22]https://noransom.kaspersky.com/
[23]https://www.torproject.org/

fake reports stating that the computer had a large number of security issues and threats. The user would then be prompted to pay a fee to get rid of the fake problems. But 2008 was also the year when the first type of locker ransomware appeared in the form of Trojan.Randsom.C[24].

In Russia during 2011 there was a report of ransomware that disables the functionality of a computer unless the user dials a premium-rate SMS number and pays around $12 ransom. In the same year, locker ransomware was at its peak. Some ransomware could even effect the Master Boot Record (MBR) of a computer, which in turn prevents the operating system from loading and displays a ransom letter.

According to Trend Micro (2016), in 2012, there was a spread of ransomware called Reveton, that locked a user's computer and impersonated law enforcement agencies. The ransomware displayed a notification from a victim's local law enforcement agency that informed them that they were caught doing a illegal activity online. Users are prompted to pay a fine through UKash, PaySafeCard or MoneyPak, which are electronic money systems where vouchers can be bought.

In 2013 one of the most well known types of ransomware, Cryptolocker, showed up on the scene. This is one of the first types of ransomware to encrypt files, and even if the malware was removed, the files remained encrypted. In recent years more encryption ransomware started to be found, such as CryptoDefense, CryptoWall, BitCrypt, Onion, Locky, TeslaCrypt, TorrentLocker, CTBLocker and more. Onion ransomware was the first known variant to use Tor (most modern ransomware use Tor). Early June 2014, ransomware found its way onto mobile platforms (Trend Micro, 2014) and between 2013 and 2014 there was a 250% increase in new crypto ransomware families (Savage *et al.*, 2015).

2.4.2 Distribution

As noted in Savage *et al.* (2015), the top countries affected by ransomware were Japan, United States, United Kingdom, Italy, Germany and Russia. Ransomware has many ways of propagation, namely SPAM campaigns with malicious attachments, pretending to be a software activator on a P2P file sharing site, malvertisement, social engineering and self propagation. Most ransomware infections start with a dropper file, which then downloads the payload.

[24]https://www.symantec.com/security_response/writeup.jsp?docid=2008-010515-1023-99

2.5 Indicators of Compromise

An Indicator of Compromise is defined by Harrington (2013) as "a piece of information that can be used to identify a potentially compromised system" or "forensic artefacts of an intrusion that can be identified on a host or network" as defined by Mandiant (2012). IOCs are made up of one or more observables combined with contextual information. Observables are stateful properties or measurable events, such as IP addresses, domain names, email addresses, file hashes, file mutex and more (Barnum, 2014). There are a number of formats that were created with the goal of facilitating information sharing (including indicators) across the cyber security community (Obrst *et al.*, 2012), but no accepted standard format exists yet. These include OpenIOC[25], Cyber Observable Expression (CybOX)[26] together with Structured Threat Information Expression (STIX) and IODEF (Danyliw *et al.*, 2007), which will be explored below.

Traditionally malware is described by a hash value, but with the existence of polymorphic and metamorphic code (Lee *et al.*, 2010), there is a need to identify malware through semantics and not only through its syntax (Hun-Ya Lock, 2013). IOCs give a more comprehensive view of a threat by being able to describe behavioural components of the malware, on the network and on a system.

2.5.1 OpenIOC

OpenIOC is an open source framework for sharing intelligence related to cyber threats (Hun-Ya Lock, 2013). The IOCs are created in an XML format, which is advantageous because it is machine and human readable (Hun-Ya Lock, 2013). Mandiant[27] developed OpenIOC to be an open standard format for sharing. It includes 37 schemas that can describe "over 500 facets of environments that can be used to track down advanced attackers"[28]. The low level attributes of the OpenIOC format, such as IP address or port number can be used as the input of security monitoring systems like Intrusion Detection Systems (IDS) or Intrusion Protection Systems (IPS).

The OpenIOC format allows for the use of logical operators such as OR and AND to indicate the relationship between mandatory or optional IOC attributes. This allows for

[25] http://openioc.org/
[26] https://cybox.mitre.org/
[27] https://www.fireeye.com/services.html
[28] http://www.openioc.org/

the malware's behaviour to be described flexibly as complex semantics (Hun-Ya Lock, 2013). There is an example of the OpenIOC format shown below in Figure 2.1. An indicator is made up of one or many `IndicatorItems` grouped with AND or OR operators. The `IndicatorItems` should indicate the attributes of a behaviour and the indicators themselves indicate the behaviour (Hun-Ya Lock, 2013).

Figure 2.1: Example of OpenIOC format

2.5.2 IODEF

The Incident Object Description Exchange Format (IODEF) format (Danyliw *et al.*, 2007) was developed in the early 2000s. Its goal is to define a common data format for describing and exchanging information on cyber incidents. It is mainly aimed at Computer Security Incident Response Teams (CSIRT) and uses the XML format (Cover, 2008). Since the main purpose is to describe incidents, IODEF is not really meant to represent one IOC like an IP address, but it can be used to describe an event as a whole (Danyliw *et al.*, 2007). Figure 2.2 shows part of an IODEF incident. Technically some of the values in this incident can be viewed as indicators.

```
<?xml version="1.0" encoding="UTF-8"?>
<IODEF-Document version="1.00" lang="en" xmlns="urn:ietf:params:xml:ns:iodef-1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:schema:iodef-1.0">
  <Incident purpose="reporting">
    <IncidentID name="csirt.example.com">189493</IncidentID>
    <ReportTime>2001-09-13T23:19:24+00:00</ReportTime>
    <Description>Host sending out Code Red probes</Description>
    ...
    <EventData>
      <Flow>
        <System category="source">
          <Node>
            <Address category="ipv4-addr">192.0.2.200</Address>
            <Counter type="event">57</Counter>
          </Node>
        </System>
        <System category="target">
          <Node>
            <Address category="ipv4-net">192.0.2.16/28</Address>
          </Node>
          <Service ip_protocol="6">
            <Port>80</Port>
          </Service>
        </System>
      </Flow>
      ...
    </EventData>
  </Incident>
</IODEF-Document>
```

Figure 2.2: Example of the IODEF format

2.5.3 CybOX

CybOX stands for Cyber Observable Expression and is now a community driven effort, originally developed by MITRE[29], but is now in the hands of OASIS[30]. CybOX is backed by the U.S. Department of Homeland Security (DHS) office of Cybersecurity and Communications (Casey *et al.*, 2015). According to Obrst *et al.* (2012), the team that developed MAEC (presented in Section 2.6.4) started off using OpenIOC objects in their framework, but subsequently were not satisfied, and used the OpenIOC format as a starting point for the creation of CybOX.

It is designed for capture, specification, characterisation and communication of cyber observables and is intended to be flexible enough to give a solution to all cyber security use cases (ul-hassan Shirazi *et al.*, 2014)(MITRE Corporation, 2016a). A cyber observable is a stateful property and a cyber indicator is made up of patterns observables with relevant contextual information. It is intended to be used for detailed automatable sharing, detection, mapping and analysis heuristics (MITRE Corporation, 2016a). CybOX was created to actively encourage the consistent capture of cyber observables (MITRE Corporation, 2016a). It is also used in other XML based formats like STIX, TAXII, CAPEC and MAEC, which are also created by MITRE.

Figure 2.3[31] shows an example of a simple CybOX Observable representing a domain name. These CybOX Observables are similar to the OpenIOC `IndicatorItems`, in the

[29] http://www.mitre.org/
[30] https://www.oasis-open.org/
[31] https://github.com/CybOXProject/schemas/blob/master/samples/CybOX_Domain_Instance.xml

sense that one or many observables can make up a STIX indicator, which will be discussed in Section 2.5.4. Like OpenIOC the observables can be logically combined using AND or OR operators to create a STIX indicator. The CybOX language also supports regular expressions to define observable patterns (MITRE Corporation, 2016c). There are 88 different types of CybOX objects at the moment ranging from Address, Process, URI and Win Kernel.

Another feature of CybOX is that you can add relationships between properties, for example a file can be related to a domain name using the "Download_From" relationship, shown in Figure 2.4[32].

```
<?xml version="1.0" encoding="UTF-8"?>
<cybox:Observables xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cybox="http://cybox.mitre.org/cybox-2"
    xmlns:cyboxCommon="http://cybox.mitre.org/common-2"
    xmlns:URIObject="http://cybox.mitre.org/objects#URIObject-2"
    xmlns:example="http://example.com/"
    xsi:schemaLocation="
    http://cybox.mitre.org/cybox-2 ../cybox_core.xsd
    http://cybox.mitre.org/objects#URIObject-2 ../objects/URI_Object.xsd"
    cybox_major_version="2" cybox_minor_version="1" cybox_update_version="0">
    <cybox:Observable id="example:Observable-0b9af310-0d5a-4c44-bdd7-aea3d99f13b6">
        <cybox:Object id="example:Object-15be6630-b2df-4bf9-8750-3f45ca9e19cf">
            <cybox:Properties xsi:type="URIObject:URIObjectType" type="Domain Name">
                <URIObject:Value>example.com</URIObject:Value>
            </cybox:Properties>
        </cybox:Object>
    </cybox:Observable>
</cybox:Observables>
```

Figure 2.3: Example of a CybOX object

```
<cybox:Object id="example:object-e970c3df-0c01-4611-8257-6b01d188983c">
    <cybox:Properties xsi:type="FileObj:FileObjectType">
        <FileObj:File_Path>C:\temp\qwerty.dll</FileObj:File_Path>
    </cybox:Properties>
    <cybox:Related_Objects>
        <cybox:Related_Object id="example:object-d9f7b338-a20f-47b2-9efd-21f5a42b54f1">
            <cybox:Properties xsi:type="DomainNameObj:DomainNameObjectType">
                <DomainNameObj:Value>example.com</DomainNameObj:Value>
            </cybox:Properties>
            <cybox:Relationship xsi:type="cyboxVocabs:ObjectRelationshipVocab-1.1">Downloaded_From</cybox:Relationship>
        </cybox:Related_Object>
    </cybox:Related_Objects>
</cybox:Object>
```

Figure 2.4: Example of a CybOX relationship

2.5.4 STIX

The Standard Threat Information Expression (STIX) is used to describe information about cyber threats. The goal of STIX is to have a format that allows information to be easily stored, analysed and shared in a consistent manner (Barnum, 2014). It was developed by MITRE and was released in 2012, but has since been updated and improved.

[32]http://cyboxproject.github.io/documentation/object-relationships/

STIX not only allows for the creation of indicators but also other contextual information such as observables, incidents, exploit targets, courses of action, campaigns and threat actors. STIX can be used in the analysis of cyber threats, specifying indicator patterns, managing cyber threat response patterns and sharing cyber threat information (MITRE Corporation, 2016d). It has been adopted by many industry leaders and threat intelligence communities and according to Fransen *et al.* (2015), CybOX, STIX and TAXII (presented in Section 2.7) are the most promising standards for threat intelligence sharing.

Since STIX uses CybOX objects, it also uses an XML schema as shown in Figure 2.5. Each STIX indicator is made up of CybOX objects, and a set of STIX objects can be grouped by a STIX report wrapper (Barnum, 2014). A STIX report can contain a combination of observables, indicators, exploit targets and more. STIX can define the relationship between constructs due to its structured nature (Farnham and Leune, 2013).

```
<stix:STIX_Package
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:stix="http://stix.mitre.org/stix-1"
    xmlns:stixCommon="http://stix.mitre.org/common-1"
    xmlns:indicator="http://stix.mitre.org/Indicator-2"
    xmlns:cybox="http://cybox.mitre.org/cybox-2"
    xmlns:AddressObject="http://cybox.mitre.org/objects#AddressObject-2"
    xmlns:stixVocabs="http://stix.mitre.org/default_vocabularies-1"
    xmlns:example="http://example.com/"
    xsi:schemaLocation=" http://stix.mitre.org/stix-1 http://stix.mitre.org/XMLSchema/core/1.2/stix_core.xsd http://stix.mitre.org/Indicator-2
    http://stix.mitre.org/XMLSchema/indicator/2.2/indicator.xsd
    http://stix.mitre.org/TTP-2 http://stix.mitre.org/XMLSchema/ttp/1.2/ttp.xsd http://stix.mitre.org/default_vocabularies-1
    http://stix.mitre.org/XMLSchema/default_vocabularies/1.2.0/stix_default_vocabularies.xsd http://cybox.mitre.org/objects#AddressObject-2
    http://cybox.mitre.org/XMLSchema/objects/Address/2.1/Address_Object.xsd"    id="example:STIXPackage-33fe3b22-0201-47cf-85d0-97c02164528d" version="1.2">
    <stix:Indicators>
        <stix:Indicator xsi:type="indicator:IndicatorType" id="example:Indicator-33fe3b22-0201-47cf-85d0-97c02164528d" timestamp="2014-05-08T09:00:00.000000Z">
            <indicator:Title>IP Address for known C2 channel</indicator:Title>
            <indicator:Type xsi:type="stixVocabs:IndicatorTypeVocab-1.1">IP Watchlist</indicator:Type>
            <indicator:Observable id="example:Observable-1c798262-a4cd-434d-a958-884d6980c459">
                <cybox:Object id="example:Object-1980ce43-8e03-490b-863a-ea404d12242e">
                    <cybox:Properties xsi:type="AddressObject:AddressObjectType" category="ipv4-addr">
                        <AddressObject:Address_Value condition="Equals">10.0.0.0</AddressObject:Address_Value>
                    </cybox:Properties>
                </cybox:Object>
            </indicator:Observable>
        </stix:Indicator>
    </stix:Indicators>
</stix:STIX_Package>
```

Figure 2.5: Example of the STIX format

Out of all the formats described above, CybOX and STIX seem to be more widely accepted. STIX won the European Identity Conference (EIC) 2016 Award for Best Innovation/New Standard in Information Security (Struse, 2016). It has been adopted by industry leaders and has many options to make flexible and detailed IOCs. Their site also has easy to follow documentation[33] and every thing is well presented.

2.6 Automated IOC Generation after Traffic Analysis

This section focuses on approaches to the analysis process after a malware sample has been analysed. It will discuss a number of scripts and systems that generate IOCs automatically

[33]https://stixproject.github.io/documentation/

without human input. Most of them create IOCs from a report created by a automated dynamic analysis system.

In terms of the automated generation of IOCs for network traffic specifically, there is not a system that generates comprehensive detailed indicators. The most comprehensive range of network indicators is shown by MAEC, with four types of indicators. All of the discussed generators identify domain names and most of them identify IP addresses. Two incorporate full URLs and one creates whole HTTP and DNS response IOCs. A potential problem with domain names being IOCs is that some malware contains a Domain Generation Algorithm (DGA) so the malware randomly connects to a certain format of a domain and if the malware is only run once, only some of the potential malicious domains will be identified (Chismon and Ruks, 2015).

2.6.1 AutoMal

In a paper by West and Mohaisen (2014), malware was dynamically analysed using a sandbox called AutoMal and the corresponding PCAP files were processed to collect domain and subdomains from DNS look ups and full URLs from HTTP requests. These potential indicators were not automatically put into an IOC format, but were viewed and assessed by expert analysts. The analysts job was to find a potential indicator, evaluate if it is a threat or a non threat and determine the level of the threat.

In the cases where a parent DNS domain is considered benign, but is hosting a malicious file, the URL would be viewed as a threat and not the domain itself. This approach however is more labour intensive for analysts. A monthly WHOIS snapshot is also recorded that can give insight into who registered a malicious domain and if the domain changes hands, how long a domain has been registered and it was found that 40% of the domains classified as threats in this paper were less than one year old. The median domain age value was found to be 2.5 years.

Once a potential indicator is considered a treat, it can be put into an IOC format, which is not done in this study. This paper found an accurate way to classify potential indicators, but it requires non-trivial expert labour.

2.6.2 IOCAware

IOCAware[34] is a custom Cuckoo reporting module, written by Matt Jezorek and Dennis Kuntz to generate STIX or OpenIOC formatted IOCs from the logs generated by the Cuckoo Sandbox (Jezorek and Kuntz, 2013). It has not been updated since June 2015 and does not create a wide range of network indicators. IOCAware creates one network IOC containing an destination IP address. It uses the CybOX *NetworkConnection* object or the OpenIOC *PortItem* object for this.

2.6.3 IOC_Creator

IOC_Creator[35] is a python script found on Git hub, that generates OpenIOC formatted IOCs from unstructured data. It mainly focuses on OS related indicators, but has two network indicators including a OpenIOC *PortItem* with a single destination IP address and a OpenIOC *DnsEntryItem* consisting of a domain name.

2.6.4 MAEC

Malware Attribute and Enumeration Characterisation (MAEC) [36] is a structured language for characterising malware by their attributes. These attributes include the malware's behaviour, artefacts and attack patterns. It was developed by the CybOX developers and relies on the structure and content that CybOX provides. MAEC can express the context, indicators, behaviours and capabilities of malware after analysis (MITRE Corporation, 2014).

MAEC has tools for Anubis, TreatTrack[37] and ThreatExpert[38] and is built into the Cuckoo Sandbox as a reporting module. A MAEC data can be used to create host-based malware detection and can be translated into the Open Vulnerability and Assessment Language (OVAL)[39], which is for reporting upon the machine state of computer systems, or OpenIOC format. These can be used to create IDS, IPS and firewall rules (Hun-Ya Lock, 2013).

[34] https://github.com/iocaware/iocgen/blob/master/iocaware_stix/iocaware_stix.py
[35] https://github.com/tklane/openiocscripts/blob/master/ioc_creator.py
[36] https://maec.mitre.org/
[37] http://www.threattracksecurity.com/
[38] http://www.threatexpert.com/
[39] https://github.com/OVALProject/Language

MAEC creates four types of network Actions, a UDP connection that includes the source and destination IP address and port and the layer 4 protocol (UDP). A TCP connection that includes the same as the UDP connection but the layer 4 protocol is set to TCP. A DNS response object that includes the layer 7 and layer 4 protocols, the request, type and answers to for the DNS response. Lastly there is a HTTP request response object that houses the layer 7 and layer 4 protocols, the HTTP request method, the path being requested, the version, the user agent, host name, port and finally the HTTP response body. These four network Actions are descriptive and a lot more detailed than the IOCAware or IOC creator network objects.

2.7 Sharing Cyber Threat Information

In order to share cyber threat information, there needs to be at least one format in which they can be shared. The following few sections discuss the types of sharing available for cyber threat sharing, if it is manual or automated and what language is chosen for each system. There are also some methodologies that have not yet been implemented but have been suggested. Most of these technologies provide automated information sharing and speed.

It would be useful to have a standard format, so that IOCs can be easily shared without having to convert between formats. This would also allow for a greater distribution of IOCs and help security teams in tackling cyber threats. However in a recent survey, 300 information security professionals from the UK were asked where they obtain their threat intelligence and over half relied on their own detection process. Out of the 300 professionals, 43% would only share the threat intelligence internally and 40% would share it with a closed community of trusted peers. Organisations are scared of exposing sensitive company information, but there are many indicators like IP addresses, URLs and domain names that would not expose anything about a company (Kirk, 2015).

This section will discuss systems that are specifically designed to share threat information. These include TAXII, Threat Central, ThreatConnect, AlienVault Open Threat Exchange and the Malware Information Sharing Platform, which will be discussed below. There are a number of online platforms, mentioned in Section 2.2, where a sample of malware can be uploaded and a summary of the behaviour is displayed, which could be considered sharing platforms too because they sometimes display indicators.

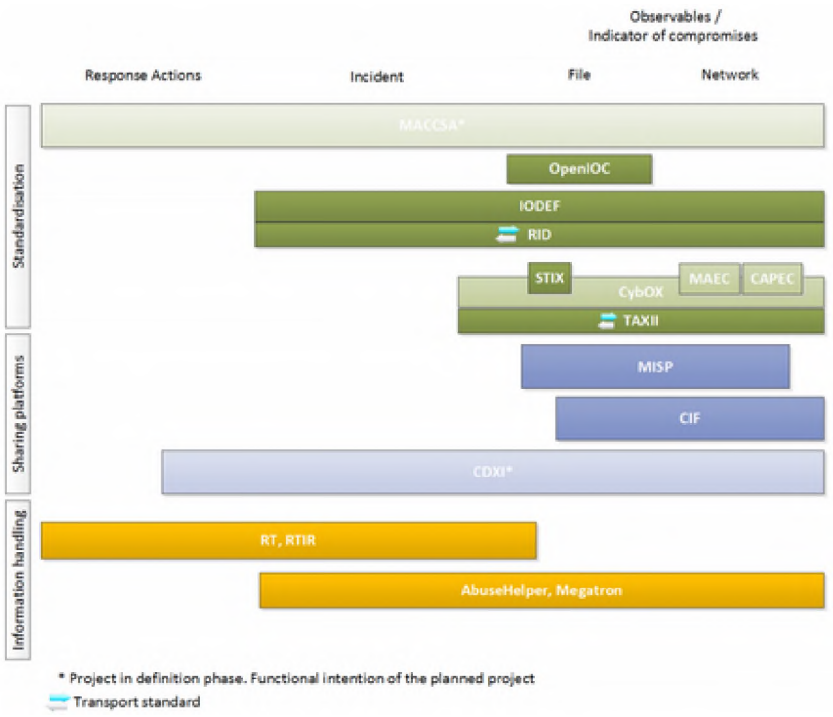

Figure 2.6: Standardisation Efforts for Sharing Cyber Threat Information (ENISA, 2013)

Figure 2.6 (ENISA, 2013) shows some data exchange formats, which were also discussed in Section 2.5. The information was gathered from a survey from 2013 (ENISA, 2013), and shows that network observables and indicators are put into the CybOX, MAEC and CAPEC formats. For sharing platforms, MISP, CIF (Collective Intelligence Framework) and the Cyber Security Data Exchange and Collaboration Infrastructure (CDXI) are shown.

2.7.1 TAXII

Trusted Automated Exchange of Indicator Information (TAXII) was created by MITRE in 2012 and defines services and message exchanges (Davidson and Schmidt, 2014). Other sharing systems typically require the manual input or tagging of information to generate IOCs, which is not scalable and would take a lot of time to generate many IOCs. One of the recent developments in the sharing of cyber threats is OASIS Cyber Threat Intelligence (CTI)[40]. The OASIS CTI Technical Committee, which includes the U.S Department of Homeland Security and other organisations have come together to develop standards to enable the analysis and sharing of threats and treat information. They are intending for

[40]https://www.oasis-open.org/

threat information to be shared among trusted partners and communities (Geyer, 2015). The work they are doing is to support automated information analysis and sharing to help with real-time network defence, threat awareness and response. The community has chosen CybOX, STIX and TAXII as their IOC format and sharing method.

The users of TAXII can choose what information they share and with who they share it with, which allows it to be flexible. TAXII supports different sharing models, which include hub and spoke, source/subscriber and peer to peer (Farnham and Leune, 2013). Hub and spoke is a sharing model where there is a main data hub and information is coordinated and shared through it. As shown in Figure 2.7 (MITRE Corporation, 2016b), there is a main hub that has many spokes attached. If one spoke wants to exchange information with another one, their communication is coordinated by the hub.

Also seen in Figure 2.7 is the source/subscriber model where there is a single source and the connected nodes get information from that one node. They are not able to contribute and share information with the main node, just receive information.

The last model shown in Figure 2.7 is the peer to peer model that allows two or more organisations to share information directly with each other. There is no coordination of information exchange so one company may end up having much less cyber threat information than another one (MITRE Corporation, 2016b).

In paper by Davidson and Schmidt (2014) on an overview of TAXII, the three main capabilities of TAXII were discussed, namely Push messaging, Pull messaging and Discovery. Push messaging is usually when data is pushed from a producer (like a source in the source/subscriber model) to a consumer (the subscribers), but it can also be when a consumer pushes information to another consumer if this has been agreed upon. Pull messaging is when a consumer requests information from a producer. The consumer would have to have permission to access the producers information, or the producer can make its information public. This can be used in the hub and spoke model and peer to peer. The discovery service that TAXII allows can find the exact TAXII services a TAXII user has in place, along with the IP address and supported binding. To be able to exchange information with another TAXII user, a sharing agreement must be put in place by humans (Davidson and Schmidt, 2014). A last minor capability that TAXII provides is for querying specific content that match a criteria. If a consumer only wants to see information on a certain type of threat then they can change their query information.

Sharing structured threat information can be considered automated from a customer's point of view, because they just get pushed information and don't manually have to pull

it. The entity pushing the information can either do it manually or automate it to push at a certain frequency. STIX provides a good framework and a useful set of capabilities.

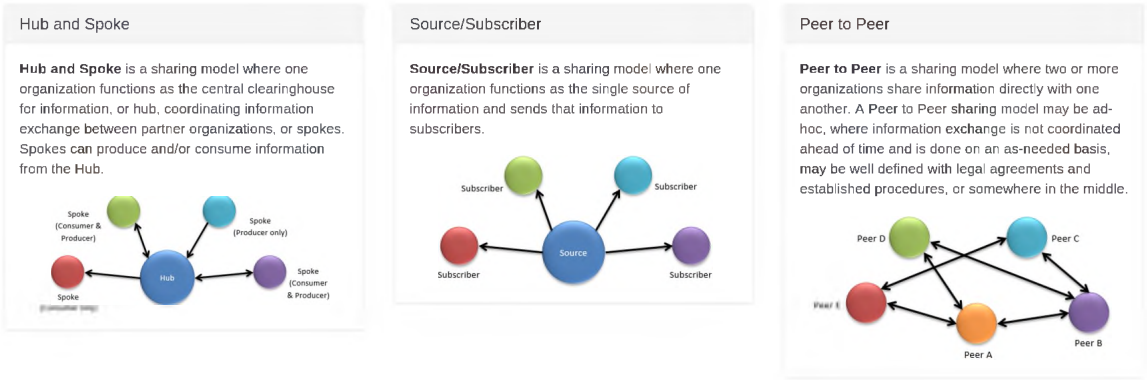

Figure 2.7: Sharing models supported by TAXII (MITRE Corporation, 2016b)

2.7.2 CDXI

CDXI stands for Cyber Security Data Exchange and Collaboration Infrastructure. The main goals are to facilitate sharing of information, enable automation and generate and refine the information through outsourcing. It is focused on structured cyber threat data and attempts to provide assurance of the data's origin and quality (Dandurand, 2013).

CDXI was developed by NATO[41] and aims to serve as a repository where members will automatically have information pushed and pulled using a variety of APIs (ENISA, 2013).

2.7.3 Collective Intelligence Framework

The Collective Intelligence Framework[42] (CIF) is for storing cyber threat information in a single repository. It was created by Research and Education Networking Information Sharing and Analysis Center (REN-ISAC)[43] and generates feeds of recent reports for different types of threats. CIF can output data in formats such as, STIX, JSON, CSV and Snort rules (ENISA, 2013).

Its goal is to collect information from multiple sources and have effective query mechanisms, correlation and sharing. It has a web interface and the information shared is

[41] http://www.nato.int/
[42] http://csirtgadgets.org/collective-intelligence-framework/
[43] http://www.ren-isac.net/

intended to be used for identification, detection and mitigation of threats. The most common types of information shared are IP addresses, domains and URLs (CSIRT Gadgets Foundation, 2015).

2.7.4 Threat Central

Threat Central is a framework created by Hewlett Packard (HP). It is a central place for the sharing of structured threat information and is community driven (Packard, 2016). It is STIX and TAXII compliant and has an API for automated sharing. It is targeted at enterprises that pass the qualification process. Threat Central allows for users to control what they share and who they share it with and also claims to provide relevant results by filtering out noise and false positives. It also provides mitigation steps by utilising other HP products like ArcSight ESM[44] (real-time event correlation and security analytics) and TippingPoint[45] (an IPS, firewall and more) (Packard, 2016).

2.7.5 ThreatConnect

ThreatConnect[46] is an intelligence sharing platform and has a free and paid for version. It has a web interface with a dashboard where the information can be viewed. According to the version, a user can choose to share their signatures and designate who can review and edit them. It can be used to share indicators and other threat information like malware signatures and IDS rules. Information can be uploaded to the system using other common data formats from tools such as like YARA, CybOX, OpenIOC, ClamAV, Snort, Suricata and Bro (ThreatConnect, 2013). The data from ThreatConnect can be pushed to other products using the API.

2.7.6 AlienVault Open Threat Exchange

There are other new solutions which allow for the uploading of IOCs in multiple formats, such as the AlienVault Open Threat Exchange (OTX)[47]. OTX is an online platform for sharing cyber threat information, from malware to fraud campaigns and more. It uses

[44] http://www8.hp.com/za/en/software-solutions/arcsight-esm-enterprise-security-management/
[45] http://www8.hp.com/us/en/software-solutions/network-security/
[46] https://www.threatconnect.com/
[47] https://otx.alienvault.com/

crowd sourcing to find out about new malware, malicious IPs, vulnerabilities and exploits (Kirk, 2015). According to Kirk (2015) many big security firms have not embraced the crowed sourcing method and like to keep their information in closed groups. However HP and Intel Security and others have partnered with AlienVault OTX (AlienVault, 2015).

OTX allows for the generation of threat information from the community, can handle collaborative research and provides automation of updating a company's threat data. The site also houses discussions on research in order to validate it. To share an IOC on AlienVault, a "pulse" must be created. A pulse is a summary of a threat with the software it targets and the corresponding IOCs. The kinds of IOCs supported are IP address, domain name, hostname (subdomain), email, URL, URI, file hashes, CIDR rules, file paths, MUTEX names and CVE number. After signing up for the OTX each user has a profile and other members can follow them, or they can follow other members and get notified when they create a pulse (AlienVault, 2016).

A pulse can be created manually or automatically using their API. Information on each indicator for the pulse can either be manually input or extracted from a source including a STIX report, OpenIOC file, blog post link, PDF threat reports or even text files. This makes AlienVault flexible as it takes many inputs. When viewing a pulse, the IOCs can be downloaded in four different formats: CSV, OpenIOC 1.0, OpenIOC 1.1 and STIX (AlienVault, 2016).

2.7.7 Malware Information Sharing Platform

Another solution is the Malware Information Sharing Platform (MISP)[48], which is a platform for sharing IOCs of targeted attacks. MISP can be joined by cyber defence and governmental related constituent of the NATO member nations (NCI Agency, 2013). The importing of data can be done manually by using a template or by uploading text, OpenIOC or the sandbox results of the Joe Sandbox or the GFI Sandbox. This can be automated by using the API, PyMISP[49] or ZeroMQ[50].

It allows for the export of data by generating Snort rules, Suricata rules, STIX, OpenIOC, text or CSV formatted data. MISP correlates relation between malware, events and attributes giving insight into probable advancements of a malware strain.

[48]http://www.misp-project.org/
[49]https://github.com/CIRCL/PyMISP
[50]http://zeromq.org/

2.7.8 IOC Bucket

IOC Bucket[51] is online sharing platform for OpenIOC formatted IOCs. It is not as established as AlienVault OTX or MISP. Manual upload of an IOC in OpenIOC format is supported. Many features are still in development, such as TAXII services and more. At the moment you do not need to sign up to the site to share IOCs and there is no API for automation at the moment.

2.8 Using Indicators for Network Defence

After network-based indicators have been generated and shared, they are only useful if they get used to create mitigation and detection mechanisms. This Section will discuss two types of network defence and mitigation systems, Intrusion Detection Systems (IDS) and firewalls. Section 2.8.1 will introduce IDSs and continue to discuss three types and their rules. Section 2.8.2 will introduce firewalls and three types of common firewalls and their rules.

2.8.1 Intrusion Detection Systems

Network Intrusion Detection Systems (NIDS) are a second line of defence against network-based attacks and have the task of detecting malicious activities on a network (Di Pietro and Mancini, 2008). These are systems that monitor network packets and are usually located on the network's border.

NIDSs are used in most large scale IT infrastructures (Di Pietro and Mancini, 2008), and attempt to identify suspicious actions such as Denial of Service (DoS) attacks, port scans and other policy violations on the network (Sykosch and Wubbeling, 2015).

This section will focus on signature based IDS systems, like Snort and Suricata and a hybrid IDS, Bro, which is both signature and anomaly based[52]. Signature based IDSs are reliant on pattern-matching algorithms and contain a database of signatures which are run against network traffic (Di Pietro and Mancini, 2008). Anomaly based IDSs attempt to judge the "normal" behaviour of a system and set off an anomaly alarm when a deviation from the usual behaviour exceeds a specific threshold (Garcia-Teodoro *et al.*, 2009).

[51] https://www.iocbucket.com/

[52] https://www.alienvault.com/blogs/security-essentials/open-source-intrusion-detection-tools-a-quick-overview

Snort

Snort is a real time open source lightweight network intrusion detection and prevention tool (Mehra, 2012). It runs over IP networks to perform content pattern matching on traffic in order to detect suspicious packets (Aydın *et al.*, 2009). It was developed as an open source tool in 1990 Sykosch and Wubbeling (2015) and is the most commonly used open source signature based IDS (Aydın *et al.*, 2009)(UpGuard, 2015). Snort is based on the `libpcap` library and was created to be rapidly deployed to fill holes on almost any node of a network (Roesch, 1999).

There are three features; `pass`, `log` and `alert`. `Pass` drops a packet, `log` writes a full packet to a log file (using tcpdump or plain text) and `alert` generates an event notification and logs the packet.

```
alert http $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"Cat detected";
                    content:"Cat"; nocase; sid:1; rev:1;)
```

Figure 2.8: Example of a Snort rule

A Snort rule has two main parts, the rule header and the rule options. The rule header has five sections: rules actions (`pass`, `log` or `alert`), protocol type (TCP, UDP, ICMP and IP), the end-to-end source and destination IP address and port (depending on the protocol) and the direction of traffic. Rule option consists of various conditions that help decide if a packet is suspicious or not. An example of a Snort rule is shown in Figure 2.8. This rule is to generate an alert when an HTTP packet containing the string "Cat" (not case sensitive because of the `nocase;`) is seen coming from `$HOME_NET` to `$EXTERNAL_NET`. `The $HOME_NET` and `$EXTERNAL_NET` are previously set IP addresses or subnets.

Suricata

Suricata is an open source, multi-threaded, high performance network IPS, IDS and network security monitoring engine. Suricata matches packets with signature matching and has a complete signature language (Suricata, 2015a).

It can automatically recognise the most common protocols of a packet. This makes it possible for rules to identify a packet's protocol, not by a port but using the protocol identifier, which allows flexibility, in case a protocol is seen using a different port. It can

also identify thousands of file types and tag them for extraction and logging (Suricata, 2015b), and supports IP reputation (Smith, 2015).

A Suricata rule has two main parts (similar to Snort) the rule header and the rule options. The rule header consists of an action, protocol, source and destination IP, source and destination port, and direction of traffic flow. For the protocol option there are a few choices, TCP, UDP, ICMP, HTTP, FTP, TLS, SMB, DNS and IP (this used as an all inclusive option). The rule options can contain many settings for pattern matching for different parts of a packet and what text to output when an alert is triggered. An example of a Suricata rule is shown in Figure 2.9 and is very similar to the Snort example rule in Figure 2.8, but the string being detected is "Dog".

```
alert http $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"Dog detected";
                    content:"Dog"; nocase; sid:1; rev:1;)
```

Figure 2.9: Example of a Suricata rule

In a study by Albin and Rowe (2012), it was found that Suricata can handle larger traffic volumes than Snort, however there was no significant speed or accuracy advantage of Suricata over Snort. In a more recent study, by Seok *et al.* (2016), comparing Snort 3.0 Alpha 2 and Suricata 2.0.11, Snort was found to have a higher performance rating.

Bro

Bro is an open source network analysis framework that does not rely on traditional signatures. Bro is targeted towards high performance networks and like Snort and Suricata provides logging of packets that have triggered an event (The Bro Project, 2014). Bro produces a number of different log files for protocols and files (The Bro Project, 2016a). The format of a log file can also be customised.

Like Snort and Suricata, Bro has a protocol analyser (The Bro Project, 2014) that can pick up a protocol without using a port identifier. It also has an extensive scripting language and provides an independent signature language for Snort style rules, but signatures are not Bro's preferred detection method (The Bro Project, 2016b).

A signature in Bro[53] has two attributes, the conditions and the actions. The conditions have to be met for an action to trigger. There are four groups of conditions, header,

[53]https://www.bro.org/sphinx/frameworks/signatures.html

content, dependency and context. Header conditions match packet headers, a content condition uses a regular expression against the raw payload, a dependency condition defines dependencies between signatures and a context condition passes a match decision onto other components of Bro (Mehra, 2012).

```
signature example-sig { ip-proto == tcp dst-port == 80 payload /.*mouse/ event "Mouse
                                    detected" }
```

Figure 2.10: Example of a Bro signature

Actions define what to do if a signature matches. The two actions are event and enable. Event raises an event, which is similar to an alert in Snort. The enable action enables the protocol analyser for the matching protocols. An example of a Bro signature is shown in Figure 2.10 with the signature labelled "example-sig". The signature will match packets using the TCP protocol with port 80, with the string "mouse" in the packet.

2.8.2 Firewall rules

A firewall, as defined by RFC 2647 (Newman, 1999), is a single device or multiple devices that put into effect an access control policy between networks. A firewall usually filters packets incoming or outgoing to a system or network. The goal is to prevent unauthorised network access to a computer, by blocking traffic that does not meet the rules set by the firewall. Indicators from network traffic, such as IP addresses and domain names, can be placed in firewall deny or log rules (Chismon and Ruks, 2015). Three types of firewalls will be discussed below.

IPFW

Also known as ipfirewall, IPFW[54] is a FreeBSD stateful firewall that filters both IPv4 and IPv6 packets (Antsilevich *et al.*, 2016). FreeBSD provide a sample rule set, but it is beneficial to add to this with custom rules and to keep up with the latest threats.

IPFW rules are evaluated in order from first to last and stopping when a rules criteria are met and the rules action is executed. IPFW had a strict rule syntax and keywords must be written in the order shown in Figure 2.11.

[54]https://www.freebsd.org/doc/handbook/firewalls-ipfw.html

```
CMD RULE_NUMBER set SET_NUMBER ACTION log LOG_AMOUNT PROTO from SRC SRC_PORT to DST
DST_PORT OPTIONS
```

Figure 2.11: IPFW rule syntax

There are mandatory keywords and optional ones. The lower case words (keywords) must come before the upper case ones (variables). Every new rule must start with the `CMD` being `ipfw add`. Every rule must also have a `RULE_NUMBER` and `SET_NUMBER` set. The `RULE_NUMBER` can range from 1 to 65534 and indicates the order of processing and multiple rules can have the same number. The `SET_NUMBER` can range from 0 to 31 and specifies what set the rules is in. Sets can be disabled or enabled, which makes the management of rules easier. One `action` must be added to every rule, which can be `allow` (or `accept`, `pass` and `permit`) or `deny` (or `drop`) for example. The `allow` action lets packets through that match this rule and the deny discards packets that match the rule. The `log` keyword is optional and if used, allows packets to be logged that match the specific rule. The `PROTO`, `SRC`, `SRC_PORT`, `DST` and `DST_PORT` are optional and specify the protocol, source address, source port, destination address and destination port. Finally the `OPTIONS` variable is optional and can be a large variety of things (Antsilevich *et al.*, 2016).

Iptables

Iptables[55] is the default installed firewall in official distributions of Ubuntu, Kubuntu and Xubuntu. It is a stateful firewall that maintains and inspects tables of IPv4 packet filter rules. There can be many different tables each containing a number of user defined and built in chains (a list of rules) (Purdy, 2004). For each packet that matches a rule a chosen action is performed. A 'target' is similar to the 'CMD' from IPFW, where an action is specified and is executed when a rule is matched. The targets in Iptables can be a special value like `ACCEPT`, `DROP`, `QUEUE` and `RETURN` or the name of a user defined chain, which will be jumped to if it is in the same table. As the names suggest, `ACCEPT` lets a packet through, `DROP` discards a packet, `QUEUE` passes a packet to a userspace process, `LOG` logs detailed information on a packet and `RETURN` means jump to the previous chain (that called this rule) and resume from the next rule.

When creating custom rules, there are a number of options that can be used. Some of the common ones are `-p` for protocol, `-s` for source address, `-d` for destination address and

[55] http://ipset.netfilter.org/Iptables.man.html

`-sport` and `-dport` for the source and destination port. Figure 2.12shows a simplified syntax for adding an Iptables rule. The `-A` is for appending the rule to a chain, `CHAIN` specifies the name of the chain to append the rules to (commonly 'INPUT' or 'OUTPUT' or a user defined chain), and `OPTIONS` can include a number of variable names and their values, like `-p` and `-s`. These variables have to be matched for a rule to be executed. The target is specified by `-j` and can have values such as `ACCEPT`, `LOG`, `DROP`, `REJECT` and more (Purdy, 2004).

```
Iptables -A CHAIN OPTIONS -j TARGET
```

Figure 2.12: Iptables syntax example

2.8.3 IOC Conversion to IDS or Firewall Rules

Most IDS rules are manually created by analysts. However, with the amount of malware that is being created everyday, the automated generation of IDS or firewall rules may be useful to help with mitigation of the malware and malicious traffic on the network. In a paper by Svajcer (2015), it was mentioned that trust may be an issue around automatically generated IDS rules that are created by a security threat sharing platform, like MISP. IOCs stored in MISP, CIF and ThreatConnect can be exported in different formats for IDS: Snort, Bro and Suricata rules. Although in MISPs case, it cannot be joined by just anyone, as stated in Section 2.7.7, so trust may not be a problem if the users sharing information are vetted first. Svajcer (2015) also suggests that IDS rules should be tested before they are applied, because usually IDS rules are created manually by experienced researchers.

In terms of conversion, Sykosch and Wubbeling (2015) created a mapping between CybOX objects and IDS signatures (Snort or Suricata). They also suggested that an IDS alert should reflect the ID number of a CybOX object. Mandiant (2012) suggests that it is easy to transform an IOC and feed it into an IDS or IPS.

2.9 Chapter Summary

This chapter showed a review of the available literature on a number of topics relating to malware and indicators of compromise. It started off with a section on malware analysis, including the difference between dynamic and static analysis techniques. It then discussed

the types of dynamic analysis systems such as TTAnalyse, CWSandbox, DRAKVUF and Cuckoo and the analysis of traffic after dynamic analysis.

Ransomware was explored next with a brief history, overview of how it works, its distribution around the world and the targets. Indicators of Compromise were discussed in detail starting with an overview and different formats they can be written in. Then the automated generation of IOCs after traffic analysis was explored with an overview of the types of sharing platforms for cyber threat information after that. Finally IDS and firewall rules were introduced and several common types were explored.

Chapter 3

System Setup and Configuration

The goal of the NetwIOC framework is to focus on the observables which can be seen on a network connection – particularly ICMP, TCP, UDP, HTTP, DNS, FTP, SSH connections. These IOCs will be created from a PCAP file containing network traffic from automated dynamic malware analysis. It was decided to use a sandbox that performs dynamic malware analysis and that runs locally on a system, instead of a web application. This is because a local instance can be customised and modules can be added when needed. The chosen sandbox environment is the Cuckoo Sandbox, and the setup will be discussed in Section 3.1. The setup of the virtual machines used by the Cuckoo Sandbox are discussed in Section 3.2. Lastly the design of the sharing platform will be explored in Section 3.3.

3.1 Cuckoo Sandbox Setup

A software framework was implemented on top of the Cuckoo Sandbox[1] (Oktavianto and Muhardianto, 2013) which was used to execute and perform a first pass analysis on malware samples. Cuckoo was chosen because it is written in Python, runs on a local system and is easily customised and extendable (Provataki and Katos, 2013). It takes a suspicious file as an input and performs real time dynamic malware analysis on it. It then executes a number of processing, signature and reporting modules that were previously enabled. A Cuckoo processing module is a Python script that analyses the raw data generated by the sandbox and appends information to the global container for use in the signature and reporting modules (Cuckoo Foundation, 2015b). A Reporting module

[1] https://cuckoosandbox.org/

takes the data from the global container and makes it accessible and consumable in many formats (Cuckoo Foundation, 2015b). These modules generate reports, screen shots and PCAP files from the analysis. The developed framework focuses on using the PCAP files to generate network related IOCs for each sample run on the sandbox.

The Cuckoo Sandbox 1.2 was downloaded and installed on a Debian 8 server. In order for the sandbox to function, it needs one or more VMs, a snapshot of each VM and a few variables in the configuration files changed to suit the setup.

Cuckoo was configured to allow the use of VirtualBox, as the virtual machine manager, and each sample was set to run for an allocated amount of time. Each Virtual machine was given Internet access through a bridged adaptor and the packet sniffer was enabled to capture packets from a specified ethernet adaptor

Cuckoo was also set up to use the NetwIOC framework, which includes custom processing and reporting modules to analyse traffic and automatically generate IOCs. These will later be discussed in detail in Sections 4.4 and 4.5.1.

The following Sections will give an overview of how five Cuckoo configuration files were set up to suit the system.

3.1.1 auxillary.conf

In order for Cuckoo to capture network traffic, a configuration file called `auxiliary.conf`[2] had to be set up to enable the packet sniffer, give the path to the local installation of the `tcpdump` utility and the name of the network adaptor to capture traffic from. The changes to the file from the default settings are shown in Listing 1 in Appendix A.

3.1.2 cuckoo.conf

The `cuckoo.conf`[3] file was changed to set the type of VM manager to VirtualBox, set the network routing to allow Internet access and set the physical adaptor that traffic is captured from. Also to set the result server to the local machine's address and finally set three time out values to allocated analysis times. These changes are shown in Listing 2 in Appendix A.

[2]https://github.com/cuckoosandbox/cuckoo/blob/master/conf/auxiliary.conf
[3]https://github.com/cuckoosandbox/cuckoo/blob/master/conf/cuckoo.conf

3.1.3 processing.conf

The `processing.conf`[4] file was set up to enable a new custom processing module (shown in Listing 3 in Appendix A), called `netwioc-proc`, that filters out non-malicious traffic. This processing module takes the PCAP captured by Cuckoo and filters out, to its best ability, non malware traffic and creates a new PCAP file for the custom reporting module to use, which will be discussed in detail in Section 4.4.

3.1.4 reporting.conf

The changes to the `reporting.conf`[5] file, is shown in Listing 4 in Appendix A. It was set up to enable a custom reporting module, called `netwioc-rep`, to create STIX indicators from the PCAP generated by the new processing module. The reporting module will be discussed in detail in Section 4.5.1.

3.1.5 virtualbox.conf

Since VirtualBox was used to manage the virtual machines, `virtualbox.conf`[6] had to be configured to enable Cuckoo to run the VMs in VirtualBox headless mode. Other settings include the IP address of the VM, name of the virtual machine together with its snapshot and the network adaptor name. All of these changes are seen in Listing 5 in Appendix A and as an example the virtual machine was called *Windows7USP1*.

3.2 Virtual Machine Setup

As previously mentioned, VirtualBox was chosen as the virtual machine manager for this system. Three operating systems were chosen: Windows XP Service Pack 2, Windows 7 Ultimate Service Pack 1 and Windows 8.1. Windows XP was chosen because it is known to be vulnerable, but it may not be too relevant these days as it is outdated and no longer supported by Microsoft[7]. Windows 7 and 8.1 were chosen because they mimic

[4]https://github.com/cuckoosandbox/cuckoo/blob/master/conf/processing.conf
[5]https://github.com/cuckoosandbox/cuckoo/blob/master/conf/reporting.conf
[6]https://github.com/cuckoosandbox/cuckoo/blob/master/conf/virtualbox.conf
[7]http://windows.microsoft.com/en-us/windows/end-support-help

systems seen more commonly these days. Windows 10 was not chosen, because it was not currently widely used at the time of conducting these experiments especially in large corporate environments.

The VMs were set up to have 2GB of RAM together with 20 GB of storage and few outdated versions of common programs were installed, shown in Table 3.1. These were chosen because they are everyday programs and the versions are outdated to ensure that the programs have as few security patches as possible. Microsoft Office was installed because Visual Basic macros are used to execute and install malware that later downloads ransomware (Microsoft, 2015).

To ensure minimal security, the firewall, Windows Defender, and Windows updates were turned off along with no installation of an antivirus. Microsoft office macros were set to be always enabled and any security settings found in the installed browsers were set to the lowest setting.

Since VirtualBox is being run on a server, VB Guest Additions had to be installed on the Windows machines in order to have shared folders. This allowed files from the server to be accessed by the VM, which helps when installing software. It was later uninstalled after its use for shared folders.

Table 3.1: Programs installed on the Virtual Machines

Windows XP	**Windows 7**	**Windows 8.1**
Python 2.7.4	Python 2.7.9	Python 2.7.9
Adobe Acrobat 6	Adobe Acrobat 9.4	Adobe Acrobat 10.1.4
Firefox 3	Firefox 3.6	Firefox 10.0
PIL1.1.7	PIL1.1.7	PIL1.1.7
Chrome 1.0.154	Chrome 4.154.25	Chrome 51.0.2704.106
Opera 7	Opera 10.15	Opera 12.00
Flash Player 6.0.21	Flash Player 9.0.115	Flash Player 11.7
Adobe Air 1.0.7.4880	Adobe Air 2.02	Adobe Air 2.02
iTunes 4.1	iTunes 9.0	iTunes 10.7
Foxit Reader 1.3.1413	Foxit Reader 3.0	Foxit Reader 5.4.2
Thunderbird 0.1	Thunderbird 3.1	Thunderbird 13.0.1
Java 6.27	Java 6.419	Java 7
Microsoft Office 2003	Microsoft Office 2007	Microsoft Office 2007

In order for Cuckoo to interact with a virtual machine, a Python script called `agent.pyw`

had to be added to the startup programs list of each VM.

3.2.1 Networking

A bridged adaptor was chosen for the virtual machines using the eth0 network adaptor of the host server. Cuckoo 2.0 allows internet access through a physical network adaptor but a file called `rooter.py` has to run before analysis starts to allow the access. Since Cuckoo requires the IP address of the VM, the network settings for each VM were manually in each VM, as dynamic IP addressing was not suitable.

3.3 Sharing Platform

The sharing platform was built in the form of a web application called IOC Xchange. It is a proof of concept platform and was developed to share STIX Packages with a focus on network indicators from malware. A number of frameworks were used in the design which include Bootstrap[8], as the HTML, CSS, and JS framework and Flask[9] as the Python Web Framework with SQLite for the database. Bootstrap was chosen because it has a comprehensive list of components, is highly flexible and uses a grid system for layouts. Flask was chosen because it is suited to smaller, simple web applications and it has numerous extensions that are easy to use. SQLite was chosen because of the nature of the web application and the fact that a production level database was not needed for the proof of concept task.

The Flask extensions that were used are Flask-SQLAlchemy[10], Flask-Login[11], Flask-WTF[12] and Flask-Security[13]. Flask-SQLAlchemy adds support to Flask for SQLAlchemy[14], which is a Python SQL tool kit and Object Relational Mapper that makes development easier and the database more secure. Flask-Login was used for user authentication and session management. Flask-WTF offers the integration of WTForms, which provides a flexible form validation and rendering library. Flask-Security allows for the addition of

[8]http://getbootstrap.com/
[9]http://flask.pocoo.org/
[10]http://flask-sqlalchemy.pocoo.org/2.1/
[11]https://flask-login.readthedocs.org/en/latest/
[12]https://flask-wtf.readthedocs.org/en/latest/
[13]https://pythonhosted.org/Flask-Security/
[14]http://www.sqlalchemy.org/

common security mechanisms, such as session based authentication and password encryption. Another useful library used was werkzeug[15], which has a function[16] for securing file names that are about to be uploaded to a system.

The hub and spoke architecture was chosen and for the web application. This is where the web application manages all the storage and information sharing of information from STIX Packages discussed in Section 2.5.4. The hub does not simply broker information but it can also perform additional processing by converting a STIX Package (full of STIX indicators) into certain formats of IDS or firewall rules.

The structure of the web application is discussed in Section 3.3.1 along with the API functionality.

3.3.1 Structure

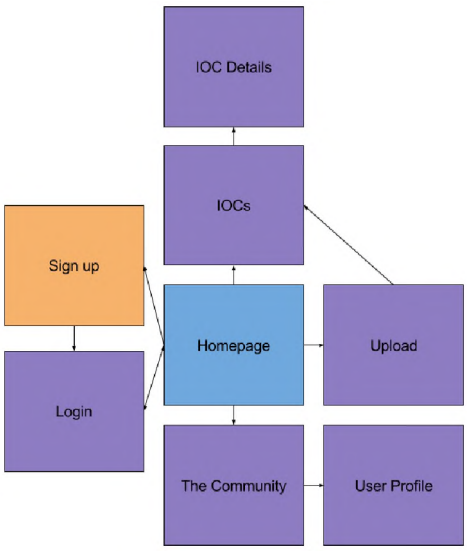

Figure 3.1: Structure of the Web Application

The simple web application needed several use cases, which included signing up, logging in, uploading, downloading, viewing a list of shared files, viewing a file's indicators, viewing site members and their profiles. Figure 3.1 shows the basic navigation of the pages and does not include pressing the back button or the navigation bar on top of the pages except for the homepage. A user is first greeted by the Homepage and can either Login or Sign

[15]http://werkzeug.pocoo.org/
[16]http://werkzeug.pocoo.org/docs/0.11/utils/#werkzeug.utils.secure_filename

up and they do not have access to the IOCs, Upload or Community page. Denying a user access to pages they are not authorised to visit is managed by the Flask-Security extension. A showcase of the sharing platform is performed in Chapter 6.

Sign up

When the Sign up option is selected and data is input into the username, email address and password fields, the username and email are checked for duplicates in the database. If there are no duplicates, the password is hashed with a SHA 512 algorithm using the hashlib Python library. When Sign up is successful and the data has been saved to the database, the Login page is displayed as shown in Figure 3.1.

Login

For a user to sign in they, must have a correct username and password combination. If either of these values are incorrect an error message is displayed. After a successful login the Homepage is displayed and the navigation bar at the top is populated with options to go to the list of uploaded files, view the Community or Upload a file.

Viewing indicators

When a the list of shared indicators is viewed (the IOCs page from Figure 3.1), information from the database is used to populate rows which are selectable and redirect to a page that displays metadata about the selected file and the indicators from the files itself (the IOC Detail page from Figure3.1).

IOC Detail page from Figure3.1 has four buttons for each type of way to download the indicators. These include the original STIX XML file or having the file converted to Snort rules, IPFW rules or Iptables rules.

Upload

When Upload is selected, a page in which the suspicious files name can be entered together with a description and the STIX IOC file itself. It is recommended that the files name is the SHA 256 hash of the malware sample to make each file name unique. For a file

to be uploaded, there must be no duplicates of it previously uploaded (by checking the file hash), it must be a .XML file and the file name is secured by using a function from the werkzeug Python library. Once a file is uploaded, it is checked to see if it contains a STIX package or not. If it is not, an error message is displayed and the file is deleted. If the file is validated a timestamp is taken, the malware name, description and timestamp are saved to a database, along with the uploader's username and file path.

Viewing site users

When the Users page is selected, a list of registered users is displayed, which are each selectable. Each user has a profile page which is simple, because it consists of the user's username and the list of files they have uploaded.

3.3.2 API functionality

The web application had two API functions, upload and download. API functions were added to make the upload and sharing scalable.

Upload

For the upload function, four values are needed: the path to the file being uploaded, the malware's name, the users API key and a description. HTTP POST methods are accepted and when the API is authenticated, and the file is only uploaded if it has the .XML extension and there are no duplicate files in the database. When the files has been uploaded the metadata is saved to the SQLite database and an upload successful message is sent as a reply to the HTTP POST request.

Download

For the download API function, only HTTP POST requests with the three values (malware name, API key and convert type) are accepted. The API key is authenticated and the malware name is validated. For each conversion type (Snort, IPFW, Iptables and None), the chosen file is input to a conversion script that parses the file and creates a new file with rules of the chosen type. If the conversion type is None, the original XML file is downloaded.

3.4 Chapter Summary

The section started off with an introduction to part of the NetwIOC framework, following with the setup of Cuckoo and the Virtual Machines used for analysis. This included the values set in the Cuckoo configuration files and the programs installed on the VMs, and how the networking was set up. Details of the proof of concept sharing platform's functionalities and design were given in Section 3.3. A showcase of the sharing platform is performed in Chapter 6.

Chapter 4

IOC Generation

There are two leading formats for the representation of Indicators of Compromise. Section 4.1 explores the similarities between the CybOX[1] and OpenIOC[2] formats, previously mentioned in Sections 2.5.1 and 2.5.3. The reasons for choosing CybOX and STIX, and the objects used in the framework are discussed in Section 4.2. This section will also go into the how the IOCs were automatically generated, starting with a system overview, and ending with a detailed discussion on how the system gathers information to generate indicators. The creation of each type of indicator is described in Section 4.5 and the creation of IDS and firewall rules from STIX indicators is explored in Section 4.6.

4.1 CybOX vs OpenIOC Network Objects

Ever since the team that developed MAEC were not satisfied with using the OpenIOC format, they developed CybOX (Obrst *et al.*, 2012). There are a few similar network objects which will be compared below. CybOX has many more network related objects compared to OpenIOC and they are multi-layer and very detailed. The following sections will compare and map the network objects that OpenIOC and CybOX have in common.

4.1.1 ARP

Both OpenIOC and CybOX have one object dedicated to the Address Resolution Protocol (ARP). They are very similar in terms of fields which are shown in Table 4.1. CybOX

[1] https://cyboxproject.github.io/
[2] http://openioc.org/

only has one extra field, which makes these objects almost identical.

Table 4.1: OpenIOC ArpEntryItem vs. CybOX ARPCacheObj

OpenIOC ArpEntryItem	CybOX ARPCacheObj
IPv4Address	ARPCacheEntryType/IP_Address
PhysicalAddress	ARPCacheEntryType/Physical_Address
CacheType	ARPCacheEntryType/Type
Interface	ARPCacheEntryType/Network_Interface
	ARPCacheEntryTypeType/datatype

4.1.2 DNS

CybOX has three DNS related objects – *DNSCacheObj*, *DNSQueryObj* and *DNSRecordObj*. OpenIOC has a *DNSEntryItem*, which is equivalent to the CybOX *DNSRecordObj*. OpenIOC does not have objects similar to the *DNSCacheObj* or *DNSQueryObj*, but the comparison between the *DNSEntryItem* and CybOX *DNSRecordObj* are shown in Table 4.2, with the CybOX having four more fields than the OpenIOC object.

Table 4.2: OpenIOC DNSEntryItem vs. CybOX DNSRecordObj

OpenIOC DNSEntryItem	CybOX DNSRecordObj
DataLength	Data_Length
Flags	Flags
Host	Domain_Name
RecordData/Host	Record_Data
RecordData/IPv4Address	IP_Address
RecordName	Record_Name
RecordType	Record_Type
TimeToLive	TTL
	Description
	Query_Date
	Address_Class
	Entry_Type

4.1.3 Email

The OpenIOC *EmailItem* object and the CybOX *EmailMessageObj* have 15 fields in common, but a lot more fields that are not in common. As shown in Table 4.3, the formats have 15 common fields. CybOX has 25 fields that OpenIOC does not have, which are shown in the extended version of Table 4.3, in Table B.1 in Appendix B. OpenIOC has 10 fields that do not have CybOX equivalents. The fields they have in common are basic email properties like To, From, Subject, Body, CC, basic information on an attachment, date and times and more. The CybOX object is multi layered, which allows parts of the object to be divided up into related groups. These groups include *AttachmentsType*, *EmailHeaderType* and *EmailReceivedLineType*.

Table 4.3: OpenIOC Email vs. CybOX EmailMessageObj

OpenIOC Email	CybOX EmailMessageObj
Attachment/Name	Attachments/AttachmentsType/FAttachmentReferenceType->FileObjectType/File_Name
Attachment/SizeInBytes	Attachments/AttachmentsTypeAttachmentReferenceType->FileObjectType/Size_In_Bytes
BCC	Header/EmailHeaderType/BCC
Body	Raw_Body
CC	Header/EmailHeaderType/CC
Content-Type	Header/EmailHeaderType/Content_Type
Date	Header/EmailHeaderType/Date
From	Header/EmailHeaderType/From
In-Reply-To	Header/EmailHeaderType/In_Reply_To
MIME-Version	Header/EmailHeaderType/MIME_Version
Received	EmailReceivedLineType/Timestamp
ReceivedFromHost	EmailReceivedLineType/From
ReceivedFromIP	EmailReceivedLineType/From
Subject	Header/EmailHeaderType/Subject
To	Header/EmailHeaderType/To

4.1.4 Network Route Entry

In terms of route entries, the OpenIOC *RouteEntryItem* object is equivalent to the *NetworkRouteEntryObj* in CybOX. The OpenIOC object has nine fields, shown in Table 4.4, and every one of them can be mapped to the CybOX object, but the CybOX object does have seven extra fields, four of which serve as flags.

Table 4.4: OpenIOC RouteEntryItem vs. CybOX NetworkRouteEntryObj

OpenIOC RouteEntryItem	CybOX NetworkRouteEntryObj
Destination	Destination_Address
Gateway	Gateway_Address
Interface	Interface
IsIPv6	is_ipv6
Metric	Metric
Netmask	Netmask
Protocol	Protocol
RouteAge	Route_Age
RouteType	RouteType/datatype
	Origin
	Preferred_Lifetime
	Valid_Lifetime
	is_autoconfigure_address
	is_immortal
	is_loopback
	is_publish

4.1.5 Network Ports

As shown in Table 4.5, the OpenIOC *PortItem* object and CybOX *SocketAddressObj* are similar with the CybOX object having three fields in common. The OpenIOC object describes a network port with details on the process created it and when. The CybOX object does not contain information on the process that created a port or socket and when. This is the only network related OpenIOC object that gives more detail than the CybOX object.

Table 4.5: OpenIOC PortItem vs. CybOX PortObject

OpenIOC PortItem	CybOX SocketAddressObj
localPort, remotePort	PortObject/Port_Value
protocol	PortObject/Layer4_Protocol
remoteIP	IP_address
path	
process	
pid	
state	
creationTime	
localIP	
	Hostname

4.1.6 Summary

So far every CybOX object except one, the Network Port object, has been more comprehensive in describing the given objects. This does make sense since CybOX took inspiration from OpenIOC when it was first developed.

There are a number of CybOX network related objects that do not have a similar OpenIOC objects, which are shown in Table 4.6. These include a highly detailed HTTP Session, a number of DNS objects, Network Connection, Flow, Packet, Route, Socket, and Subnet and more. The only network related OpenIOC object that does not have a CybOX equivalent is the Network Object.

Table 4.6: CybOX network related objects that do not have a similar OpenIOC objects

CybOX object type
HTTP Session
Address
AS
DNS Cache
DNS Query
Domain Name
Host Name
Network Connection
Network Flow
Network Packet
Network Route
Network Socket
Network Subnet
Socket Address
Unix Network Route Entry
User Session
URI
Whois

4.2 IOC Generator

This section discusses how the information extracted from the Cuckoo analysis can be used to create STIX indicators. STIX is a promising IOC format, as it provides the detailed structure needed by NetwIOC and has been adopted by many industry leaders (Fransen *et al.*, 2015).

4.2.1 Overview

Nine primary types of Indicators (which are shown later in Table 4.7 in Section 4.3) where chosen for the system. The main protocols focused on are TCP, UDP, ICMP, DNS, HTTP, SSH and FTP. SHH and FTP were chosen to get a more comprehensive view of

the behaviour of a sample, as some samples upload files to FTP servers and use SSH. In order to start getting the information needed to create each of the individual objects, a malware sample had to be submitted to the Cuckoo Sandbox.

Table 4.7: Types of STIX indicators chosen

Indicator Type	CybOX Objects
IP address string found in binary	Address
IP address resolved from domain	Address
ICMP (echo or echo reply)	Network Connection, Socket Address
TCP (connection failed or established)	Network Connection, Port, Socket Address
UDP connection	Network Connection, Port and Socket Address
HTTP request (GET/POST)	HTTP Session, URI, Port
DNS request	Network Connection, Port, Socket Address, DNS Query
FTP connection (codes: 220, 230, 250)	Network Connection, Port, Socket Address
SSH connection	Network Connection, Port, Socket Address

Figure 4.1 shows the flow of the IOC generator, with IOCs (in a STIX Package format) as the final product, and the generation of IDS or firewall rules as an optional final product. As seen in Figure 4.1 and stated previously in Section 3.1, a malware sample is given to the Cuckoo Sandbox that dynamically analyses the sample and records the network traffic. Once a sample is finished execution, Cuckoo starts running the enabled processing modules. A custom processing module was created, that will be discussed in Section 4.4, that filters out baseline traffic to leave a PCAP of malicious traffic.

After Cuckoo has run its configured processing modules, it starts executing reporting modules. Cuckoo had a default reporting module that does generate some network information. This includes the IP addresses the VM contacted, domain names queried and HTTP post requests, but since the reporting module uses the unfiltered PCAP, the values are not guaranteed to be from a malicious source. Therefore a custom reporting module

was created that uses the filtered PCAP and extracts data that is needed for the creation of the chosen STIX indicators. The reporting module will be discussed in Section 4.5.

By using the filtered PCAP the system had access to the entire packets, not just a single piece of information, such as an IP address, found in the Cuckoo default report. This allows for the creation of more detailed IOCs with the support for a larger variety of protocols.

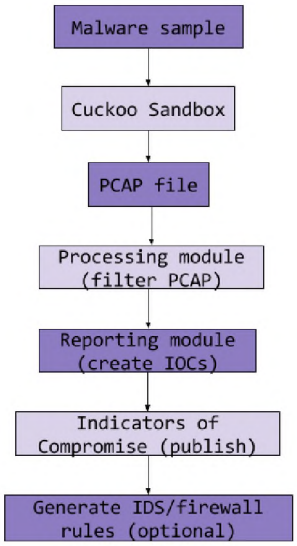

Figure 4.1: System flow diagram

4.2.2 CybOX

CybOX 2.1 was chosen to be the format to represent cyber observables. CybOX is designed for capture, specification, characterisation and communication of cyber observables. It is the underlying format for STIX, as CybOX Observables are placed individually or grouped into a STIX object wrapper, which is presented in Section 4.2.4. As demonstrated in Section 4.1, the CybOX language is a lot more comprehensive than OpenIOC which lead to it being chosen for this system. Another reason for this choice is that CybOX and STIX are stable and were chosen by the OASIS CTI team to be one of standard the formats for the future of cyber threat sharing (Geyer, 2015).

4.2.3 STIX

STIX 1.2 was chosen as the format of choice because is able to represent network level IOCs with the level of detail needed by the system, both simple and complex. It incorporates

the use of CybOX objects in the creation of STIX objects. Since STIX and CybOX are XML based, the data stored by a STIX object can be exchanged easily across software applications, different system platforms and simply rendered using style sheets (Casey *et al.*, 2015). The level of detail of STIX objects can be very flexible and can vary from a single property object, to multiple properties in one object and even to logical (AND/OR) combination of objects (MITRE Corporation, 2015). It was decided to create a custom reporting module that uses the filtered PCAP to generate network based STIX IOCs.

4.2.4 CybOX and STIX Relationship

The relationship between CybOX objects, STIX indicators, Reports and Packages are shown in Figure 4.2. In order to create a STIX indicator, one or more CybOX Observables are wrapped in a STIX indicator wrapper, alone or logically combined using AND or OR.

Related STIX indicators can be grouped using a STIX Report[3] which is a wrapper to give context to related Indicators. As shown in Figure 4.2, the Report was used to wrap Indicator ID s and not the actual Indicators themselves. This allows for the same Indicators to appear in multiple Reports, by easily including the ID.

Finally the Indicators and the Report are wrapped in a STIX Package wrapper. A Package can include multiple Reports and multiple Indicators and is used to group related or unrelated content. Details on each CybOX and STIX objects used can be found in Section 4.3.

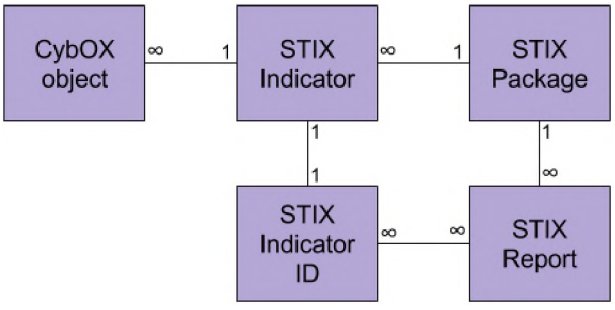

Figure 4.2: Relationship between CybOX and STIX

[3]http://stixproject.github.io/data-model/1.2/report/ReportType/

4.3 CybOX and STIX Objects Used

Nine primary classes of indicators were chosen to be represented in the form of a STIX indicator and they are described in detail in Section 4.5. This section will address the different types of CybOX and STIX objects selected to represent the types of indicators. These nine types are shown in Table 4.7, together with the corresponding CybOX objects used to represent them. The STIX objects and Indicator, Report and Package are described in Section 4.3.2.

4.3.1 CybOX

The following Section gives an overview of the specific CybOX objects used in the system and what their relationship is to each other, if any. The seven types of CybOX objects described below can be seen in the second column of Table 4.7, where a combination of them make up each of the nine indicator types.

Address

CybOX has an object called `Address`[4] that has a property called *Address Value* that can be used to store addresses of the types: ASN, ATM, CIDR, E-mail, Mac, IPv4 address, IPv4 network, IPv4 net mask, IPv6 address, IPv6 network and IPv6 netmask. This object was chosen to represent an IPv4 address for the framework.

URI

The `URI` object[5] can describe two properties, the *Value* and the *Type*. The *Value* field would house the URI being described and the type field can be specified as a URL, general URN or a domain name.

[4]http://cybox.mitre.org/XMLSchema/objects/Address/2.1/Address_Object.xsd
[5]http://cybox.mitre.org/XMLSchema/objects/URI/2.1/URI_Object.xsd

Port

The CybOX `Port` object[6] is simple and has two properties, namely the *Port Value* and the *Layer4 Protocol*. It is used in the Socket Address object to represent the port part of a socket.

Socket Address

The `Socket Address` object[7] can show three properties, which are an *IP address*, *Hostname* and *Port*. To represent a port, the `Socket Address` uses the `Port` object. The `Socket Address` object is used as part of the `Network Connection` object to represent a socket.

DNS Query

The `DNS Query` object[8] can represent many different and detailed properties using other objects such as the `Port` and `Socket Address`. The main properties include the *Transaction ID*, *Question*, *Answer Resource Records*, *Authority Resource Records*, *Additional Records*, *Date Ran* (date the query was run), *Services Used* and *Successful* (boolean specifying if the query was successful or not). This object is used in the `Network Connection` object to specify a Layer 7 Connection.

HTTP Session

A CybOX `HTTP Session` object[9] makes use of the `Address`, `Port` and `URI` field along with a number of properties that are not represented with objects, such as HTTP header fields. This object has many layers to it starting with the outer layer, an *HTTP Request Response*, which consists of a *HTTP Client Request* and a *HTTP Server Response*.

The *HTTP Client Request* is made up of a *HTTP Request Line*, *HTTP Request Header* and a *HTTP Message Body*. The *HTTP Server Response* consists of a *HTTP Status Line*, *HTTP Response Header* and a *HTTP Message Body*. Each of the mentioned objects consist of a wide range of detailed properties. The properties used for the system will be discussed in Section 4.5.

[6] http://cybox.mitre.org/XMLSchema/objects/Port/2.1/Port_Object.xsd
[7] http://cybox.mitre.org/XMLSchema/objects/Socket_Address/1.1/Socket_Address_Object.xsd
[8] http://cybox.mitre.org/XMLSchema/objects/DNS_Query/2.1/DNS_Query_Object.xsd
[9] http://cybox.mitre.org/XMLSchema/objects/HTTP_Session/2.1/HTTP_Session_Object.xsd

Network Connection

The CybOX `Network Connection` object[10] can house a large number of information and makes use of other CybOX objects. The main properties include *Connection Time*, *Layer 3 Protocol*, *Layer 4 Protocol*, *Layer 7 Protocol*, *Source Socket Address* (uses the `Socket Address` object above), *Source TCP State*, *Destination Socket Address* (uses the `Socket Address` object above), *Destination TCP State*, *Layer 7 Connections* (can take a `DNS Query` or `HTTP Session` object), *TLS Used*. This object was used to represent the ICMP, UDP, TCP, SSH and FTP connections.

4.3.2 STIX

This Section will present the three STIX objects that were introduced in Section 4.2.4. All three are wrappers with metadata fields and they are important because they give context to a bunch of CybOX Observables.

Indicator

The STIX `Indicator` object[11] was chosen to represent each IOC in this system. An `indicator` object is intended to represent artefacts and/or behavioural conditions together with contextual information. It is suggested by the creators of STIX that an indicator should include a selected number of metadata fields, namely *Title*, *Type*, a confidence assertion and a time window for which the indicator is valid. There are many more metadata fields ranging from *Description*, to *Test Mechanisms*, to *Related Indicators* and more.

Report

The `Report` object[12] is intended to be used as a contextual wrapper for a group of STIX objects that are related in some way. A Report is suggested to include a number of fields, the *Title*, *Intent* and *Information Source* as a minimum. For the system, a `Report`

[10] http://cybox.mitre.org/XMLSchema/objects/Network_Connection/2.1/Network_Connection_Object.xsd

[11] http://stixproject.github.io/data-model/1.2/indicator/IndicatorType/

[12] http://stixproject.github.io/data-model/1.2/report/ReportType/

could be used to show that a number of `Indicators` are related (like IOCs for a single malware sample) and provide metadata about them. Other fields that can be used to add metadata are *Description* and *Short Description*. Objects, other than `Indicator`, that can be grouped are `Observables`, `Incidents`, `Courses` of `Action`, `Campaigns`, `Threat Actors`, `Target Exploits` and more. A `Report` can group the actual objects together or group the ID numbers of the objects, where the actual objects are housed in the `Package`.

Package

The STIX `Package`[13] is the outer most wrapper and encases all the above mentioned objects that may or may not be related in some way. It is used to define a group of STIX objects and should include a `Report` object, for grouping related objects and adding contextual information. Like a `Report` it can include STIX objects such as `Observables`, `Indicators`, `Exploit Targets`, `Incidents`, `Courses of Actions`, `Campaigns` and more.

4.4 Filtering out Clean Traffic

A Python script was developed and used to extract and analyse information generated by Cuckoo. First the script was created to filter out (as best as possible) non malware related traffic from each PCAP file. This script makes use of a `tshark` filter, which was created after analysis. A non-malicous file `unicorn.png` was first submitted to the Cuckoo Sandbox five times to observe the network traffic created by the VM as a baseline. By using the baseline PCAP files, all of the IP addresses that the VM communicated with were extracted (excluding the pre-configured DNS server and the VM IP address).

Next the DNS queries and responses were extracted, which allowed us to create a list of clean domain names and the resolved IP addresses (if the domain was resolved). These domains include `microsoft.com`, `google.com`, `bing.com`, `windowsupdate.com`, `apple.com` and `sun.com`. These domains of course depend on the specific OS and program versions installed on the VM and should be periodically updated.

The IPs and domains were added to a `tshark`[14] filter that reads the Cuckoo generated PCAP and creates a new PCAP with packets of the type TCP, UDP, DNS, HTTP, ICMP and SMTP. The filter also filters out packets to or from the clean IP addresses and clean

[13] http://stixproject.github.io/data-model/1.2/stix/STIXType/
[14] https://www.wireshark.org/docs/man-pages/tshark.html

DNS requests and responses for domains from the baseline test. Because of dynamic IP addresses these two filter lists would have to be regenerated, before running a number of samples through Cuckoo.

The `tshark` filter was saved to a bash script so it could be utilised by a custom Cuckoo processing module that was developed as part of NetwIOC. It is suggested that the filter creating script be run periodically to refresh the `tshark` filter. A custom processing module calls the `tshark` filter script to create a filtered PCAP after Cuckoo has executed the sample and has generated the unfiltered PCAP. This custom processing module proved to work well for the system and managed to filter out most unwanted baseline traffic. However, some samples would very rarely be found contacting local university IP addresses, such as IT management servers and printers, so these IPs were added manually to the clean filter list.

The next step was to work with these filtered PCAPs assuming they only contain potentially malicious traffic resulting from a sample run and gather information that may be useful in the creation of IOCs. After all the enabled Cuckoo processing modules are finished executing, the reporting modules are run. The custom reporting module was created for the purpose of retrieving information from the filtered PCAP and using it for the creation of STIX indicators.

Secondary filtering had to be implemented in the reporting module because of dynamic IP addressing where some Microsoft domains would resolve to a different address as found in the baseline PCAP files largely due to the use of Content Distribution Networks (CDN). The reporting module will be discussed below in Section 4.5.

4.5 IOC Generation

This section focuses on how the CybOX and STIX objects were used and how the network based Indicators were generated from a malware sample. The custom reporting module, that is run after all the processing modules are finished, is discussed. This reporting module uses the python-cybox[15] and python-stix[16] library to create indicators and save them to an XML file.

[15]https://github.com/CybOXProject/python-cybox
[16]https://github.com/STIXProject/python-stix

4.5.1 Reporting module

The reporting module was written in Python to parse a previously filtered PCAP, looking for specific packet types and properties. It extracts the information and uses it in the creation of the different STIX indicators. Table 4.8 shows the nine types of indicators that were created and the individual network properties that were used to create them. The creation of each of the objects will be discussed in the following sections.

Table 4.8: The nine indicators created and the individual properties needed

Indicator Type	Network Properties
IP address string found in binary	IPv4 address
IP address resolved from domain	IPv4 address
ICMP (echo or echo reply)	IPv4 address, Type
TCP (connection failed or established)	IPv4 address, Port, TCP state
UDP connection	IPv4 address, Port
HTTP request (GET/POST)	Method, URI, Version, Host, Port, Accept, Accept language, Accept encoding, Authorization, Cache control, Connection, Cookie, Content length, Content type, Date, Proxy authorization
DNS request	IPv4 address, Port, Domain name, Type
FTP connection (codes: 220, 230, 250)	IPv4 address, Port, Request argument, Response argument
SSH connection	IPv4 address, Port, Public key

Although the processing module was created to filter out clean traffic, it was found that implementing secondary filtering in the reporting was useful and made the system more accurate. Instead of filtering that PCAP itself, any domain, particularly sub-domains, about to be added to a CybOX object would be checked against a list of clean domains, just in case the VM queried different clean domains to the baseline. In this case the end of the domain was checked for the strings in Table 4.9, which are relevant to the particular VM that was used. If the domain was found to be non malicious, an indicator was not created.

Table 4.9: Clean domains from the baseline tests

Domain
microsoft.com
windows.com
windowsupdate.com
trafficmanager.net
msocsp.com
gvt1.com
verisign.com

These domains however solely depend on the operating system and programs installed on the virtual machine. The domains in the table either relate to Microsoft, Google or certificates. These second level domains were extracted from the baseline filter full domains and are being used to improve the accuracy of the IOCs being created. For example, the first pass filter would filter full domains like "watson.microsoft.com" and "teredo.ipv6.microsoft.com", that were extracted directly from the baseline network traffic from the virtual machine. If the VM does generate clean traffic different from the baseline PCAP, the listed domains will filter them out.

Another part of the secondary filtering involved taking the clean domains that slipped through the first pass filter and making a list of all the IP addresses they resolved to. Every time an IP address was to be added to a CybOX object, the IP address would be check against the clean IP list. If the IP was found to be clean, the object was not created.

The next nine sections detail the types of indicators from Table 4.8 created by the reporting module and how they were generated.

4.5.2 IP address string found in binary

There is a Cuckoo processing module called *Strings*, that reads in a file and extracts any strings it finds. The list of strings is appended to the Cuckoo global container so that they can be reached by any reporting module later on.

The reporting module that was created makes use of this list of strings and uses a regular expression to check for IPv4 addresses. Once an IP address had been found, a method is

called that starts by creating a CybOX Address object and adding the IP address to the `address_value` field of the Address. The Addresses category is set to IPv4, but can be extended to search for IPv6 addresses in the future.

A STIX indicator object is created and the `title`, `description` and `set_time_produced` metadata fields are populated accordingly. The CybOX Address is then added to the STIX indicator. Finally the Indicator ID number is added to a STIX report and the Indicator and Report are added to a STIX Package object, that was created at the beginning of the module.

4.5.3 IP address resolved from a Domain Query

For each malicious domain name request, the resolved IP addresses were retrieved. These IP addresses are different from the above ones, because they are extracted from the PCAP its self and not the static malware sample.

To start off, `tshark` was used to retrieve two DNS query properties: the queried domain and the answer to the query. The `tshark` filter used is shown in Listing 7 in Appendix C, and it can be seen that `-Y dns.flags.response==1` ensured that only DNS response packets were utilised. This filter saves the domain names and answers found to a CSV file.

Next, line by line, the CSV file is iterated through (ignoring duplicates). For each domain and IP address, the domain is checked against the clean domain list, mentioned in Section 4.5.1. If the domain is found to be suspicious, a CybOX Address object is created and the DNS query answer (IP address) is added to the `address_value` field. The Addresses category field is then set to IPv4 and a STIX indicator object is created with necessary metadata, and the CybOX Address is added to it.

Finally the Indicator ID number is added to the STIX report and the Indicator and Report are added to the same STIX Package object as the rest of the Indicators.

It is noted that these resolved IP addresses may not stay relevant for long as they may later be associated with a different domain. In this case the STIX indicator object has a property called `Valid_Time_Position`, that is used to specify a time window in which the indicator is valid.

4.5.4 ICMP (echo or echo reply)

Instead of creating an indicator out of every ICMP packet seen, the proof of concept system only looks at ICMP echo and echo reply packets (incoming or out going). For an echo, the ICMP `type` field is set to 8 and for a echo reply the `type` field is set to 0.

The `tshark` filter that was used in shown in Listing 8 in Appendix B. As shown in the listing, `-Y icmp`, was used so that only ICMP packets are analysed. Three values were extracted from the ICMP packets - type, source IP address and destination IP address, and saved into a CSV file.

This CSV file, that contained three values on each row, was iterated through (ignoring duplicate rows). The source and destination IP addresses were tested against the clean IP list and if they were found to be clean, the row would be skipped. Rows were chosen for indicator creation if they were "malicious" and the ICMP type was 0 or 8. For each selected row, it was decided if the ICMP packet was ingoing or outgoing. If a packet was found to be incoming, only the source IP and ICMP type would be used for the indicator. On the other hand, if a packet is outgoing, the destination IP and ICMP type would be used.

A CybOX Network Connection and Socket Address's objects are created and the source or destination IP address is added to the Socket Address `ip_address` field. The Network Connection's `layer3_protocol` is set to ICMP. A STIX indicator is created and the metadata fields are set to describe if the packet was an echo or echo reply. Then the Socket Address is added to the Network Connection, which is then added to a STIX indicator. The object's ID is added to the Report and the object itself is added to the Package.

4.5.5 TCP (connection failed or established)

For TCP, incoming or outgoing, failed connections and established connections were used to create indicators. Listing 9 in Appendix C shows one of the three similar filters used to gain the information needed. The three filters were used to get TCP SYN packet information, TCP SYN-ACK packet information and TCP ACK packet information into three separate CSV files.

Each CSV file's rows in the SYN and ACK files contained a source IP address, destination IP address, source port and destination port. The SYN-ACK files rows were slightly

different with destination IP address, source IP address, destination port and source port. This made for easy comparisons, when establishing if a TCP SYN had a corresponding SYN-ACK and ACK.

Each row of the CSV was iterated through and the IP addresses were tested against the clean IP list. Once it had been confirmed if a malicious TCP connection had been established or not, a CybOX Network Connection, Socket Address and Port object were made. The Network Connection's `layer3_protocol` and `layer4_protocol` are set to IPv4 and TCP accordingly. The Network Connection's `destination_tcp_state` or `source_tcp_state` are set to either "SYN-SENT" or "ESTABLISHED", depending on if the connection had been established or not.

If a connection was outgoing, the Socket Address's `ip_address` is set to the destination IP address and the Port's `port_value` and `layer4_protocol` are set to the destination port and TCP.

Once all the values had been set, the Port is added to the Socket Address, which is added to the Network Connection. Then a STIX indicator is made, which wraps the Network Connection and has appropriate metadata set. Finally the Indicator ID number is added to a STIX report and the Indicator and Report are added to a STIX Package object, that was created at the beginning of the module.

4.5.6 UDP connection

An incoming or outgoing UDP connection was used to create the UDP connection indicators. The filter listed in Listing 10 in Appendix C, which extracts the source port, destination port, source IP and destination IP, was used on all UDP packets to create a CSV with the four values.

Similarly to the TCP connection, CSV was iterated through and the IP addresses were tested against the clean IP address list. Now that only malicious packet data is chosen, a CybOX Network Connection object is created, together with a Socket Address and a Port. The Network Connection's `layer3_protocol` and `layer4_protocol` were set to IPv4 and UDP respectively. If the connection was found to be outgoing, the destination IP and port were set in the Socket Address's `ip_address` and Port's `port_value` accordingly. Incoming connections similarly used the source IP and port values.

The Port is added to the Socket Address, which is added to the Network Connection with some additional metadata, like a title and description. The same process as above, of adding the CybOX object to the STIX indicator, then the Report and Package is used.

4.5.7 HTTP request (GET/POST)

Outgoing HTTP GET or POST requests were used to create indicators and the `tshark` filter shown in Listing 11 in Appendix C was used to get the many values needed. Like the other `tshark` filters shown, it creates a CSV file with all the selected values. Table 4.8 shows the 16 different values that the filter can extract.

The CSV file is then iterated through with the host name being filtered through the secondary filter for clean domain names. For the malicious requests, a CybOX Network Connection object is created, along with a HTTP Session, URI, Port object.

The HTTP Session contains the sub objects starting with the HTTP Request Response, which consists of a HTTP Client Request. The HTTP Client Request is made up of a HTTP Request Line, HTTP Request Header. The HTTP Request Line's `http_method`, `value` and `version` fields were to house three vaules from the CSV.

HTTP Request Header's `accept`, `accept_language`, `accept_encoding`, `authorization`, `cache_control`, `connection`, `cookie`, `content_length`, `content_type`, `date` and `proxy_authorization` fields were used to store 11 of the 16 values (if they exist). The HTTP Request Header's `host` field is used and it takes the Port and URI objects and their fields.

Once an HTTP Session object has been set up, a sub object called Layer 7 Connection is made and the HTTP Session is added to it.

The Network Connection's `layer3_protocol` and `layer4_protocol` and `layer7_protocol` are set to IPv4, TCP and HTTP respectively. The Layer 7 Connection is then added to the Network Connection, which is in turn added to a new STIX indicator object with the respective metadata.

Finally the same process as above, of adding the CybOX object to the STIX indicator, then the Report and Package is used.

4.5.8 DNS request

The information from outgoing DNS requests was chosen to be used in the creation of DNS request indicators. The `tshark` filter show in Listing 12 in Appendix C was used to extract seven values to aid in the creation of CybOX Network Connection, DNS Query, Socket Address and Port objects.

Like the other `tshark` filters, a CSV is created with each row consisting of the seven values (if they are found in a packet). The CSV is iterated through, testing the domain name against the list of clean domains and if a domain is potentially malicious, the row is passed to a method that uses the information to create an Indicator.

First a CybOX Network Connection object is instantiated and its `layer3_protocol` and `layer4_protocol` and `layer7_protocol` are set to IPv4, UDP and DNS respectively. A Port and Socket Address object are made and the destination IP and port were set in the Socket Address's `ip_address` and Port's `port_value` accordingly.

To store the information on a DNS Query, a Network Connection sub object, Layer 7 Connection is created. This takes as input a CybOX DNS Query object, which in turn takes a DNS Question object. The DNS Question's properties `qname` and `qtype` are used to store the queried domain name and its type.

These objects are then wrapped by the correct CybOX wrappers described above and a STIX indicator is created that wraps the outer CybOX Network Connection. Finally the Indicator's ID is placed in the STIX report and the Indicator is added to the STIX package.

4.5.9 FTP connection

FTP was chosen because some malware utilises FTP servers to push and pull information. Since it is an unencrypted connection, details about the server and the usernames and passwords used to log in can be retrieved. For the POC system, a few types of FTP interactions were chosen to be represented.

FTP traffic (incoming or outgoing) with the return codes 220, 230 and 250 or request commands of USER, PASS, STOR and RETR were chosen to create FTP indicators (Postel and Reynolds, 1985). The explanation of the codes and commands are shown

in Table 4.10 and 4.11. These commands were chosen to be used because they show an overview of how a sample is interacting with a FTP server.

Table 4.10: FTP Server Return Codes used

Return Code	Description
220	Service ready for new user
230	User logged in, proceed. Logged out if appropriate
250	Requested file action okay, completed

Table 4.11: FTP Request Commands used

Request Command	Description
USER	Authentication username
PASS	Authentication password
STOR	Accept the data and to store the data as a file at the server site
RETR	Retrieve a copy of the file

The `tshark` filter shown in Listing 13 in Appendix C, shows the eight values being extracted from FTP packets and placed in a CSV file. While iterating through the CSV files only packets with the chosen response code and request commands were chosen for the creation of Indicators.

To start off the creation of a STIX indicator, a CybOX Network Connection is crated and the `layer3_protocol` and `layer4_protocol` and `layer7_protocol` are set to IPv4, TCP and FTP respectively. It is then decided if a connection is incoming or outgoing. A Socket Address and Port object are created and if a connection is outgoing, the destination IP and port are placed in the objects. If the connection in incoming, the source IP and port are placed in the objects.

Next the Socket Address and Port are wrapped by the Network Connection object. An Indicator object is created and since there is no variable to hold the FTP return code or request command, the Indicator `description` field was used. For the return codes the description found in Table 4.10, was placed in the Indicator `description` field. For the FTP request commands, both them and the requested values (such as a username, password or file name) were placed in the `description` field.

Finally the Indicator ID number is added to a STIX report and the Indicator and Report are appended to a STIX Package object, that was created at the beginning of the module.

4.5.10 SSH connection

SSH connection attempts and SSH packets containing a SSH public key are used in the creation of these indicators. The filter shown in Listing 14 in Appendix C extracts seven properties from and SSH packet and saves them to a CSV file. While iterating through the CSV file, the IP addresses are checked against the clean IP address list and if they are found to be potentially malicious, the CSV row is used to create a indicator.

For each chosen row a STIX Network Connection object is created and it is decided if the connection is incoming or outgoing. The Network Connection's `layer3_protocol` and `layer4_protocol` and `layer7_protocol` are set to IPv4, TCP and SSH respectively. Next a CybOX Socket Address and Port object are created and if the connection if found to be incoming, the source IP and port values are added to these objects accordingly.

If the packet has only the TCP SYN packet set, a STIX indicator is created with metadata describing the indicator as a SSH connection attempt. If the packet contains an SSH public key, a STIX indicator is created and the metadata describing a public SSH key is added with the SSH key itself being placed in the Indicator's `description` field.

Finally the Socket Address and Port are wrapped by the Network Connection object, which is then wrapped by the Indicator. The Indicator's ID is added to the Report and the Indicator itself is added to the Package.

4.6 STIX to IDS and Firewall rules

Each STIX package was parsed using a Python script and and the `stix-python`[17] library. For each type of indicator, certain information was extracted which is shown in Table 4.12. IPFW is used in FreeBSD and does not support string searching, so the DNS and HTTP indicators could not be converted into IPFW rules. Since this is a POC system, it can be extended in the future to cater for more IDS and firewall rules.

[17] https://github.com/STIXProject/python-stix

Table 4.12: Properties used for each rule from an indicator

	Snort	IPFW	Iptables
IP address	IPv4 address	IPv4 address	IPv4 address
ICMP	IPv4 address	IPv4 address	IPv4 address, Type
UDP	IPv4 address, Port	IPv4 address, Port	IPv4 address, Port
TCP	IPv4 address, Port	IPv4 address, Port	IPv4 address, Port
DNS	Domain, Port	-	Domain, Port
HTTP	Method, URI, Host, Port	-	URI, Host, Port
SSH	IPv4 address, Port	IPv4 address, Port	IPv4 address, Port
FTP	IPv4 address, Port	IPv4 address, Port	IPv4 address, Port

4.6.1 Snort

Snort IDS rules were made for all of the STIX indicators. The action chosen for the rules was `alert`, which logs the event and sends an alert message. The appropriate protocol was used for each rule, either TCP, UDP, ICMP or IP.

Shown in Table 4.12, in the Snort column, are the network properties used to create each rule. UDP, TCP, SSH and FTP rules were made with IPv4 addresses and Ports from the STIX indicators. IP address and ICMP rules were made with just an IPv4 address. Snort has a `content` field that takes a string and uses it to match against packets that have a certain IP address and port combination. This field was used in the DNS rules and HTTP rules.

The DNS used the domain (for the `content` field) and Port. The HTTP rules used the HTTP method (for the `msg` field), URI (for the `content` field in the `http_uri`), host (for the `content` field in the `http_header`) and port values from the STIX indicators.

Each rule had its own appropriate message (describing the rule and including the SHA 256 hash of the sample) that will be written in a specified log file. They also have a `classtype` set, which is a keyword used to characterise a rule. Class types have different priorities from high, medium, low and very low. For this system, the classtypes were set to `bad-unknown` for proof of concept, which has a medium priority level and is for potentially bad traffic.

4.6.2 Iptables

Rules for Iptables were created for all of the types of Indicators. The network properties from each Indicator used to make the Iptables rules are shown in Table 4.12. Thankfully Iptables has a string matching extension that can match a string anywhere in a packet[18], which was used for matching domains and URIs.

Similarly to the Snort rules, the UDP, TCP, SSH and FTP rules were created with an IPv4 address and port from each Indicator. The `TARGET` of these rules were set to `LOG_AND_REJECT` and the protocol was set to TCP or UDP accordingly. The rules catered for incoming and outgoing traffic (depending on what an Indicator specified). The rule to filter a suspicious IP address was made with just the address.

Rules to filter suspicious outgoing DNS requests were made with a domain and port from an Indicator and the rule's protocol was set to UDP. The domain was used by the string matching part of the rule and the `TARGET` of these rules was `LOG_AND_REJECT`. It was suggested in Marie (2005), that the string matching extension should not be used together with the `TARGET` `LOG_AND_REJECT` with string matching because it can easily be defeated by any IDS evasion method.

Outgoing HTTP request filter rules used the host, URI and port from the IOCs and the URI was used for string matching and the rule's protocol was set to TCP. The `TARGET` of these rules were set to `LOG_AND_REJECT` and an appropriate log message was set.

Finally the ICMP filtering rules used the IP address and type values from the IOCs and protocol was set to ICMP. These rules were set to be for incoming and outgoing traffic and the `TARGET` was set to `LOG_AND_REJECT`.

4.6.3 IPFW

IPFW was used to create rules for the ICMP, UDP, TCP, FTP and SSH protocols. Like the Snort and Iptables rules for UDP, TCP, FTP and SSH, the IP address and ports from the IOCs were used for incoming and outgoing packets. The TCP, FTP and SSH rules had the TCP protocol set and the UDP appropriately had the UDP protocol set. The rules were set to use the reject method, which drops packets. The ICMP filter rule just used the IP address from the IOCs with the ICMP protocol set.

[18] http://www.netfilter.org/documentation/HOWTO/netfilter-extensions-HOWTO-3.html

4.7 Chapter Summary

This section started off with a comparison between CybOX and OpenIOC network objects, which found that CybOX has many more and more comprehensive objects for the creation of network based cyber observables. An overview of the system and reasons for choosing CybOX and STIX were made clear with a discussion on the CybOX and STIX objects chosen to be used in the system. Details on how traffic from the sandbox was filtered to eliminate as much non-malicious traffic as possible was then explored. Next a detailed look at the reporting module and how each object was generated from network traffic is presented, with the section ending with an overview of how IDS and firewall rules were generated from the aforementioned indicators. Chapters 5 and 6 will use the objects described in this chapter and conduct evaluations using them.

Chapter 5

Evaluation

Before IOCs are generated, it is important to look at network traffic and observe what type of traffic is present and how it can be used. This chapter contains two sections that show analysis of traffic and the indicators generated by the framework. Section 5.1 starts off by presenting the three datasets used in this chapter. Sections 5.2 and 5.3 are two case studies where malware is submitted to the framework and the traffic is analysed in detail. The last section, Section 5.4, is the final case study that compares the amount of IOCs generated of three different analysis times.

5.1 Datasets

The following datasets used in the three evaluations were received from VirusTotal[1] and VirusShare[2] and consist of RATs, ransomware and some unknown samples that were labelled to create certain traffic.

5.1.1 Dridex

Dridex (US-CERT, 2015) was chosen as it is a topical malware strain and since the framework can take any malware binary as an input, it was thought that Dridex would be a useful demonstrative family. For the purposes of this research, 100 binaries identified to contain variants of the Dridex strain were downloaded from VirusShare and analysed to show how the system operates. These were dated within the last twelve months.

[1] https://www.virustotal.com/
[2] https://virusshare.com/

5.1.2 Ransomware

A total of 585 malware samples were sourced from Virustotal using the API to search for and download samples. To get a variety of samples a number of search variables were used. Two variables used, were the file type (or extension) and the malware variety (by name). Two file types were chosen which are commonly used for ransomware, which were executables such as EXE and DLL. The download query retrieves the top 25 samples that match the query. For the malware variety name, `malwarebytes:locky` was used, as Malwarebytes[3] had a generic naming system that uses the common ransomware names. The varieties of ransomware that were targeted are show in Table 5.1.

Table 5.1: Ransomware varieties analysed

Name	# of samples
Locky	71
CryptoWall	122
DMA Locker	22
VirLock	99
Chimera	2
CTB Locker	30
FileLocker	63
TeslaCrypt	109
CryptoLocker	39
TorrentLocker	28

5.1.3 Traffic and Analysis Time

This dataset consisted of 325 samples, 100 of which were retrieved from Virustotal with labels that indicate that they generate packets of a certain protocol. The types chosen were SSH, FTP, DNS and UDP. The first 25 files that were tagged with `tag:ftp`, `tag:ssh`, `tag:suspicious-dns` or `tag:suspicious-udp` were used. Another 25 were labelled to execute on Windows XP and the other 200 samples were a random assortment of malware.

[3]https://www.malwarebytes.com/

5.2 Dridex

Parts of this case study were previously published in Rudman and Irwin (2016a). Dridex is a type of malware with the primary goal of infecting computers, stealing credentials, and obtaining money from victims bank accounts (US-CERT, 2015). It was first observed in the wild in November 2014 (Sanghavi, 2015). When the malware is installed, the computer becomes part of a botnet (US-CERT, 2015), which can be used to send phishing emails.

In mid October 2015 many command-and-control servers used by the Dridex botnet were taken down by the Federal Bureau of Investigation (FBI) with the help of the National Crime Agency (NCA) (Trend Micro, 2015). However in late October, security researchers found signs that the botnet might still be functioning (Bisson, 2015). According to O'Brien (2016), Dridex was barely seen from 24 December 2015, but resumed its operations again in early January 2016. In February of 2016, it was found that part of the Dridex botnet may have been hacked as part of its distribution channel was changed by replacing malicious links with an installer for the Avira antivirus (Frink, 2016). In March 2016, the Dridex botnet started to send SPAM emails with JavaScript attachments that eventually install Locky ransomware (SecurityWeek News, 2016). According to Zorz (2016b), in May 2016, the botnet was compromised again to distribute a "dummy file" instead of the Dridex binary.

The actual Dridex malware is spread through multiple types of SPAM email attacks with a Microsoft Word or Excel document attached, which includes a payload that downloads the malware (Sanghavi, 2015). Macros must be enabled in Microsoft Word for the payload to work (Sanghavi, 2015). Once installed, Dridex uses HTML injections to retrieve banking details (Sanghavi, 2015) and can even steal user credentials through keystroke logging, form grabbing, stealing cookies and screenshots US-CERT (2015) Trend Micro (2015) O'Brien (2016). It is able to steal banking details of nearly 300 financial institutions of generally English speaking countries (Zorz, 2016a).

5.2.1 Network Traffic Overview

From the 100 Dridex samples that ran for 30 minutes, Cuckoo was able to execute 100% of them, with 50 samples generating network traffic, which added up to 31,45 MB. It is unclear as to why some of the samples did not show network activity. According to Rossow *et al.* (2011), this may be because the samples were invalid, or only active when there is

user activity, or they detected the sandbox and stopped working. It is suspected that it may also be because the malware needed more time to run (longer than 30 minutes) in order to generate traffic. Another possibility is that some of the malware was designed to activate only within a certain time period (which had expired). The take down of much of the Dridex infrastructure in October 2015 may also have played a role in the reduced volumes of observed traffic. The following subsections will discuss the results found when analysing the filtered PCAPs which are assumed contain malicious traffic. It will show information about TCP connections, DNS requests and responses, HTTP requests and any hard-coded IP addresses.

5.2.2 DNS

Of the 50 samples that generated traffic within 30 minutes of running, 40 of the samples used the DNS protocol (port 53 UDP). Of the 40 samples, all of them used the pre-configured DNS server. If some of the samples had used their own resolver, the information could have been used in the creation of an IOC. Some malware strains use their own iterative/recursive resolvers to avoid leaving traces in logs or caches of pre-configured resolvers on a victim network (Rossow *et al.*, 2011).

Table 5.2 shows the 14 domain names that were requested by some of the 40 samples. Ten of the domains were resolved, which could mean that the other four are no longer in use or they could be blocked by the pre-configured resolver set by the university.

Looking at the TTL values of the ten resolved domains, seen in Table 5.2 the most popular domain, `icanhazip.com` has a TTL of 5 minutes. There are three very small values seen such as 3, 10 and 20 seconds, which are generally related to Content Delivery Networks (CDNs) (Fujiwara *et al.*, 2013). In this case the three domains are related to online certificates, which could explain why the values are low. A TTL of zero, which is not seen in these results, can indicate the use of fast flux domains which are used to give flexibility among the command and control infrastructure of bots (Holz *et al.*, 2008). However, many domains using a TTL of zero could be included as part of an extended IOC in future work.

The most popular domain `icanhazip.com` was requests by 33 of the 100 samples and 66% of all samples that generated network traffic. This site is non malicious and is used to determine the IP address of the host that loaded the page.

Table 5.2: Domain names requested from 40 of the samples

Domain name	Number of samples	IPs resolved	DNS TTL
icanhazip.com	33	64.182.208.185 64.182.208.184	300
th.symcb.com	3	23.42.5.163	20
th.symcd.com	3	23.42.11.27	3
ocsp.thawte.com	3	23.42.11.27	10
crl.thawte.com	2	23.42.5.163	900
ho7rcj6wucosa5bu.tor2web.org	1	194.150.168.70 38.229.70.4 65.112.221.20	3600
api.ipify.org	1	50.17.192.14 107.20.229.58 54.243.252.101	60
malwagroup.org	1	182.50.130.67	3600
thedirtydelicious.com	1	184.168.27.45	600
nerdmeetsgirl.com	1	184.168.47.225	600
tanhadhidown.ru	1	-	-
herssofhaprigh.ru	1	-	-
nohissandbo.ru	1	-	-
mcreport.org	1	-	-

According to Teo (2015), AplusWebMaster (2015) and Duncan (2015b), malware authors use this domain and similar sites to obtain the IP of an infected computer, as part of the environmental determination used prior to contacting the Command and Control (C&C) node(s). This domain is often suggested to be used as an IOC according to Duncan (2015b). In this case, it seems like using this domain as an IOC is a good idea as it appeared many times from the samples. The other domain, `api.ipify.org`, is also a non malicious site and is similarly used to check the IP address of a client computer.

As seen in Table 5.2, three distinct samples were seen using the following domains: `th.symcb.com`[4], `th.symcd.com`[5] and `ocsp.thawte.com`[6] and two of the three samples also queried `crl.thawte.com`[7]. These domains are non malicious in themselves, but have

[4]https://www.virustotal.com/en/domain/th.symcb.com/information/
[5]https://www.virustotal.com/en/domain/th.symcd.com/information/
[6]https://www.virustotal.com/en/domain/ocsp.thawte.com/information/
[7]https://www.virustotal.com/en/domain/crl.thawte.com/information/

been identified to be requested by malware in some cases as referenced above on VirusTotal.

The domains `malwagroup.org`, `thedirtydelicious.com`, `nerdmeetsgirl.com`, `tanhadhidown.ru`, `herssofhaprigh.ru`, `nohissandbo.ru` were requested by the same sample and according to the VirusTotal report[8], these domains are used to download the Pony and Dyre banking malware. These six domains can be combined used to create a single IOC as they are only seen in one sample.

One sample used the ho7rcj6wucosa5bu.tor2web.org and `api.ipify.org` domains. As stated above, `api.ipify.org`, is used to retrieve the IP address of an infected host and `ho7rcj6wucosa5bu.tor2web.org` is a known malicious domain[9] Duncan (2015a) and is using the Tor2web network gateway[10]. These two domains can be used to create two indicators or one combined indicator. The last domain, `mcreport.org`, was not resolved and is only mentioned as the result of the analysis of a malicious file on Malwr[11]. This domain seems to be out of use at the moment, but can still be used to create an IOC.

Other than the domains themselves, the IP addresses that were resolved can also be used in the generation of IOCs. The resolved IP addresses may change, so this may not be as effective as an IOC because it may not be relevant for long.

5.2.3 HTTP

Only three samples from the 50 that generated network traffic used the HTTP protocol. This is a significantly small amount of of samples and does not give too much to work with in terms of IOC generation. Between the three samples, there were seven HTTP requests with three `200 OK` replies and four `404 Not Found` replies.

When looking at some of the HTTP header fields, it was found that six out of the seven User-Agent fields did not correspond to the programs and operating system on the testing VM. For example, one of the requests specified the Opera browser (not installed on the VM) and another specified Ubuntu as the operating system. The User-Agents are shown in Table 5.3 along with the ID of the sample. Sample 4 used three different User-Agents

[8] https://www.virustotal.com/en/file/facc9a5f02e8d18c9cbac9ee760ffa38b2854e5d5c89a529e368be8857bc55a9/analysis/

[9] https://www.virustotal.com/en/domain/ho7rcj6wucosa5bu.tor2web.org/information/

[10] https://tor2web.org/

[11] https://malwr.com/analysis/ZjJkMmJkNTk3YmUyNDliZWFkMDNiZmQ3MmQ1YjJkZGU/

and so did Sample 48. It is interesting that every HTTP request had a different User-Agent value. Sample 99 was the only sample to use a correct User-Agent field. In Rossow *et al.* (2011), which also found that malware forges their own User-Agents, it was suggested that one sample can use a different User-Agent because of the modular nature of malware. Each different module has its own way of forging an HTTP request.

Table 5.3: HTTP User Agents and sample IDs

User Agent	Sample ID
Mozilla/5.0 (Windows NT 5.1; rv:21.0) Gecko/20130401 Firefox/21.0	4
Mozilla/4.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/5.0)	4
Mozilla/5.0 (compatible; MSIE 9.0; AOL 9.7; AOLBuild 4343.19; Windows NT 6.1; WOW64; Trident/5.0; FunWebProducts)	4
Mozilla/5.0 (Windows NT 6.2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/28.0.1464.0 Safari/537.36	48
Opera/9.80 (Windows NT 6.1; U; es-ES) Presto/2.9.181 Version/12.00	48
Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:21.0) Gecko/20130331 Firefox/21.0	48
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; GWX:QUALIFIED)	99

Sample 4, mentioned in Table 5.3, HTTP GET requested three URIs. These were `70.127.18.124/online.htm`, `178.137.58.176/main/htm` and `94.139.196.46/home.htm`. The first URI did not receive a reply and the last two received a `404 Not Found` reply. Sample 4 did not send any DNS requests, so these addresses were not resolved from a domain name.

Sample 48, sent three unique HTTP GET requests to IP addresses that had not been resolved from DNS requests. The three requests were `79.119.76.125/online.htm`, `79.119.76.12/welcome.htm` and `122.118.192.8/index.htm`, the last of which brings up a login page to a router. There is most likely a compromised computer behind the router that is part of the Dridex botnet. Since the previously mentioned IP addresses had not been

resolved from DNS requests it makes them a good property to add to part of an indicator, as it may have been hard coded. Each of the HTTP requests can be made into an HTTP Session CybOX object, that can be wrapped by a STIX indicator.

Sample 99 sent three HTTP GET requests seen in Figure 5.1. These headers are very similar and show that the sample was attempting to download a file called 'k1.exe' from the three domains. The headers have identical accept-language, accept-encoding, accept, and user-agent values in the fields and two of the headers have the same GET parameter. The malware most likely sends three requests for the same file (assuming it is the same file) in case one or more of the domains is down. In the case of the sample run, the `nerdmeetsgirl.com` request (shown in Figure 5.1a) received a `200 OK` response. For HTTP requests for `thedirtydelicious.com` and `malwagroup.org`, shown in Figures 5.1b and 5.1c, response codes were received indicating that the files were no-longer present for download.

From the four different HTTP requests all of them used the GET request method. Rossow *et al.* (2011), analysed the network output of multiple types of malware samples, they also found that GET request was the most popular request method over POST.

5.2.4 Other Protocols

In terms of TCP requests, 42 out of the 50 samples that generated TCP traffic with 108 connections established and 869 connections failing. The samples did not show any malicious ICMP, UDP, FTP or SSH traffic.

5.2.5 Indicators Created

Each malware sample ended up with an XML file (STIX Package) containing all the indicators that were created. From the 40 samples that used DNS, 52 DNS Indicators were created, showing that some samples queried the same domains. Listing 15 in Appendix D shows a DNS request, for `ho7rcj6wucosa5bu.tor2web.org`, represented in the STIX indicator structure. The domain, `ho7rcj6wucosa5bu.tor2web.org`, was mentioned in Section 5.2.2 as a known malicious domain. As shown in Listing 15, the STIX part of the indicator has a `Title` 'DNS Request', the `Description` 'An indicator containing information about a DNS Request' and the `Time` set, which is the contextual information.

```
GET /wp-content/plugins/cached_data/k1.exe HTTP/1.0
accept-language: en-US
accept-encoding: identity, *;q=0
host: nerdmeetsgirl.com
accept: */*
user-agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows
NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR
2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729;
Media Center PC 6.0; .NET4.0C; .NET4.0E; GWX:QUALIFIED)
connection: close
```

(a) HTTP request to 'nerdmeetsgirl.com'

```
GET /wp-includes/simplepie/net/k1.exe HTTP/1.0
accept-language: en-US
accept-encoding: identity, *;q=0
host: thedirtydelicious.com
accept: */*
user-agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows
NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR
2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729;
Media Center PC 6.0; .NET4.0C; .NET4.0E; GWX:QUALIFIED)
connection: close
```

(b) HTTP request to 'thedirtydelicious.com'

```
GET /wp-content/plugins/cached_data/k1.exe HTTP/1.0
accept-language: en-US
accept-encoding: identity, *;q=0
host: malwagroup.org
accept: */*
user-agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows
NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR
2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729;
Media Center PC 6.0; .NET4.0C; .NET4.0E; GWX:QUALIFIED)
connection: close
```

(c) HTTP request to 'malwagroup.org'

Figure 5.1: Three HTTP GET requests generated by one of the samples

The DNS and underlying protocols, together with the DNS server IP address, port, domain and query type are provided in the CybOX part of the Indicator in Listing 15.

Seven HTTP GET STIX indicators were created and one of the Indicators for the host `nerdmeetsgirl.com` is shown in Listing 16 in Appendix D. The STIX indicator `Title` and `Description` are set to 'HTTP Request' and 'An indicator containing information about a HTTP Request' respectively, with the `Time` set near the bottom of the Listing. The HTTP and underlying protocols are stored in the CybOX `NetworkConnectionObj` part of the object. The `HTTPSessionObj` in Listing 16 houses the HTTP Method, URI, Version, Accept, Accept Language, Accept Encoding, Host and port.

Listing 17 in Appendix D shows a TCP Connection Established STIX indicator for the IP address `166.62.114.65`. This is a simpler than the DNS and HTTP ones as it has less properties. The STIX indicator `Time` is set near the bottom of the Listing, with the `Title` set to 'TCP Connection Established' and the `Description` set to 'An indicator containing information about a successful TCP handshake' as shown in Listing 17. The

properties in the CybOX Observable are the destination IP address, port and protocol.

5.2.6 Summary

The Dridex malware strain was used as an example set of malware for analysis where the samples generated PCAP files during dynamic analysis. An overview of the network traffic for DNS and HTTP protocols was shown, which resulted in some suspicious domain names, and HTTP request packets. The information gathered from these suspicious packets was used to automatically generate different types of IOCs. Although Dridex did not generate a large array of traffic, further tests of malware are shown in Section 5.3. However, it can be confirmed that network-based IOCs can be generated from dynamic malware analysis.

5.3 Ransomware

The concept of ransomware was introduced in Section 2.4. It is a type malware that restricts access to a part of a user's computer. This can range from locking the computer to encrypting important files and asking for a fee to grant access again. Section 5.3.1 will briefly discuss the setup of the system and Section 5.1.2 will introduce the ransomware samples used in this evaluation. Section 5.3.2 gives an in depth analysis of the network traffic observed from 10 different strains of ransomware shown in Table 5.1 and numbers of Indicators that were created from the gathered information. Finally Section 5.3.9 will show some Indicators that were created and explore their usefulness as Indicators for malware.

5.3.1 Setup

The malware was run through the Cuckoo Sandbox 1.2 along with a custom processing and reporting module that automatically analysed the traffic and generated Indicators. Each sample was run for 600 seconds in the Windows 7 VM described in Section 3.2. The desktop contained three Microsoft Word documents, a PDF, an image and several shortcuts. The documents and images were placed there to visually observe how they would change if encrypted, since Cuckoo generates screenshots. The VM had Internet access through a bridged adaptor, in which Cuckoo captured traffic from.

5.3.2 Network Statistics

Since certain ransomware connects to randomly generated domains (Abrams, 2013; Bottomley, 2015; Avast, 2016) to download a public key for some variations of ransomware, traffic including requests for generated domains may be expected. However many variants use encryption, like AES256, to and from the command and control servers[12]. Sometimes these URLs are infected WordPress sites[13].

Out of the 585 samples that were submitted, 579 samples executed and were successfully analysed by Cuckoo. Exactly 386 samples generated network traffic and had STIX Packages created for them, which is 66.67%. Each STIX Package full of Indicators were saved to individual XML files. Some of the files may not have executed because they were old, or had anti-VM detection.

The most popular protocol in terms of number of samples that used it was TCP, then DNS, HTTP and UDP. The observables seen from these protocols will be discussed in detail below, along with the statistics of each protocol. When counting the number of individual DNS requests, TCP streams, HTTP GET or POST, UDP streams and certain FTP, SSH and ICMP traffic, it is found that DNS was the top protocol. The summary of the traffic totals are shown in Table 5.4. TCP is the second most observed protocol, with HTTP second and UDP fourth with two streams. The rest of the protocols did not show any activity within the ransomware dataset.

Table 5.4: Top network protocols observed

Protocol	Total	%
DNS	7912	73.09
TCP	2014	18.60
HTTP	897	8.29
UDP	2	0.02
FTP	0	0.00
SSH	0	0.00
ICMP	0	0.00

[12]http://www.isightpartners.com/2015/09/teslacrypt-2-0-cyber-crime-malware-behavior-capabilities-and-communications/

[13]http://blog.brillantit.com/?p=15

5.3.3 TCP Connections

There were 373 samples that generated suspicious TCP traffic when run on the sandbox. The 373 samples collectively had 831 "TCP connection established" Indicators generated and 1183 "TCP failed connection" Indicators generated. Most popular ports were 80 then 443. Later in this Section, other ports that appeared occasionally are discussed. Of the 831 TCP EST indicators from the 373 samples, there were 237 unique ones.

Connections Established

The three largest groups of samples that queried the same IP address were 133 Locky and CryptoWall samples querying 93.184.220.20:80 (which displays a `404 - Not Found` message), 90 CryptoWall and DMALocker samples querying 216.146.38.70:80 (web page that displays the IP of the host) and 49 CryptoLocker and TorrentLocker samples querying 38.229.70.4:443 (Tor2Web website for the download of the tor browser). This demonstrates that multiple ransomware families connect to the same IP addresses, therefore making them potentially effective to be used in IDS or firewall rules.

Table 5.5: Top Outgoing TCP EST Ports and Total IOCs

Port	Total IOCs	Popular use	Ransomware Variant
80	589	HTTP	Locky, CryptoWall, DMALocker, VirLock, CTB Locker, FileLocker, TeslaCrypt, CryptoLocker, TorrentLocker
443	107	HTTP over SSL	Locky, CryptoWall, VirLock, CTB Locker, FileLocker, TeslaCrypt, CryptoLocker, TorrentLocker
9001	62	Games, Tor	TorrentLocker
139	16	NetBIOS	TorrentLocker, CTB Locker
9101	9	Bacula	CTB Locker, TorrentLocker
9007	5	Open Grid Service Client	TorrentLocker
9002	4	Bacula	CTB Locker, TorrentLocker
7654	3	SSH Tunneling	TorrentLocker
993	2	IMAP over SSL	TorrentLocker

The Top 9 destination ports observed are shown in Table 5.5. There were 12 other ports

found in one TCP Connection Established Indicator each. These include, 53, 4051, 5724, 9090, 8080, 9003, 9010, 9035, 9040, 10308, 20714 and 34521.

The top three ports for established connections are port 80, 443 and 9001. Port 80 makes up 70.9% of the ports observed for TCP Established Connections. As mentioned in Section 5.3.2, certain ransomware connects to domains through HTTP to download a public key in order to encrypt files, so port TCP/80 can be expected. The common port uses in Table 5.5 were found on (Speed Guide, 2016) and show some uses, like Tor, that relate to ransomware. Port 9001 is a default Tor port which was used by TorrentLocker samples.

Port 443 was used to connect to Tor related sites like 193.23.244.244 (queried by CTB Locker samples) which gives the message "This is a tor authority" through port 80 in a standard browser. Another request was to 38.229.70.4, which is the Tor2Web homepage, and was queried by CryptoLocker and TorrentLocker samples. The rest of the IPs included compromised sites, malicious sites, or they were unused at the time of this write up.

The three largest groups of IP addresses connected to by a sample were 24 IP addresses contacted by one CryptoWall sample (geolocated to North and South America and Europe), 23 IP addresses connected to by another CryptoWall sample (geolocated to Europe, North and South America and Indonesia) and 21 IP addresses (geolocated to Europe and North America) connected to by one TorrentLocker sample. For each of these samples, the IP addresses did not have a dominant origin country, but a variety of European countries and the USA, Brazil and Canada. This shows that the ransomware samples, from the dataset, that connected to a large amount of IP addresses use a variety of countries. In Section 5.3.5, it is shown that many of the hosts connected to are compromised WordPress or other sites, which could account for the variety of location of IP addresses used.

Connections Failed

The three largest groups of samples that queried the same IP address were 98 Virlock samples querying 200.119.204.12:9999 and 190.186.45.170:9999, 89 VirLock samples querying 200.87.164.69:9999, and 72 CryptoWall, DMALocker and VirLock samples querying 184.106.112.172:80.

Table 5.6 shows the top 8 TCP ports from failed connections. There were five more ports that was observed once each. Like Table 5.5, port 80 is the most frequently used, with port 443 also being highly ranked. Port 9999 was the second highest ranked port, and

together with three other ports (666, 8080 and 1604) in Table 5.6, are known to be used by malicious software.

Table 5.6: Top TCP Outgoing Failed ports and the total IOCs

Port	Total IOCs	Popular use	Ransomware Variant
80	903	HTTP	Locky, CryptoWall, DMALocker, VirLock, CTB Locker, FileLocker, TeslaCrypt, CryptoLocker, TorrentLocker
9999	53	Trojans	VirLock
443	51	HTTP over SSL	Locky, CryptoWall, VirLock, CTB Locker, FileLocker, TeslaCrypt, CryptoLocker, TorrentLocker
8080	39	HTTP, games, trojans	TorrentLocker
666	6	Trojans	VirLock
1604	2	DarkCommet RAT	CryptoWall
9101	2	Bacula	CTB Locker, TorrentLocker
20050	2	Games	FileLocker

Largest group of IP addresses with failed connections by a group of samples were, 448 IP addresses with failed connections from 1 CTB Locker sample. This is an example of a sample whose infrastructure may be down. The second largest group was 7 IP addresses with failed connections from 4 FileLocker samples and the third largest was 6 IP addresses with failed connections from 3 TeslaCrypt and 3 Locky samples. These show that the same IP addresses are used in different samples and may make good IOCs, if they are not common non-malicious IPs.

Interesting Ports

Port 80 and port 443 were the most popular ports, shown in Table 5.5 and 5.6, and are used commonly for web applications, but there were some other uncommon ports found too. Although used in one Indicator each, they are shown in Table 5.7 along with the ransomware variant that used it. TorrentLocker and CryptoWall samples seemed to use the most unusual ports. A number of the TCP ports, like port 2556, 9090, 9035, 9040, 9010 and 53, are known to be used maliciously by certain malware according to (Speed

Guide, 2016). Speed Guide (2016) uses a number of external resources like IANA Official Ports list[14], SANS Institute, nmap.org services list and gameconfig.co.uk (for port setting for online games) to gather the port information. Since some of the ports are inherently malicious, it could mean that some of the samples had modules of the certain variants, or the authors chose them at their will.

Table 5.7: Interesting Ports Used Once

Port	Ransomware Variant	Connection
10308	CTBLocker	Established
4051	CTBLocker	Established
9035	TorrentLocker	Established
9040	TorrentLocker	Established
5724	TorrentLocker	Established
9003	TorrentLocker	Established
9010	TorrentLocker	Established
20714	TorrentLocker	Established
9090	FileLocker	Fail and Established
53	DMALocker	Established
34521	CryptoWall	Established
2556	CryptoWall	Fail
2448	CryptoWall	Fail
9796	CryptoWall	Fail
10405	CryptoWall	Fail

5.3.4 DNS

A large number of samples used the DNS protocol, with 354 out of the 386 that generated traffic, which is 91.7%. Sometimes malware uses their own resolvers (DNS server) (Rossow *et al.*, 2011), however all of the files used the preconfigured DNS resolver of the VM. It was suggested by Rossow *et al.* (2011) that DNS resolver logs could be an interesting source for network based malware detection, but this is not the case in this instance.

A total of 7912 DNS requests were observed, with 903 unique domains seen. Only 223 of the 903 domains were resolvable A total of 2296 DNS Query Indicators were created, distributed among the 354 STIX Package files.

[14]https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

Domains

There were 903 unique domains queried in DNS requests and the top 20 domains by number of files that queried them are shown in Table 5.8. The total column shows the number of individual samples that queried a domain. The top ranked domain was queried by 160 samples including four variations of ransomware (Locky, CryptoWall, DMA Locker and VirLock). The `cdp1.public-trust.com` domain houses Certificate Revocation Lists (CRLs) and is controlled by Verizon and may not be suited for the contents of an Indicator as it is technically a safe domain. The second most queried domain, `api.bitcoincharts.com`, which provides a simple API to include markets data in applications, was most likely used to show BitCoin market information to the victim and is a benign domain.

The domains queried by TeslaCrypt seem to be compromised sites, domains that are no longer active or sites that are down. The last domain in Table 5.8, `ipecho.net`, is for a site that returns the IP address of the host (similar to `icanhazip.com` from Section 5.2). Since many of these domains are not malicious themselves (the starred ones), it demonstrates the risk of automated generation of Indicators. As atomic Indicators, these domains could cause false alarms if used in individual IDS or firewall rules. However, if a compromised site is hosting a malicious file, a field in a STIX indicator wrapper can be used to give a relevance period.

The Resolved column in Table 5.8 shows if domains were resolved or not. It is shown that 19 out of the 20 domains were resolved, which is a likely result as many of the sites are compromised or non-malicious. If they were malicious, they may not be constantly registered because they were generated by a DGA. The Contacted column in Table 5.8 shows which domains were later contacted via HTTP, which was 18 out of the 20 shown. The domain `mahmutersan.com.tr`, was not resolved, so therefore could not be contacted and the domain `kknk-shop.dev.onnetdigital.com` was resolved, but not contacted.

There were 658 domains out of the 903 seen that were queried only once. Many of these look like they were generated by a Domain Generation Algorithm, as mentioned in Section 2.6.

Table 5.8: Top 20 domain names requested

Rank	Domain	Total	Ransomware families	Resolved	Contacted
1	cdp1.public-trust.com *	160	Locky, CryptoWall, DMALocker, VirLock	Yes	Yes
2	api.bitcoincharts.com *	46	VirLock	Yes	No
3	naturstein-schubert.de *	25	TeslaCrypt	Yes	Yes
4	csskol.org *	25	TeslaCrypt	Yes	Yes
5	forms.net.in *	25	TeslaCrypt	Yes	Yes
6	kknk-shop.dev.onnetdigital.com *	25	TeslaCrypt	Yes	No
7	casasembargada.com *	25	TeslaCrypt	Yes	Yes
8	mahmutersan.com.tr *	25	TeslaCrypt	No	No
9	www.informaticauno.net *	22	TeslaCrypt	Yes	Yes
10	edge-institut.org *	22	TeslaCrypt	Yes	Yes
11	goktugyeli.com *	18	TeslaCrypt	Yes	Yes
12	saludaonline.com *	18	TeslaCrypt	Yes	Yes
13	newculturemediablog.com *	18	TeslaCrypt	Yes	Yes
14	conspec.us *	16	TeslaCrypt	Yes	Yes
15	iqinternal.com *	16	TeslaCrypt	Yes	Yes
16	tmfilms.net *	16	TeslaCrypt	Yes	Yes
17	ahlanmedicalcentre.com *	15	TeslaCrypt	Yes	Yes
18	cam-itour.info	15	TeslaCrypt	Yes	Yes
19	specializedaccess.co.uk *	14	TeslaCrypt	Yes	Yes
20	ipecho.net *	13	CTBLocker, TorrentLocker, FileLocker	Yes	Yes

Table 5.9 shows the number of unique domains queried by each ransomware family. All of the domain query types were type `A`, which are used for the conversion of domain names to a corresponding IP address. FileLocker was the top ranked with 306 unique domains, with Locky second with 206 domains and TorrentLocker third with 149 domains. None of the domains queried by FileLocker made the top 20 most queried domains. This shows that the 63 FileLocker samples queried a large variety of domains, but none of the domains were queried by more than 12 samples. For example the

domains `xolydjuugvlmyokoe.net`, `spffhtmvbktlfqjgv.com`, `gptfsjfjqapqcltae.pw`, `tfdeipoabhjkjdycx.in`, `jklmmoodyynqymyri.me` and `blkvqfrhjvilwywyf.cc` and 161 more were generated by one FileLocker sample. These domains consist of a 17 character string and a top level domain of either `net`, `com`, `pw`, `in`, `me`, `cc`, `su` or `tw`. When one sample was found querying a large amount of domains, they were usually found to have a DGA.

TeslaCrypt samples queried a large variety of domains, shown in Table 5.9 and also had 17 domains in Top 20 domains queried in Table 5.8. This shows that groups of TeslaCrypt samples query the same domains and that there are many domains being used by the malware authors. It is interesting to note that some families did not query a large amount of domains. These were CTB Locker, CryptoLocker, VirLock and DMA Locker (Chimera samples did not show any network activity). It could mean the malware authors are confident that the domains being queried would be resolvable, unlike when domain generation algorithms are used and many of the domain are not being used, or these samples did not need to query domains.

Table 5.9: Ransomware families and number of unique domains

Rank	Family	No. of Domains
1	FileLocker	306
2	Locky	206
3	TorrentLocker	149
4	CryptoWall	148
5	TeslaCrypt	122
6	CTB Locker	7
7	CryptoLocker	6
8	VirLock	2
9	DMA Locker	1
10	Chimera	0

Table 5.10 shows domains that were queried by more than one family of ransomware. The group column shows the combinations of ransomware and the second column shows the domains that the combination of ransomware queried.

It is interesting that CryptoWall and TeslaCrypt have 26 domains in common. However, when doing some research (Sinitsyn, 2015), it was found that TeslaCrypt 2.0 uses the same HTML page, that shows the ransom note of CryptoWall 3.0. Dela Paz (2016) claims that

TeslaCrypt has been disguised as CryptoWall since TeslaCrypt 2.0, so this may explain the similar domains. Taking a closer look at the samples and how they were labelled in VirusTotal, it was found that two samples that were labelled as CryptoWall by one of the search properties that was used previously, were labelled as TeslaCrypt by the other antiviruses so it is most likely mislabelled.

Domains such as `myexternalip.com`, `ipecho.net`, `ip-addr.es` and `wtfismyip.com` are used to obtain the IP address of a host, that may be behind a NAT. From the Table 5.10, it can be seen that these types of domains are popular for ransomware and are used by numerous families. Although not malicious themselves, they could be used in combination with other indicators found during analysis to create a more accurate indicator for a sample.

`en.wikipedia.org` and `www.torproject.org` were DNS queried, because sometimes ransomware shows an informative webpage, with a link to Wikipedia about RSA encryption and a link to download the Tor browser. These domains should not be used as IOCs alone, or should be filtered out in the future.

Looking at the last row of Table 5.10, it is found that Locky and TeslaCrypt samples query the same domains. Taking a closer look at the samples that queried these domains, one that was downloaded as TeslaCrypt was labelled mainly as TeslaCrypt, but by one antivirus as Locky. One sample downloaded earlier as Locky was labelled as Locky and TeslaCrypt by different antiviruses[15] too. These are therefore mislabelled and are probably of the same family.

The group of 23 domains and 9 domains (in the 4th and 9th rows of Table 5.10), show domains generated by a DGA, as they have similarities and patterns. From the 23 domains, 12 start with *pop* and end with *ru* and the rest may be compromised sites. From the 9 domains, the sub strings of what looks like random letters and numbers may have been generated automatically.

[15]https://www.virustotal.com/en/file/00b56667d794895fe8342f4d616e303cea222b88d2af8f0042b8e264a759e975/analysis/

Table 5.10: Domains queried by multiple families

Group	Domains	No. of Domains
Locky + CryptoWall + DMA Locker + VirLock	cdp1.public-trust.com *	1
CryptoWall + FileLocker + TorrentLocker	myexternalip.com *	1
CBT Locker + FileLocker + TorrentLocker	ipecho.net *	1
CryptoWall + TeslaCrypt	pop.w8start.ru, pop.consultinginc.ru, pop.xonpqigw.ru, pop.vfukgsuopav.ru, pop.natntbuo.ru, pop.thelove740.ru, pop.connect4.ru, pop.tinyupdates.ru, pop.vindustry.ru, pop.eebgghfs.ru, pop.jwzuyjyk.ru, pop.itfutureclub.ru, classemgmt.testbada.com, ebookstoreforyou.com, resumosdenovela.net, pop.qlmkxqlx.com, shampooherbal.com, exaltation.info, masterlegue.com, commonsenseprotection.com, primasentrausaha.com, tradinbow.com, mkis.org,	23
CryptoWall + FileLocker	ip-addr.es *	1
FileLocker + TeslaCrypt	en.wikipedia.org *, www.torproject.org *	2
FileLocker + TorrentLocker	wtfismyip.com *	1
FileLocker + Locky	pulseaudio.duckdns.org	1
Locky + TeslaCrypt	marketathart.com, esbook.com, joshsawyerdesign.com, emmy2015.com, prodocument.co.uk, nlhomegarden.com, kkr4hbwdklf234bfl84uoqleflqwrfqwuelfh.brazabaya.com, a64gfdsjhb4htbiwaysbdvukyft5q.zobodine.at, 974gfb-jhb23hbfkyfaby3byqlyuebvly5q254y.mendilobo.com	9

TTL analysis

Over 80 DNS TTL values were found when observing the 903 domains and do not show much evidence of fast flux domains. Fast flux domains are domains that are mapped to a changing set of IP addresses (Ranjan, 2012). The Top 10 TTL values are shown in Table 5.11. Most DNS TTLs that are used are a multiple of 100 or multiples of whole minutes van Zyl *et al.* (2015) and TTLs such as 300, 60, 3600, 600, 1800 and 7200 are popular choices. Taking a look at Table 5.11, the top eight TTLs are multiples of 100 and do not seen to be unusual. This could be because many samples were using compromised websites to host binaries. The ninth and tenth ranked TTLs, which only had either four or three unique domains associated with them, show unusual TTLs. The TTLs 2560, 3601, 276 and 2334 are used for the sites, `xsso.sisttwtdogpyjlcc.info`, `sso.anbtr.com`, `sisttwtdogpyjlcc.info`, `jkrobnqpnnrrysuu.info`, `hknsjsu.org`, `cam-chat-teufel.com`, `ebookstoreforyou.com` (Seen in Table 5.10), `www.chernobylair.com`, `kovgynnthj.info`, `njtyltqfijualtdwg.org`, `hrqxlatbck.click`, `mrytkraiukdge.pl`, `xhwuqxembmwiylg.pl` and `ybpgpqhjvwuajh.pl`. Since the domains consist of seemingly random letters, it could be assumed that they have been generated by a DGA (Stalmans and Irwin, 2011). Since these domains are either not up, or do not contain a web application, it can be assumed that unusual TTL values are often associated with malware.

Table 5.11: Top TTL values for domain names

Rank	TTL(s)	Time	Domains	%
1	900	15 mins	323	35.77
2	3600	1 hour	143	15.84
3	10800	3 hours	81	8.97
4	600	10 mins	50	5.54
5	300	5 mins	20	2.21
6	7200	2 hours	8	0.89
7	14400	4 hours	7	0.78
8	1200	20 mins	5	0.55
9	5400, 60, 2560, 3601	1.5 hours, 1 min, 42.67 mins, 60.01 mins	4	0.44
10	276, 2334	4.6 mins, 38.9 mins	3	0.33

This is again demonstrated when, 70 TTLs had only one domain associated with each of them. They were all unusual TTL numbers like, 6869, 3224, 411, 2471, 598, 5075, 414,

415 with domains like salesandmarketing101.net, pop.thelove740.ru, conspec.us, pop.tinyupdates.ru, shirongfeng.cn, naturstein-schubert.de, tmfilms.net and saludaonline.com. By manual inspection, it was found that these domains were either not up, broken, or a compromised site.

However, many other domains that were not compromised sites, such as mdjplmqlup.org, jkepjxidrvmlwbxsw.tw, ucaqrqurwesrtruxo.tw, gfinvxx.biz, axeftvixuw.baggir.org, cdpmhahxejfbmhrni.me and exuvezywody.baggir.org had a TTL of 900. These domains appear to be generated from DGA. So unusual TTLs can usually indicate something suspicious, and a normal looking TTL could indicate either.

5.3.5 HTTP

Out of the 386 samples that generated traffic, 299 used the HTTP protocol. In total there were 903 HTTP requests seen (either GET or POST) and 897 HTTP GET or POST Indicators created and distributed between the 299 samples.

GET

There were 330 unique GET requests, with 28 unique hosts observed. Table 5.12 shows the hosts queried by more than one sample and the corresponding ransomware families. The top ranked host, cdp1.public-trust.com, is also the most queried domain from Table 5.10 in Section 5.3.4. The domains, ipecho.net, ip-addr.es and www.edge-institut.org are also in the Top 20 most queried domains from Table 5.10, which makes sense. The hosts, ipecho.net and ip-addr.es are also in Table 5.10, that shows domains queried by multiple ransomware families.

The host, xsso.sisttwtdogpyjlcc.info, is in both the Top 20 hosts from HTTP GET and HTTP POST from Section 5.3.5. The host, sso.anbtr.com is mentioned previously in Section 5.3.4 because it had a strange TTL of 3601. Both of these domains were used by Locky samples and are associated with an interesting bunch of URIs, which will be discussed later on.

The hosts, grosirkecantikan.com, webligheven.su, and www.minteee.com have not appeared in previous tables, and are all associated with TeslaCrypt. The host, grosirkeca ntikan.com, seems to be a compromised site and the other two were not up. The host,

`marketathart.com` (compromised site), was seen in Table 5.10 showing domains queried by more than one malware family.

There were 17 hosts only queried by one sample each and are not shown in Table 5.12. These hosts consist of compromised sites from mainly Bahrain, and some from the Soviet Union and Indonesia. Sample 127[16] sent an HTTP GET request to 11 of these 17 domains. An interesting case is the host, `nssdc.gsfc.nasa.gov`, in which a malware sample requested a benign .txt from the NASA site. It is unclear as to why the sample did this.

Table 5.12: Hosts from more than one sample

Rank	Host	No. of samples	Families
1	cdp1.public-trust.com	159	Locky, CryptoWall, DMALocker, VirLock
2	p3nlhclust404.shr.prod.phx3.secureserver.net	11	TeslaCrypt
3	ipecho.net	11	CTBLocker, TorrentLocker, FileLocker
4	ip-addr.es	8	CryptoWall, FileLocker
5	xsso.sisttwtdogpyjlcc.info	6	Locky
6	sso.anbtr.com	6	Locky
7	www.edge-institut.org	4	TeslaCrypt
8	grosirkecantikan.com	3	TeslaCrypt
9	webligheven.su	3	TeslaCrypt
10	www.minteee.com	2	TeslaCrypt
11	marketathart.com	2	Locky, TeslaCrypt

There were 70 unique URIs (62 of which were not used by multiple samples). The 11 URIs that were observed in multiple samples are shown in Table 5.12, along with the corresponding malware families.

[16] https://www.virustotal.com/en/file/2cd9366af8bb2bc55f16badf70902386d863d6c6517c8ebbd279376f7af0b525/analysis/

Table 5.13: Top 20 URIs and the number of samples that used it

Rank	URI	Samples	Families
1	/CRL/Omniroot2025.crl	159	Locky, CryptoWall, DMALocker, VirLock
2	/	62	CryptoWall, VirLock, TeslaCrypt
3	/SharedContent/redirect_0.html	11	TeslaCrypt
4	/plain	11	CTBLocker, FileLocker
5	/domain/sisttwtdogpyjlcc.info	6	Locky
6	/cgi-sys/suspendedpage.cgi	5	TeslaCrypt, Locky
7	/wp-content/themes/bstr.php	4	TeslaCrypt
8	/index.php	2	TeslaCrypt, CryptoWall

Since there were 330 unique GET requests, many of the URIs were used in combination with multiple domain names. This trait was previously observed in the Dridex evaluation in Section 5.2.3, where one sample was requesting for `k1.exe` from multiple hosts. This indicates that a number of different web applications are hosting the same file, or have a similar structure.

The top ranked URI, `/CRL/Omniroot2025.crl`, in Table 5.13, is a clean URI and is used along with the domain `cdp1.public-trust.com`[17], which is controlled by Verizon, and houses Certificate Revocation Lists (CRLs). It is unclear as to why this domain and URI did not appear in the baseline traffic or why it was queried while some malware samples were running. It may have to do with Microsoft services and warrants future exploration.

The URI, `/wp-content/themes/bstr.php`, which was used by four TeslaCrypt samples, is from a WordPress site and the sample querying it received an HTTP `301 Moved Permanently` reply. The HTTP payload that the four TeslaCrypt samples were sending to the previously mentioned URI was also sent to three other hosts, and received `404 Not Found`. Since the URIs were related to compromised WordPress sites, it is likely that the site had been notified of the issue and removed the malicious content.

There were 4 other WordPress URIs observed and were similar to `/wp-content/plugins/`

[17]https://www.virustotal.com/en/url/1d596694a22962ef19dfc2b8ca4b1d571206d7ced6ea5e4111d1c20361a468a3/analysis/

`wp-db-backup-made/Ve4wgx.php?t=wxw1mhu8l7p` and were used by one CryptoWall sample. WP-DB-Backup is plugin that can be exploited[18][19], which reinforced the fact that many of the IOCs are compromised sites.

The second ranked URI, does not provide useful information. The URI, `/domain/sisttwtdogpyjlcc.info`, was used with the Host in Table 5.12, `xsso.sisttwtdogpyjlcc.info`, and would be a good IOC.

The URI, `/cgi-sys/suspendedpage.cgi`, is shown by popular website management program called cPanel for reasons that include violating Terms and Conditions, exceeding the allocated bandwidth and the site owner not paying for the hosting service (Segura, 2015). According to Segura (2015), a fair amount of sites display this page that were previously used to distribute malware. Segura (2015) also shows an example of where malicious code had been injected into the suspended page that has an `iframe` that points to an exploit kit. This URI does not make a good IOC.

The URI, `/plain` from Table 5.12 was observed exclusively with the host ipecho.net, from Table 5.13. When browsing to `ipecho.net/plain`, no html is received, only a single IP address of the host. This is a special page on ipecho.net, as the plain domain returns the host IP with HTML code. The URI `SharedContent/redirect_0.html`, and the host `p3nlhclust404.shr.prod.phx3.secureserver.net` from Table 5.12, were queried together. This IOC is however not useful, as it is not malicious.

A number of similar URIs were observed and look like they were generated with an algorithm. Coincidentally they were all created by one sample, Sample 127 (CryptoWall) mentioned previously in this Section. Some of the URIs are listed below:

- `/index.php?o=sbr0m7oti2pcus`
- `/index.php?k=sbr0m7oti2pcus`
- `/index.php?z=sbr0m7oti2pcus`
- `/index.php?n=9fg2ucvtkrb9x`
- `/index.php?j=9fg2ucvtkrb9x`
- `/index.php?l=lfte0iec41q`

[18]http://www.securityfocus.com/archive/1/534073
[19]https://thehackerblog.com/tag/wp-db-backup-exploit/

- `/index.php?i=lfte0iec41q`
- `/index.php?f=5krabf63zqz`
- `/index.php?h=5krabf63zqz`

Other interesting URIs that also look they have been generated by an algorithm are listed below. The were generated by numerous Locky samples and resemble a MD5 hash. These URIs would make better IOCs than URIs such as `/index.php` or `/submit.php`, as they are not commonly seen and would not trigger excessively if made into IDS or firewall rules.

- `/d6aad9ab53589d342d26effa764b1b07`
- `/8718da51c3a39609726108aad54f6ef9`
- `/bfcb23feab67b80b5780f68f2d6be267`
- `/da631a33a1de51b8cac9358fdb5363aa`
- `/b06dbcb643bc039c2e0055cfe3b68b9c`

Only one HTTP GET request had an Accepted-Language, which was `en-us,en;q=0.5`. Sometimes malware does not respect the locale system settings for language and specifies other languages like Chinese (zh) or Russian (ru) (Rossow *et al.*, 2011). These ransomware samples, except for one did not need to utilise the Accepted-Language header field.

Two packet requests were found to have an Accepted-Encoding field set and they are shown in Table 5.14. The samples in Table 5.14 were both CryptoWall samples and the one URI, `/index/amobi.exe,` was the only URI to contain the '.exe' string at the end.

Table 5.14: Accepted Encoding and sample ID number

Accepted-Encoding	Sample ID	URI
identity, *;q=0	172	/index/amobi.exe
gzip,deflate	166	/redir?o1=SHIM_NOVERSION_FOUND&version=(null)&processName=M.exe&platform=0009&osver=5&isServer=0&shimver=4.0.30319.34209

POST

There were 573 unique POST requests, with 147 unique hosts. HTTP POST requests were a lot more popular with the samples than HTTP GET. Table 5.15 shows the Top 10 hosts observed from the POST traffic. There were 74 hosts used by multiple samples and 73 hosts that were not observed with multiple samples. The hosts in Table 5.15, all appear in the top 20 domains queried in Table 5.8 and were all queried by TeslaCrypt. Other ransomware families that used POST are CryptoWall, Locky and FileLocker. Locky was the only ransomware where hosts were seen as IP addresses and not domains, which could mean that these IPs were hardcoded as no domain name resolution had to be performed.

The ten hosts, seen in Table 5.15, are either not registered or a compromised site. The host, `naturstein-schubert.de`, is a compromised site and was mentioned in Section 5.3.4 for having an unusual TTL value.

Table 5.15: Top 20 Hosts

Rank	Host	No. of files	Families
1	casasembargada.com	20	TeslaCrypt
2	naturstein-schubert.de	20	TeslaCrypt
3	csskol.org	20	TeslaCrypt
4	forms.net.in	20	TeslaCrypt
5	newculturemediablog.com	11	TeslaCrypt
6	saludaonline.com	11	TeslaCrypt
7	goktugyeli.com	11	TeslaCrypt
8	iqinternal.com	9	TeslaCrypt
9	conspec.us	9	TeslaCrypt
10	tmfilms.net	9	TeslaCrypt

The types of content being sent via POST were one variable with hex, or one variable with 640 hex characters or encrypted looking data.

There were 275 unique URIs observed and since there were 573 POST requests and 147 hosts, it is evident that for one host address, more than one URI was queried for it in some cases.

Taking a look at Table 5.16, it is clear to see that many of the URI are WordPress related. As a total there were 97 unique WordPress related URIs. Although five of the URIs in

Table 5.16 are generated by TeslaCrypt samples, the majority of the WordPress URIs were generated by one CryptoWall sample (mentioned previously in the above Section 5.3.5, Sample 127). The URIs contain numerous references to different WordPress plugins, such as Slider Revolution Responsive, Jetpack, Clef Two-Factor Authentication, Visual Composer and the WordPress Multilingual Plugin. This shows that different types of WordPress plugins on sites are targeted and exploited in order to compromise them. In Section 5.3.5, there were five WordPress related HTTP GET URIs, again showing how compromised sites are used for malware hosting.

Taking a look at the files being requested in Table 5.16, `stringfile.php` and `wstr.php` appear multiple times. As eluded to previously, the URL `/submit.php` would not make a good indicator, as it would trigger excessively if made into an IDS or firewall rule.

Table 5.16: Top 20 URIs

Rank	URI	No. of files	Families	Replies
1	/wp-content/plugins/js_composer/assets/lib/font-awesome/src/assets/font-awesome/fonts/stringfile.php	21	TeslaCrypt	404 Not Found
2	/modules/mod_cmscore/stringfile.php	21	TeslaCrypt	404 Not Found
3	/wp-content/plugins/formcraft/php/swift/lib/classes/Swift/Mime/HeaderEncoder/stringfile.php	21	TeslaCrypt	404 Not Found
4	/cgi-bin/stringfile.php	21	TeslaCrypt	500 Internal Server Error
5	/submit.php	15	Locky	404 Not Found, 200 OK
6	/wstr.php	14	TeslaCrypt	404 Not Found, 500 Internal Server Error, 200 OK
7	/wp-includes/fonts/wstr.php	11	TeslaCrypt	302 Found
8	/pmtsys/fonts/wstr.php	9	TeslaCrypt	404 Not Found
9	/wp-content/plugins/nextgen-galleryOLD/products/photocrati_nextgen/modules/i18n/wstr.php	9	TeslaCrypt	200 OK
10	/wp-content/plugins/binary.php	9	TeslaCrypt	Nothing

The Content-Type header shows which type of web content will be received by the sam-

ples. Table 5.17 shows that most samples that used HTTP POST, used forms to submit data to a server. All the observed Content-Types are displayed in Table 5.17. The top Content-Type was `application/x-www-form-urlencoded` and was used by a number of ransomware families. The contents of the bodies of the HTTP requests mainly consisted of a single variable set to a 640 hex character string, which were sent to multiple hosts.

In Rossow *et al.* (2011), it was found that `application/octet-stream` was the most used Content-Type to download PE Binaries. Since it was observed once in this case study, it could mean that most of the samples did not download PE binaries because the samples were already secondary binaries.

Table 5.17: HTTP Content-Types

Content-Type	No. of Samples	Families
application/x-www-form-urlencoded	87	Locky, CryptoWall, FileLocker, TeslaCrypt
application/x-www-form-urlencoded; charset=UTF-8	1	FileLocker
application/octet-stream	1	CryptoWall
multipart/form-data, boundary=7E0041D0B200003	1	FileLocker

One sample made use of the Accepted-Language header field and it was a TeslaCrypt one. The value in the field is shown in Table 5.18 and is unusual.

Table 5.18: Accepted-Language

Accepted-Language	Family
\xe0u+\x02, Y\xb9}\xa5 \xba, py+\x02@\x04^\x02<, \x90(^\x02h)_\x028\x01+\x02E b\x0f	TeslaCrypt

Three samples used the Accepted-Encoding field and they are shown in Table 5.19. The `\xe0u+\x02` encoding type is unusual and could be used as a TeslaCrypt IOC, and possibly for an IDS rule.

Table 5.19: Accepted-Encoding

Accepted-Encoding	Family
\xe0u+\x02	TeslaCrypt
identity *;q=0	CryptoWall
identity	FileLocker

In one Locky sample, four CryptoWall samples, two FileLocker samples and 39 TeslaCrypt samples, there were POST requests with encoded text, which changed every time the sample was run. An example of the POST request for a sample is shown in Figure 5.2. They have one form item called "data". Each sample sent the same form data to a number of hosts via a POST request.

```
POST /wp-includes/images/bstr.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0;
Touch; rv:11.0) like Gecko
Host: adrianadoms.com
Content-Length: 645
Cache-Control: no-cache

data=C5578C52C52CD45901891EE1810D73EB0437978009A50D99F6087E27EF
A6F8CCEB64180025C49F5675A5470F326CBCF2E0889D8AD8E30274FAB71B9F4
56544B375DB41C6BA6FD1712AB659ED100A6329506E1C2DB1FD85680F37C441
1BD8F970659A4E90EFCBBEC791FCF5D4575B3036B5B487C074DAF8522FFAD40
BDE511D1ADF73B8C27634DE45124832CC5064D488D477ECC266FD73E4AA0247
A14267DAB80A249A7E92A4C943E648EEE9F2C8D77BEBCF3E23700F83E98A94C
59C44A5BA026480A278D6ED099744E31EFD77E662C5302E0AB73EBDA008FA6B
641F57A0CC12F132075DA6340EF8F4AF350E554B061F47DFF61C7659DD0E177
623D0FDD603760C65BD2FDDDAB10B54557EDFDB6AF900C848F8A78E69F6001E
F3D6ED38DD54D098400EAC52A315D4725972BE308D2B33C9426EB420FE35EF7
74BE68C58D064A9
```

Figure 5.2: HTTP Post request for a TeslaCrypt sample

5.3.6 UDP

Other than recorded DNS traffic (53/UDP), there were two UDP connections, both to the same IP address (195.22.26.248[20]) using the destination port 8670. One was by a TeslaCrypt sample and the other by a CryptoWall sample. Both sample's UDP packets got an ICMP packet with a `Destination unreachable` message. As discussed in Section 5.3.4, there may have been a mislabelled sample from one of the antiviruses so these requests could be from the same family. The two samples get the IP address from a DNS response to the same address and immediately send a UDP packet to the address.

[20]https://www.virustotal.com/en/ip-address/195.22.26.248/information/

5.3.7 Other Traffic

Since the reporting module creates ICMP, FTP and SSH Indicators, traffic from those protocols was analysed too. However, it was found that none of the ransomware samples used ICMP, FTP or SSH.

5.3.8 Hard coded IP addresses

Since a regular expression was used to find IP address strings from Cuckoo's 'Strings' processing module[21], there were many strings that turned out not to be a an IP address, but the version number of something in the sample. The strings such as 6.0.0.0, 1.0.0.0 and 4.0.0.0 were identified as IPs and technically they could be, but not in this context. Most of the false positives had a string in front of it like, "`ProductVersion 7.0.40.20`".

There were supposedly 150 hard coded IP addresses, but on closer observation, most of them were false positives. By manual inspection, it was found that there were seven actual hard coded IP addresses.

The samples that did contain IP addresses were, a Locky sample which had three hard coded IP addresses in the string "`92.63.87.134,84.19.170.249,91.200.14.73`". The same sample also had HTTP header content "`Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET4.0C; .NET4.0E) HTTP/1.1 Content-type: application/x-www-form-urlencoded clicksu /submit.php`". However, none of the packets from this sample included the URI, `submit.php` or the IP addresses. Another Locky sample had the IP address, `86.104.134.144` hard coded.

A FileLocker sample had one hard coded IP address appearing in three different strings. For example: "`http://202.181.194.227/cryptowall/index.html`" was one of the strings containing the address.

Lastly, one Chimera sample had two hard coded IPs, `95.165.168.168` and `158.222.211.81`, although none of these were contacted. It also had a number of hard coded URLs such as `http://bot.whatismyipaddress.com/`.

[21] https://github.com/cuckoosandbox/cuckoo/blob/master/modules/processing/strings.py

5.3.9 STIX indicators

The Indicators shown in this section are generated from the analysis of sample 127, as it generated the largest variety of Indicators.

Figure 5.3 shows the STIX and CybOX representation of an outgoing established TCP connection. Looking at the line numbers, lines 5611 and 5612 in Figure 5.3 and lines 1237 and 1238 in Figure 5.4, show STIX contextual information using about the CybOX Observables contained inside of the STIX indicator. Line 5618 in Figure 5.3 and line 1244 in Figure 5.4, contains a string with the TCP state, 'ESTABLISHED' or 'SYN_SENT', which is the main difference between the two CybOX Observables. Figure 5.3 and 5.4 also contain an IP address, port, protocols and data of the creation of the Indicator. If indicators are created immediately after dynamic analysis, the stored date and time can be used to keep track of when a host was up or not. Other than the information stored in the XML, this indicator also shows that the host address was up and responding to connections around about the time stored in line 5633 in Figure 5.3.

```
5610        <stix:Indicator id="example:indicator-7f8bf3b7-2f69-4b48-82a9-d6d68e855bb0" timestamp="2016-04-26T16:58:28.541957+00:00"
    xsi:type='indicator:IndicatorType'>
5611            <indicator:Title>TCP Connection Established</indicator:Title>
5612            <indicator:Description>An indicator containing information about a successful TCP hand shake</indicator:Description>
5613            <indicator:Observable id="example:Observable-5609f262-a65c-45a2-9cfe-24f5b7efbf1a">
5614                <cybox:Object id="example:NetworkConnection-27d0375e-3bf9-4447-baf4-a6bc3d339524">
5615                    <cybox:Properties xsi:type="NetworkConnectionObj:NetworkConnectionObjectType">
5616                        <NetworkConnectionObj:Layer3_Protocol>IPv4</NetworkConnectionObj:Layer3_Protocol>
5617                        <NetworkConnectionObj:Layer4_Protocol>TCP</NetworkConnectionObj:Layer4_Protocol>
5618                        <NetworkConnectionObj:Source_TCP_State>ESTABLISHED</NetworkConnectionObj:Source_TCP_State>
5619                        <NetworkConnectionObj:Destination_Socket_Address xsi:type="SocketAddressObj:SocketAddressObjectType">
5620                            <SocketAddressObj:IP_Address xsi:type="AddressObj:AddressObjectType">
5621                                <AddressObj:Address_Value>212.91.26.155</AddressObj:Address_Value>
5622                            </SocketAddressObj:IP_Address>
5623                            <SocketAddressObj:Port xsi:type="PortObj:PortObjectType">
5624                                <PortObj:Port_Value>80</PortObj:Port_Value>
5625                                <PortObj:Layer4_Protocol>TCP</PortObj:Layer4_Protocol>
5626                            </SocketAddressObj:Port>
5627                        </NetworkConnectionObj:Destination_Socket_Address>
5628                    </cybox:Properties>
5629                </cybox:Object>
5630            </indicator:Observable>
5631            <indicator:Producer>
5632                <stixCommon:Time>
5633                    <cyboxCommon:Produced_Time>2016-04-26T16:58:28.542225+00:00</cyboxCommon:Produced_Time>
5634                </stixCommon:Time>
5635            </indicator:Producer>
5636        </stix:Indicator>
5637        <stix:Indicator id="example:indicator-bb2bf82d-c138-4f53-9adb-3631f738cc27" timestamp="2016-04-26T16:58:28.544003+00:00"
    xsi:type='indicator:IndicatorType'>
```

Figure 5.3: Example of a TCP Connection Established Indicator

```
1236        <stix:Indicator id="example:indicator-381fd63e-f8ea-449c-9cff-fcb39ce1cb96" timestamp="2016-04-26T16:58:20.180363+00:00"
    xsi:type='indicator:IndicatorType'>
1237            <indicator:Title>TCP Connection Fail</indicator:Title>
1238            <indicator:Description>An indicator containing information about a failed TCP hand shake</indicator:Description>
1239            <indicator:Observable id="example:Observable-f9d6a6e4-3a5e-4850-83a2-98799c78b967">
1240                <cybox:Object id="example:NetworkConnection-344cc56f-1228-4cca-becb-8b9252fa18c3">
1241                    <cybox:Properties xsi:type="NetworkConnectionObj:NetworkConnectionObjectType">
1242                        <NetworkConnectionObj:Layer3_Protocol>IPv4</NetworkConnectionObj:Layer3_Protocol>
1243                        <NetworkConnectionObj:Layer4_Protocol>TCP</NetworkConnectionObj:Layer4_Protocol>
1244                        <NetworkConnectionObj:Source_TCP_State>SYN_SENT</NetworkConnectionObj:Source_TCP_State>
1245                        <NetworkConnectionObj:Destination_Socket_Address xsi:type="SocketAddressObj:SocketAddressObjectType">
1246                            <SocketAddressObj:IP_Address xsi:type="AddressObj:AddressObjectType">
1247                                <AddressObj:Address_Value>184.106.112.172</AddressObj:Address_Value>
1248                            </SocketAddressObj:IP_Address>
1249                            <SocketAddressObj:Port xsi:type="PortObj:PortObjectType">
1250                                <PortObj:Port_Value>80</PortObj:Port_Value>
1251                                <PortObj:Layer4_Protocol>TCP</PortObj:Layer4_Protocol>
1252                            </SocketAddressObj:Port>
1253                        </NetworkConnectionObj:Destination_Socket_Address>
1254                    </cybox:Properties>
1255                </cybox:Object>
1256            </indicator:Observable>
1257            <indicator:Producer>
1258                <stixCommon:Time>
1259                    <cyboxCommon:Produced_Time>2016-04-26T16:58:20.180649+00:00</cyboxCommon:Produced_Time>
1260                </stixCommon:Time>
1261            </indicator:Producer>
1262        </stix:Indicator>
1263        <stix:Indicator id='example:indicator-bfdcb58c-3bd7-4750-87c9-4a49a2899871" timestamp="2016-04-26T16:58:20.182601+00:00"
    xsi:type='indicator:IndicatorType'>
```

Figure 5.4: Example of a TCP Connection Failed Indicator

Figure 5.5 shows the STIX and CybOX content created for a DNS Query Indicator. Lines 17811 and 17812 contain contextual information about the Indicator being a DNS Request. Lines 17816 to 17818 show the protocols used at the different layers. The IP address of the DNS server, if contained in line 17821 and the port used is in line 17824. The domain queried and the record type are shown in line 17832 and 17834.

```
17809        <stix:Indicator id="example:indicator-66144b3d-5dae-4352-9272-03a074267882" timestamp="2016-04-26T16:58:21.343674+00:00"
17810 xsi:type='indicator:IndicatorType'>
17811            <indicator:Title>DNS Request</indicator:Title>
17812            <indicator:Description>An indicator containing information about a DNS Request</indicator:Description>
17813            <indicator:Observable id="example:Observable-6a770859-5248-4ef0-bf31-04d7b23b056e">
17814                <cybox:Object id="example:NetworkConnection-5956251a-e211-475d-b682-253cb1b8f45e">
17815                    <cybox:Properties xsi:type="NetworkConnectionObj:NetworkConnectionObjectType">
17816                        <NetworkConnectionObj:Layer3_Protocol>IPv4</NetworkConnectionObj:Layer3_Protocol>
17817                        <NetworkConnectionObj:Layer4_Protocol>UDP</NetworkConnectionObj:Layer4_Protocol>
17818                        <NetworkConnectionObj:Layer7_Protocol>DNS</NetworkConnectionObj:Layer7_Protocol>
17819                        <NetworkConnectionObj:Destination_Socket_Address xsi:type="SocketAddressObj:SocketAddressObjectType">
17820                            <SocketAddressObj:IP_Address xsi:type="AddressObj:AddressObjectType">
17821                                <AddressObj:Address_Value>146.231.129.97</AddressObj:Address_Value>
17822                            </SocketAddressObj:IP_Address>
17823                            <SocketAddressObj:Port xsi:type="PortObj:PortObjectType">
17824                                <PortObj:Port_Value>53</PortObj:Port_Value>
17825                                <PortObj:Layer4_Protocol>UDP</PortObj:Layer4_Protocol>
17826                            </SocketAddressObj:Port>
17827                        </NetworkConnectionObj:Destination_Socket_Address>
17828                        <NetworkConnectionObj:Layer7_Connections>
17829                            <NetworkConnectionObj:DNS_Query xsi:type="DNSQueryObj:DNSQueryObjectType">
17830                                <DNSQueryObj:Question>
17831                                    <DNSQueryObj:QName xsi:type="URIObj:URIObjectType">
17832                                        <URIObj:Value>netoria.mydevil.net</URIObj:Value>
17833                                    </DNSQueryObj:QName>
17834                                    <DNSQueryObj:QType>AAAA</DNSQueryObj:QType>
17835                                </DNSQueryObj:Question>
17836                            </NetworkConnectionObj:DNS_Query>
17837                        </NetworkConnectionObj:Layer7_Connections>
17838                    </cybox:Properties>
17839                </cybox:Object>
17840            </indicator:Observable>
17841            <indicator:Producer>
17842                <stixCommon:Time>
17843                    <cyboxCommon:Produced_Time>2016-04-26T16:58:21.345358+00:00</cyboxCommon:Produced_Time>
                </stixCommon:Time>
17844            </indicator:Producer>
17845        </stix:Indicator>
17846        <stix:Indicator id="example:indicator-8eb55153-f745-48b9-94ef-bb3afaf6ecea" timestamp="2016-04-26T16:58:21.347168+00:00"
17847 xsi:type='indicator:IndicatorType'>
```

Figure 5.5: Example of a DNS Query Indicator

Figure 5.6 shows a STIX indicator for a HTTP GET request to the host `www.diamondglassbh.com`. Lines 7248 and 7249 show the contextual information, saying that this is an

HTTP Request Indicator. Lines 7253 to 7255 show the protocols observed and line 7261 indicates that the HTTP method was GET. The URI is shown on line 7262 and the corresponding host is on line 7272. Other HTTP header values are that were found in the HTTP header are stored in the CybOX Observable too. This Indicator provides a comprehensive set of information that can be used in the creation of defence mechanisms on a network level.

```
7247        <stix:Indicator id="example:indicator-f55fab78-614b-42f8-bee4-9328d5744d97" timestamp="2016-04-26T16:58:20.789811+00:00"
    xsi:type='indicator:IndicatorType'>
7248            <indicator:Title>HTTP request</indicator:Title>
7249            <indicator:Description>An indicator containing information about a HTTP request</indicator:Description>
7250            <indicator:Observable id="example:Observable-f2879d99-7e48-4861-9e39-a39788d4914f">
7251                <cybox:Object id="example:NetworkConnection-d14a5ff7-f535-4573-ba8f-bfd8a87ff5b8">
7252                    <cybox:Properties xsi:type="NetworkConnectionObj:NetworkConnectionObjectType">
7253                        <NetworkConnectionObj:Layer3_Protocol>IPv4</NetworkConnectionObj:Layer3_Protocol>
7254                        <NetworkConnectionObj:Layer4_Protocol>TCP</NetworkConnectionObj:Layer4_Protocol>
7255                        <NetworkConnectionObj:Layer7_Protocol>HTTP</NetworkConnectionObj:Layer7_Protocol>
7256                        <NetworkConnectionObj:Layer7_Connections>
7257                            <NetworkConnectionObj:HTTP_Session xsi:type="HTTPSessionObj:HTTPSessionObjectType">
7258                                <HTTPSessionObj:HTTP_Request_Response>
7259                                    <HTTPSessionObj:HTTP_Client_Request>
7260                                        <HTTPSessionObj:HTTP_Request_Line>
7261                                            <HTTPSessionObj:HTTP_Method>GET</HTTPSessionObj:HTTP_Method>
7262                                            <HTTPSessionObj:Value>/index.php?i=lfte0iec41q</HTTPSessionObj:Value>
7263                                            <HTTPSessionObj:Version>HTTP/1.1</HTTPSessionObj:Version>
7264                                        </HTTPSessionObj:HTTP_Request_Line>
7265                                        <HTTPSessionObj:HTTP_Request_Header>
7266                                            <HTTPSessionObj:Parsed_Header>
7267                                                <HTTPSessionObj:Accept>*/*</HTTPSessionObj:Accept>
7268                                                <HTTPSessionObj:Cache_Control>no-cache</HTTPSessionObj:Cache_Control>
7269                                                <HTTPSessionObj:Connection>Close</HTTPSessionObj:Connection>
7270                                                <HTTPSessionObj:Host>
7271                                                    <HTTPSessionObj:Domain_Name xsi:type="URIObj:URIObjectType">
7272                                                        <URIObj:Value>www.diamondglassbh.com</URIObj:Value>
7273                                                    </HTTPSessionObj:Domain_Name>
7274                                                    <HTTPSessionObj:Port xsi:type='PortObj:PortObjectType'>
7275                                                        <PortObj:Port_Value>80</PortObj:Port_Value>
7276                                                    </HTTPSessionObj:Port>
7277                                                </HTTPSessionObj:Host>
7278                                            </HTTPSessionObj:Parsed_Header>
7279                                        </HTTPSessionObj:HTTP_Request_Header>
7280                                    </HTTPSessionObj:HTTP_Client_Request>
7281                                </HTTPSessionObj:HTTP_Request_Response>
7282                            </NetworkConnectionObj:HTTP_Session>
7283                        </NetworkConnectionObj:Layer7_Connections>
7284                    </cybox:Properties>
7285                </cybox:Object>
7286            </indicator:Observable>
7287            <indicator:Producer>
7288                <stixCommon:Time>
7289                    <cyboxCommon:Produced_Time>2016-04-26T16:58:20.790071+00:00</cyboxCommon:Produced_Time>
7290                </stixCommon:Time>
7291            </indicator:Producer>
7292        </stix:Indicator>
7293        <stix:Indicator id="example:indicator-0ad11a80-5504-4148-839e-4b9df60ee948" timestamp="2016-04-26T16:58:20.791935+00:00"
    xsi:type='indicator:IndicatorType'>
```

Figure 5.6: Example of a HTTP GET Indicator

A STIX indicator for a HTTP POST request is shown in Figure 5.7. As usual, two lines contain contextual information describing the type of Indicator and three lines, 13938 to 13940, containing the protocols used. Line 13946 shows the HTTP method, POST, which makes the Indicator different from the HTTP request in Figure 5.6, which was GET. Line 13947 contains the URI and line 13960 shows the corresponding host. The port used, 80, is on line 13963 and line 13952 to 13857 contain more HTTP header values, including a cookie containing a session ID. Again, this STIX indicator contains the information needed for creating IDS and firewall rules.

```
13932       <stix:Indicator id="example:indicator-e294bcbd-53b7-4e38-8592-1437f36a1bc6" timestamp="2016-04-26T16:58:21.137247+00:00"
    xsi:type='indicator:IndicatorType'>
13933           <indicator:Title>HTTP request</indicator:Title>
13934           <indicator:Description>An indicator containing information about a HTTP request</indicator:Description>
13935           <indicator:Observable id="example:Observable-f353584a-17f2-462b-9549-88348117bbba">
13936               <cybox:Object id="example:NetworkConnection-8dd1f5a3-f2e9-459a-9464-9e0855280e68">
13937                   <cybox:Properties xsi:type="NetworkConnectionObj:NetworkConnectionObjectType">
13938                       <NetworkConnectionObj:Layer3_Protocol>IPv4</NetworkConnectionObj:Layer3_Protocol>
13939                       <NetworkConnectionObj:Layer4_Protocol>TCP</NetworkConnectionObj:Layer4_Protocol>
13940                       <NetworkConnectionObj:Layer7_Protocol>HTTP</NetworkConnectionObj:Layer7_Protocol>
13941                       <NetworkConnectionObj:Layer7_Connections>
13942                           <NetworkConnectionObj:HTTP_Session xsi:type="HTTPSessionObj:HTTPSessionObjectType">
13943                               <HTTPSessionObj:HTTP_Request_Response>
13944                                   <HTTPSessionObj:HTTP_Client_Request>
13945                                       <HTTPSessionObj:HTTP_Request_Line>
13946                                           <HTTPSessionObj:HTTP_Method>POST</HTTPSessionObj:HTTP_Method>
13947                                           <HTTPSessionObj:Value>/wp-content/cache/db/000000/all/225/521/5.php?g=9fg2ucvtkrb9x</HTTPSessionObj:Value>
13948                                           <HTTPSessionObj:Version>HTTP/1.1</HTTPSessionObj:Version>
13949                                       </HTTPSessionObj:HTTP_Request_Line>
13950                                       <HTTPSessionObj:HTTP_Request_Header>
13951                                           <HTTPSessionObj:Parsed_Header>
13952                                               <HTTPSessionObj:Accept>*/*</HTTPSessionObj:Accept>
13953                                               <HTTPSessionObj:Cache_Control>no-cache</HTTPSessionObj:Cache_Control>
13954                                               <HTTPSessionObj:Connection>Close</HTTPSessionObj:Connection>
13955                                               <HTTPSessionObj:Cookie>qtrans_front_language=pb; PHPSESSID=euj6nsqsaphouckin2u9qi2hv1</HTTPSessionObj:Cookie>
13956                                               <HTTPSessionObj:Content_Length>104</HTTPSessionObj:Content_Length>
13957                                               <HTTPSessionObj:Content_Type>application/x-www-form-urlencoded</HTTPSessionObj:Content_Type>
13958                                               <HTTPSessionObj:Host>
13959                                                   <HTTPSessionObj:Domain_Name xsi:type="URIObj:URIObjectType">
13960                                                       <URIObj:Value>devolus.com.br</URIObj:Value>
13961                                                   </HTTPSessionObj:Domain_Name>
13962                                                   <HTTPSessionObj:Port xsi:type="PortObj:PortObjectType">
13963                                                       <PortObj:Port_Value>80</PortObj:Port_Value>
13964                                                   </HTTPSessionObj:Port>
13965                                               </HTTPSessionObj:Host>
13966                                           </HTTPSessionObj:Parsed_Header>
13967                                       </HTTPSessionObj:HTTP_Request_Header>
13968                                   </HTTPSessionObj:HTTP_Client_Request>
13969                               </HTTPSessionObj:HTTP_Request_Response>
13970                           </NetworkConnectionObj:HTTP_Session>
13971                       </NetworkConnectionObj:Layer7_Connections>
13972                   </cybox:Properties>
13973               </cybox:Object>
13974           </indicator:Observable>
13975           <indicator:Producer>
13976               <stixCommon:Time>
13977                   <cyboxCommon:Produced_Time>2016-04-26T16:58:21.137592+00:00</cyboxCommon:Produced_Time>
13978               </stixCommon:Time>
13979           </indicator:Producer>
13980       </stix:Indicator>
13981       <stix:Indicator id="example:indicator-91798aad-829f-417e-b02f-0db314029afd" timestamp="2016-04-26T16:58:21.140255+00:00"
```


Figure 5.7: Example of a HTTP POST Indicator

Since sample 127 did not create any purely UDP packets, one of the two UDP indicators from a different sample is shown in Figure 5.8. A UDP indicator contains less information than the DNS, HTTP and TCP indicators. The main values are the IP address on line 257 and the port on line 260. Since the IP and port are destination values, this in an outgoing UDP packet. Contextual information is on the lines 248 and 249, and the protocols, IP and UDP are stored on lines 253 and 254. The protocol, IP address and port can be used to create IDS and firewall rules.

```
247         <stix:Indicator id="example:indicator-6e7718a1-80d8-47b2-bfdb-57d8c3fbe76a" timestamp="2016-04-26T23:48:05.786480+00:00"
    xsi:type='indicator:IndicatorType'>
248             <indicator:Title>UDP connection</indicator:Title>
249             <indicator:Description>An indicator containing information about a UDP connection</indicator:Description>
250             <indicator:Observable id="example:Observable-13778266-a0a8-4a3f-a505-97e2ad0b737a">
251                 <cybox:Object id="example:NetworkConnection-a3463c13-1668-49ff-9147-08190bb58827">
252                     <cybox:Properties xsi:type="NetworkConnectionObj:NetworkConnectionObjectType">
253                         <NetworkConnectionObj:Layer3_Protocol>IPv4</NetworkConnectionObj:Layer3_Protocol>
254                         <NetworkConnectionObj:Layer4_Protocol>UDP</NetworkConnectionObj:Layer4_Protocol>
255                         <NetworkConnectionObj:Destination_Socket_Address xsi:type="SocketAddressObj:SocketAddressObjectType">
256                             <SocketAddressObj:IP_Address xsi:type="AddressObj:AddressObjectType">
257                                 <AddressObj:Address_Value>93.190.140.243</AddressObj:Address_Value>
258                             </SocketAddressObj:IP_Address>
259                             <SocketAddressObj:Port xsi:type="PortObj:PortObjectType">
260                                 <PortObj:Port_Value>867B</PortObj:Port_Value>
261                                 <PortObj:Layer4_Protocol>UDP</PortObj:Layer4_Protocol>
262                             </SocketAddressObj:Port>
263                         </NetworkConnectionObj:Destination_Socket_Address>
264                     </cybox:Properties>
265                 </cybox:Object>
266             </indicator:Observable>
267             <indicator:Producer>
268                 <stixCommon:Time>
269                     <cyboxCommon:Produced_Time>2016-04-26T23:48:05.786840+00:00</cyboxCommon:Produced_Time>
270                 </stixCommon:Time>
271             </indicator:Producer>
272         </stix:Indicator>
273         <stix:Indicator id="example:indicator-7c5936ad-8c0c-4389-9f0d-2d7cb81969e2" timestamp="2016-04-26T23:48:05.789172+00:00"
    xsi:type='indicator:IndicatorType'>
```


Figure 5.8: Example of a UDP Indicator

Lines 293 and 294 in Figure 5.9 show the contextual information about the indicator,

which is an IP address resolved from a DNS response to a request. This is a simple indicator as it just contains one IP address on line 298 and date of creation.

```
292         <stix:Indicator id="example:indicator-4a24b546-b972-44c8-946c-111029cb703a" timestamp="2016-04-26T16:58:28.036941+00:00"
    xsi:type='indicator:IndicatorType'>
293             <Indicator:Title>Suspicious IP address</Indicator:Title>
294             <Indicator:Description>An indicator containing a IPv4 address resolved from a suspicious domain</Indicator:Description>
295             <Indicator:Observable id="example:Observable-b1f4db9c-8462-477c-932b-8ceb7fc763bd">
296                 <cybox:Object id="example:Address-725e06ae-44e9-4f41-a999-6373393df7ca">
297                     <cybox:Properties xsi:type="AddressObj:AddressObjectType" category="ipv4-addr">
298                         <AddressObj:Address_Value>108.167.168.63</AddressObj:Address_Value>
299                     </cybox:Properties>
300                 </cybox:Object>
301             </Indicator:Observable>
302             <Indicator:Producer>
303                 <stixCommon:Time>
304                     <cyboxCommon:Produced_Time>2016-04-26T16:58:28.037268+00:00</cyboxCommon:Produced_Time>
305                 </stixCommon:Time>
306             </Indicator:Producer>
307         </stix:Indicator>
308         <stix:Indicator id="example:indicator-f229118b-fd4c-46bd-bfc2-6e418f34b5c5f" timestamp="2016-04-26T16:58:28.039193+00:00"
    xsi:type='indicator:IndicatorType'>
```

Figure 5.9: Example of an IP address resolved from a domain name Indicator

5.3.10 Summary

The section started off describing the setup used for analysis of 585 ransomware samples and then looked at an overview of the network traffic statistics. An in depth traffic analysis followed, which showed that there was clear evidence of DGAs, URI generation algorithms, compromised sites hosting malicious files, and sites that return the IP address of the infected host.

The last section took a look at each type of indicator generated and describes the important parts of the STIX XML snippets. A number of IOCs, like TCP, UDP, HTTP and DNS contain useful information that would fit well into IDS rules, like Snort rules and firewall rules like Iptables. However, since many of the domains are not malicious in themselves, these domains should be filtered. If they for example appear in the Alexa top one million websites list[22], or something similar, they would trigger many IDS alerts.

5.4 Traffic and Analysis Time

This is a brief evaluation that compares the network traffic from three virtual machines, after executing 325 malware samples, that were labelled on VirusTotal to produce certain types of traffic.

According to Egele *et al.* (2012), denying a sandbox Internet access usually results in incomplete behavioural analysis of the malware and in terms of collecting network information. There are many advantages of allowing Internet access, which include potentially

[22]http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

having access to files that malware downloads, and observing the behaviour of malware that needs Internet access to receive commands (Katsamakis, 2014). In this evaluation, Internet access will be allowed and three analysis times will be evaluated. One of times being the default value for the Cuckoo sandbox, which is 300 seconds (5 minutes). The evaluation tests if there is a significant difference in the type of traffic and number of indicators generated by the framework when samples are run for 5, 30 and 60 minutes.

Three Virtual Machines were used in this evaluation, with the operating systems used being Windows XP, 7 and 8.1. Each had the same programs installed, with different outdated versions for each OS. Each sample was run through the Cuckoo sandbox, with the custom modules analysing the traffic generated. A script then analysed the IOCs created and the PCAPs to gain network traffic information. These were later compared between the three VMs.

5.4.1 Network Statistics

Tables 5.20, 5.21 and 5.22 show the total number of indicators created for the 325 samples for each timing. The columns show the three different times and each row indicating the type of IOC created.

No indicators were created for the Windows XP machine as indicated in Table 5.20, which shows zeros for all types of IOCs. This may be because Windows XP is too old for the samples to execute in, even though there were 25 samples that were specified to run on the XP architecture. Windows XP is also outdated and not the best OS to test modern malware on and as the results indicate, none of the samples could execute successfully.

Table 5.20: Network Traffic totals for Windows XP VM with different analysis times

Protocol	5 mins	30 mins	1 hour
ICMP	0	0	0
UDP	0	0	0
TCP EST	0	0	0
TCP FAIL	0	0	0
DNS	0	0	0
FTP	0	0	0
SSH	0	0	0
HTTP	0	0	0

There were many indicators created for the 325 samples using the Windows 7 OS. The totals of which are shown in Table 5.21. No ICMP indicators were created for each of the three execution times. The samples did generate traffic within 5 minutes of execution and the number of indicators increased for each time. There was a 26110% increase in UDP indicators between 5 and 30 minutes, in which the majority of the packets were generated by the 25 samples that were labelled to generate suspicious UDP packets. This shows that these samples needed more than 5 minutes to generate their intended traffic. There was a 24.93% increase in UDP packets from 30 minutes to one hour, which indicates that the majority of packets were generated between 5 and 30 minutes.

The large number of DNS requests (4736 IOCs from the samples within the first 5 minutes) was due to DGAs and many domains not resolving, which resulted in more domains being generated. It is indicated that most of the DGA requests occurred in the first 5 minutes of execution, with a 19.82% increase then a 3.75% increase in indicators from 5 to 30 minutes and 30 to 60 minutes respectively. This indicates that most of the samples using DGA, stopped the DGA after 30 minutes.

The other types of indicators, except for SSH, had a steady increase in the number of IOCs. The majority of which occurred in the first 5 minutes of execution.

Table 5.21: Network Traffic totals for Windows 7 VM with different analysis times

Protocol	5 mins	30 mins	1 hour
ICMP	0	0	0
UDP	30	7863	9823
TCP EST	550	671	785
TCP FAIL	76	93	132
DNS	4736	5675	5888
FTP	49	58	64
SSH	42	73	49
HTTP	758	946	1027

Like the Windows 7 VM, there were no ICMP indicators created for the Windows 8.1 VM. The Windows 8.1 results are shown in Table 5.22. There was a 4720.63% increase in UDP IOCs opposed to the 26110% increase on the Windows 7 VM. This shows that the generation of UDP IOCs from these samples was more successful using Windows 7, than Windows 8.1.

There were also more established and failed TCP connections, DNS, FTP, SSH and HTTP IOCs generated by the Windows 7 VM than the Windows 8.1 VM. This again confirms that, with the assortment of malware in this evaluation, it is more useful to use the Windows 7 VM for analysis. Since more indicators were generated with Windows 7, more can be known about the malware and a greater amount of defence or detection mechanisms can be created.

Table 5.22: Network Traffic totals for Windows 8.1 VM with different analysis times

Protocol	5 mins	30 mins	1hour
ICMP	0	0	0
UDP	63	3037	7799
TCP EST	205	641	690
TCP FAIL	71	116	152
DNS	231	3629	3764
FTP	3	15	18
SSH	49	52	68
HTTP	57	655	675

5.4.2 Interesting Observations

With the Windows 7 execution for 60 minutes, the SSH indicator number decreases from the 30 minute execution. It is unclear why, but 10 samples in the 30 minute execution had at least one or more SSH indicators than the 1 hour execution. Maybe at the time, a number of the SSH servers were down.

5.4.3 Summary

Windows 7 was the OS with the most generated indicators as opposed to the Windows XP and Windows 8.1 VMs. Although Windows 8.1 did not perform badly, until more malware is tested it is clear that Windows 7 64 bit is an optimal OS for the generation of IOCs out of the three operating systems.

Execution time does effect the amount of IOCs created. When the time increases, the amount of IOCs increases too. Since the majority of IOCs were created from the first 5 minutes of traffic, which is the default Cuckoo analysis time, then increases, it suggests

that an optimal execution time that balances the amount of IOCs generated and time efficiency is between 5 and 30 minutes.

This evaluation does however show that the Cuckoo default time is useful and this allows for enough time for malware to execute and generate useful traffic.

5.5 Chapter Summary

Section 5.1 started off by presenting the three datasets used in the chapter. Sections 5.2 and 5.3 were two detailed case studies where malware network traffic analysed in detail and the IOCs generated were evaluated. The Chapter ended off with Section 5.4, with a case study that compared the amount of IOCs generated during three different analysis times. The following chapter, Chapter 6, will use the IOCs generated in this Chapter for the evaluations.

Chapter 6

Sharing and Defence

Section 6.1 evaluates the effectiveness of the automatically generated IOCs by turning them into IDS and firewall rules and testing how good they are at detecting and defending against malware on a network level. Section 6.2 showcases the simple sharing platform to demonstrate the generated IOCs can be shared.

6.1 Defence

This case study was designed to evaluate if automatically generated IOCs from the framework, discussed in Chapter 3, can be used to automatically produce IDS and firewall rules that are successful in detecting and mitigating malware as detailed by the IOC. One type of IDS (Snort) and one type of firewall (Iptables) were chosen for testing. Snort was chosen as the test IDS because it is the most popular open source IDS. Iptables was chosen because it is the Linux firewall and Linux is a popular OS in the server space. Custom rules for each sample were generated from the sharing platform's IOC conversion feature, discussed in Section 6.2, and were used to evaluate traffic from a Windows 7 VM executing each malware sample. This Section will discuss the rules generated for Snort and Iptables and the results from 20 ransomware samples that were executed in the virtual machine. Section 6.1.6 explores the types of packets found by Snort and Iptables, and give reasons as to why some rules were unsuccessful. It will also be evaluated to see if any ransomware run in this case study was unable to encrypt data with packets being blocked.

6.1.1 Dataset

A selection of 20 ransomware samples from Section 5.3 were chosen to be executed while custom rules for the Snort IDS or Iptables firewall were running. These samples consisted of malware, that were obtained from VirusTotal, and are detailed in Table 5.1. These include Locky, Cryptowall, DMALocker, VirLock, CTBLocker, File Locker, Tesla Crypt, CryptoLocker and Torrent Locker. All of the files were Windows executables and were used previously, in the evaluation in Section 5.3.

6.1.2 STIX indicators

The IOC types created for the samples during Section 5.3 consisted of TCP Connection Established, TCP Connection Failed, DNS Queries and HTTP Requests. The samples did not generate any UDP, SSH, FTP or ICMP traffic. Because of this result, these types of indicators were not tested.

In order to create IDS or firewall rules, values (described in Section 4.6) were extracted from the STIX indicators using a Python script and placed into a custom rule, which was written to a file. Examples of the Python functions for the creation of Snort and Iptables rules are shown in Listing 18 and 19 from Appendix D. The file contained a number of rules, each on their own line, which made it easy to add them to the Snort rules file or into the terminal (for Iptables).

6.1.3 Snort

Three types of rules were created for the indicators of the ransomware samples. These include TCP, HTTP and DNS rules that trigger an alert when a suspicious packet is identified. Each rule has its own `sid`, which is a keyword used to uniquely identify Snort rules. Examples of the rules are discussed in this Section and shown in Appendix D and the values used to create them.

TCP rule

The TCP Snort alert rule included the source or destination IP addresses and ports from TCP indicators. An example of a TCP rule is shown in Listing 20 from Appendix D.

Because Snort rules can specify a protocol to be IP, ICMP, TCP or UDP, the TCP protocol was naturally selected for these rules. The rule in Listing 20 from Appendix D is for TCP traffic coming from the specified home network, `$HOME_NET`, on any source port to the IP address `184.106.112.172` through port 9001. A message is written to the alert file which is specified in `msg`.

HTTP rule

An example of an HTTP Snort rule is in Listing 21 from Appendix D. The TCP protocol was specified, along with the destination port TCP/80. Two types of content for matching were set, which were for two strings, one in the HTTP header and one in the HTTP URI. The option `nocase` was used so that the content specified can be matched regardless of case. If the strings are matched, a message is written into the alert file.

DNS rule

The Snort rule for alerting to suspicious DNS Requests was created with the destination port value and domain name from the STIX DNS indicators. An example of a DNS Request is shown in Listing 22 from Appendix D. The UDP protocol was specified as DNS uses it and the `content` contains hexadecimal numbers because DNS packets use a byte to show the number of characters that proceed it, as stated in RFC 1035 (Mockapetris, 1987). The `byte_test` field in this case is used to test if the packet is a DNS Query as it performs a NOT AND (!&) on a certain part of the packet.

6.1.4 Iptables

Since the chosen ransomware samples generated indicators of the TCP, HTTP and DNS types, these were used to create Iptables rules that `LOG`, then `REJECT` traffic. `REJECT` is different from `DROP` in that Iptables sends an ICMP destination-unreachable packet to the source instead of sending no response or an ACK/RST.

TCP rule

For this rule, the source or destination port and IP address were used. An example of this rule is shown in Listing 23 from Appendix D. `-A` is for direction, `-j` is the action, `-p`

is the protocol, `--dport` is the destination port and `-d` is the destination address. This rule has the same purpose as the TCP Snort example in Listing 20 from Appendix D.

HTTP rule

For a rule to filter out malicious HTTP Request traffic, the port and full URL address were used. An example of a rule is in Listing 24 from Appendix D. In the rule, `-d` was set as the destination host, `-m` is for matching, `--algo` is the pattern matching method `and` `--string` is the pattern used for matching. This rule filters traffic to the specified host, 'xsso.sisttwtdogpyjlcc.info', if the specified string, '/c8619ce03b36001f0d7c4258684707e5', is found in the packet. As the host domain is resolved when the rule is submitted, it may cause the rule to be outdated if the domain's IP address changes. It might be better not to include the destination in a production system.

DNS rule

The DNS rule rejects DNS requests for certain domains, if the specified hex string is matched. An example of the DNS rule is shown in Listing 25 from Appendix D. The `-m` is for matching, `--u32 "28 & 0xF8 = 0"` means take 4 bytes from the 28th byte of the packet and mask the value with 0xF8[1], `--from 40` starts the string matching in the question part of the DNS packet and `--hex-string` is the string to be matched.

6.1.5 Test environment

The samples were run in a Windows 7 VM for 20 minutes each. The time of 20 minutes was chosen as a result of the outcome of Section 5.4, as it was found that a time between 5 and 30 minutes would be optimal for running that malware. For testing the Snort rules, an Ubuntu 14.04 VM was configured and Snort 2.9.8.3 was installed. Figure 6.1 shows the network setup for the Snort rule evaluation. The Ubuntu and and Windows VMs were on a VirtualBox NAT Network and the Ubuntu VM's adapter was set to promiscuous mode. The Windows machine's gateway address was set to the Ubuntu machine's IP address.

[1]http://unix.stackexchange.com/questions/245763/Iptables-hex-string-block-dns-query

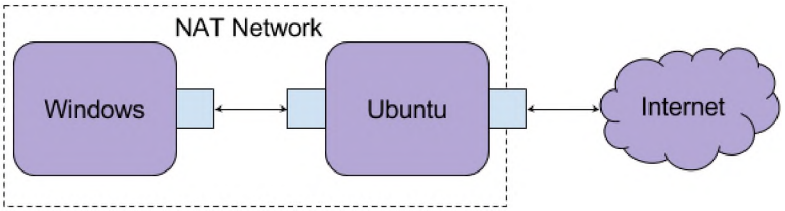

Figure 6.1: Snort network setup

For each of the samples, the custom Snort rules were downloaded via the sharing platform. Snort was set up to use only the custom rules for each sample and none of the default rules that come with it. Snort does come with a set of default rules, but these were not tested as they were not generated by the in scope system. Since all the Snort rules were created to trigger alerts, the Snort alert file was observed and analysed after each sample execution. This was to see if Snort picked up any packets and if so, what type they consisted of.

For the Iptables part of the evaluation, shown in Figure 6.2, the Windows and Ubuntu VMs were on a Host-only network, with the Ubuntu VM having a second network adaptor which was a bridged adaptor.

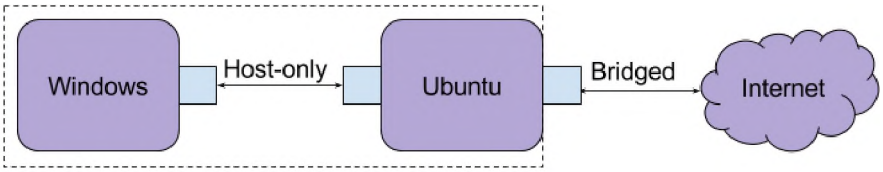

Figure 6.2: Iptables network setup

Like for Snort, custom rules for Iptables were downloaded from the sharing platform. When each sample ran it had its own custom rules set. These rules logged each packet that they picked up into the `/var/log/kern.log` file. This file was then analysed after each sample's execution to see which packets were picked up and rejected.

6.1.6 Results

This section will show the results from testing the Snort IDS with custom rules and the Iptables firewall with custom rules.

Snort

Fifteen out of twenty samples had the intended traffic alerted to, which are shown in Table 6.1. These sample numbers being 3, 4, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18, 19 and 20.

Samples 1, 2, 5, 11 and 14 either did not trigger any Snort alerts or only alerted for some protocols. Taking a closer look at these five samples, starting with sample 1, which did not alert to HTTP traffic, even though there were 10 HTTP alert rules created, seven of which were for the same host but with similar URIs which look like they had been generated from a Domain Generation Algorithm, as mentioned in Section 2.6. Since a DGA may have been used, these IDS rules may only pick up a small proportion of the domain strings generated.

Sample 2 had 20 DNS rules, all for domains that were generated by a DGA. By looking at the PCAP, the sample was sending DNS Requests for other generated domains, which were not picked up. A snippet of the PCAP shown in Wireshark is in Figure 6.3.

811	238.028336	146.231.129.97	53 DNS	Standard query 0x2f81 A jfocdvfqcavo.biz
802	235.180609	146.231.129.97	53 DNS	Standard query 0x182c A ywpxyer.info
790	223.322182	146.231.129.97	53 DNS	Standard query 0xab6d A mucxffmvcl.pl
678	127.323926	146.231.129.97	53 DNS	Standard query 0xaeb3 A iqtgwjfguxdk.biz
669	125.039687	146.231.129.97	53 DNS	Standard query 0x6a25 A smpuuvcyftd.pl
635	116.230647	146.231.129.97	53 DNS	Standard query 0x900f A nhwwykopd.ru

Figure 6.3: DGA domains queried from sample 2

Sample 5 was meant to generate TCP requests, which were not sent during execution in this case study.

Sample 11 had rules to alert at a large number of domain names, but closer inspection suggests they were generated by a DGA. Since none of these domains were picked up, it further suggests that indicators may not be useful when a DGA is in play, unless every one of the generated domains is known. For HTTP, there were two rules with one host domain and different URIs, but these rules did not work because the host domain was never queried by the DGA during the execution in this case study. The same goes with the TCP rule. It was never triggered, because different domains were being queried and the DNS responses would come back with a different IP.

Sample 14 had 13 DNS rules, which partly look like they are from a DGA, which is most likely the reason for none of its traffic being detected.

Table 6.1: Indicators of the 20 samples and traffic types caught by Snort

Sample #	Indicators	TCP alert	HTTP alert	DNS alert
1	TCP, HTTP, DNS	☑		☑
2	TCP, DNS	☑		
3	TCP, HTTP, DNS	☑	☑	☑
4	TCP, HTTP, DNS	☑	☑	☑
5	TCP			
6	TCP, DNS	☑		☑
7	TCP	☑		
8	TCP	☑		
9	TCP, HTTP, DNS	☑	☑	☑
10	TCP, HTTP, DNS	☑	☑	☑
11	TCP, HTTP, DNS			
12	TCP, DNS	☑		☑
13	TCP, HTTP, DNS	☑	☑	☑
14	DNS			
15	TCP, HTTP, DNS	☑	☑	☑
16	TCP, HTTP, DNS	☑	☑	☑
17	TCP, DNS	☑		☑
18	TCP, HTTP, DNS	☑	☑	☑
19	TCP, HTTP, DNS	☑	☑	☑
20	TCP, HTTP, DNS	☑	☑	☑

Iptables

As indicated in Table 6.2, three samples (2, 7, 8), had all the intended traffic rejected. This was because the three samples did not have HTTP indicators. In the Snort rule evaluation, Snort was not able to pick up DNS requests because none of the domains being created by the DGA in the sample were queried. However, when the sample ran again for the Iptables evaluation, the DGA generated some domains that matched the reject rules. This shows that when this specific sample runs, different domains are queried and only some of which can be blocked as they were identified in the initial ransomware case study. The automated generation of IDS and firewall rules are not consistent when working with malware with DGAs.

No HTTP traffic was blocked throughout the whole evaluation, because a number of DNS

requests were being blocked and the TCP traffic being blocked did not allow established connections to the HTTP servers, or there were new HTTP requests generated that were not seen in the original executions on the binaries.

Seven samples had the Iptables firewall blocking both TCP and DNS requests with no HTTP connections being established in the process. This may indicate that in terms of these samples, the HTTP firewall rules were not needed. Eight samples only had their DNS requests blocked, which seemed to be enough to stop TCP requests from occurring.

This shows that the DNS indicators were successfully used to create Iptables firewall rules that block traffic effectively. One malware sample did not execute completely, like it did in the Snort case study. It may be because of the blocked traffic. Sample 19 is an example of the rules being used to block communication with the C&C server. Figure 6.4 shows a screenshot of the packets in Wireshark, where the reply to a TCP request to the IP address 154.35.32.5 is an ICMP packets with the destination unreachable message. The ICMP request was sent by the VM with the Iptables firewall. Figure 6.5 is a screen shot of DNS requests with the replies being ICMP destination unreachable ports.

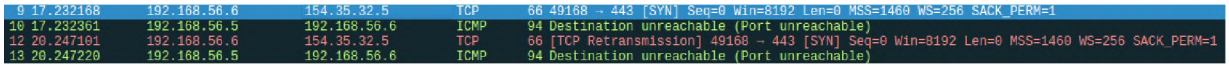

Figure 6.4: Wireshark screen shot of packets from sample 19

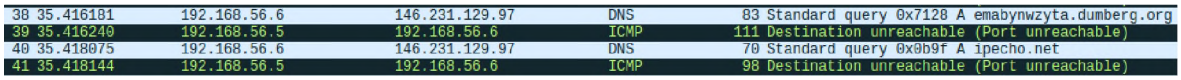

Figure 6.5: Wireshark screen shot of DNS requests blocked from sample 19

Table 6.2: Indicators of the 20 samples and traffic types caught by Iptables

Sample #	Indicators	TCP reject	HTTP reject	DNS reject
1	TCP, HTTP, DNS	☑		☑
2	TCP, DNS	☑		☑
3	TCP, HTTP, DNS			☑
4	TCP, HTTP, DNS			☑
5	TCP			
6	TCP, DNS			☑
7	TCP	☑		
8	TCP	☑		
9	TCP, HTTP, DNS	☑		☑
10	TCP, HTTP, DNS	☑		☑
11	TCP, HTTP, DNS			
12	TCP, DNS			☑
13	TCP, HTTP, DNS			☑
14	DNS			
15	TCP, HTTP, DNS			☑
16	TCP, HTTP, DNS			☑
17	TCP, DNS			☑
18	TCP, HTTP, DNS	☑		☑
19	TCP, HTTP, DNS	☑		☑
20	TCP, HTTP, DNS	☑		☑

6.1.7 Successful Encryption

The VM had five files on the desktop, which were visible when Cuckoo took screenshots of the VM. These screenshots were observed to see if the files on the desktop had new file extensions or changed in anyway indicating they had been tampered with. Since each sample was run manually for the Snort and Iptables evaluations, the encryption status of the desktop files in the VM were checked after each sample execution completed.

Table 6.3 shows when the ransomware samples were able to execute correctly and encrypt the files in the VM. Column two shows the results from when the ransomware was initially run through the Cuckoo sandbox and column three and four are when they were manually run on a VM. The last column is the name of the ransomware family. This comparison

was done to see if the malware changed its behaviour from being run with Cuckoo and then manually in a VM. Also to see if the IDS and firewall rules generated from the IOCs effected the malware's execution.

Ten samples were not able to encrypt data (in all execution environments) in the first 20 minutes of running and four encrypted data every time they executed. The four samples (6, 7, 13 and 16) included two VirLock, one FileLocker and one TeslaCrypt.

In three cases (samples 5, 15 and 17) the malware did not end up encrypting data while running in Cuckoo, but did successfully encrypt data in the VM. These samples were of DMA Locker, TeslaCrypt and CryptoLocker type. It is unclear as to why this happened.

In two cases, samples 10 and 20 (CTBLocker and TorrentLocker), the ransomware successfully encrypted files when run in the Cuckoo sandbox, but not manually in the VM. In Table 6.1, Snort was able to pick up TCP, HTTP and DNS indicators, which shows that the malware uses static values, and not a DGA for example, in the generation of network packets. None of the domains were resolved for both of these samples in the Cuckoo, Snort and Iptables evaluations. This indicates that the domains were no longer in use, because they may have been taken down or disposed of.

Only one sample, 19 (Torrent Locker), had a varying execution between the Snort and Iptables case studies. As Iptables was blocking traffic for the indicators, this suggests that the malware was unable to execute when the connections it was attempting to make were being blocked. This is one case where it seems that the indicators were useful in mitigating the execution of a ransomware sample. The other two TorrentLocker samples, 18 and 20, did not encrypt data during the Snort and Iptables evaluations.

Table 6.3: Comparison of successful execution of ransomware samples

Sample #	Cuckoo	Snort	Iptables	Family
1	no	no	no	Locky
2	no	no	no	Locky
3	no	no	no	CryptoWall
4	no	no	no	CryptoWall
5	no	yes	yes	DMALocker
6	yes	yes	yes	VirLock
7	yes	yes	yes	VirLock
8	no	no	no	CTBLocker
9	no	no	no	CTBLocker
10	yes	no	no	CTBLocker
11	no	no	no	FileLocker
12	no	no	no	FileLocker
13	yes	yes	yes	FileLocker
14	no	no	no	TeslaCrypt
15	no	yes	yes	TeslaCrypt
16	yes	yes	yes	TeslaCrypt
17	no	yes	yes	CryptoLocker
18	no	no	no	TorrentLocker
19	yes	yes	no	TorrentLocker
20	yes	no	no	TorrentLocker

6.1.8 Summary

Custom rules for 20 ransomware sample were tested for each Snort and Iptables. It was found that Snort was able to detect all but three of the samples. Snort was also able to pick up all the intended packets, according to type of IOC, from 15 of the samples. For three samples Iptables was able to block all the intended traffic. No HTTP traffic was blocked because it was not seen because of the TCP and DNS requests being blocked. Iptables was successful in blocking DNS and TCP packets using information from the IOCs. One sample in the Iptables case study was not able to execute, which may have been from traffic being blocked, which shows that IOCs were useful for detecting malicious traffic and occasionally stopping the intended execution.

In terms of ransomware families, Locky, CryptoWall samples never encrypted data in the first 20 minutes of execution. DMA Locker and CryptoLocker did not encrypt in Cuckoo, but in the VM. VirLock encrypted data every time it ran. CTB Locker did not encrypt, except for one of the three samples executing in Cuckoo. FileLocker either executed in every execution of none of them. TeslaCrypt and Torrent Locker had varying encryption statues for each of their three samples executions. This shows that ransomware, even from the same family, has different behaviours and variations of executions from Cuckoo and VMs.

The automatically generated IDS and firewall rules have the greatest impact on malware samples that do not use DGAs and generate the same packets each time they execute. DGAs are troublesome and with only knowing a few of the domains (from the original execution), a comprehensive list of DNS IOCs can not be made.

6.2 Sharing Platform

In this case study, it is demonstrated how the sharing component, developed for the dissertation to demonstrate how IOCs can be shared, works. Parts of the case study were published in (Rudman and Irwin, 2016b). A user is signed up, then logged in, then all of the indicators and users are viewed, and eventually a STIX file is uploaded. The last part demonstrates that the upload and download API functions of the web application work.

6.2.1 Functionality

Figure 6.6 shows the homepage of IOC Xchange. There are two options to choose from on the homepage, Sign up or Login.

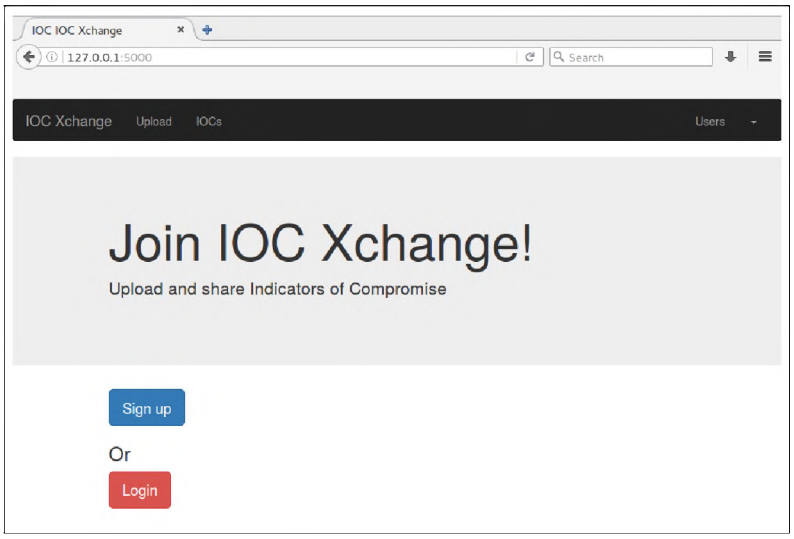

Figure 6.6: Homepage of IOC Xchange

When the Sign up button was selected, the page shown in Figure 6.7 was loaded. A new user called "1337h4k3r" was added in to the Username field and an email address and password were entered. The register button was then pressed and the user data was saved to the database.

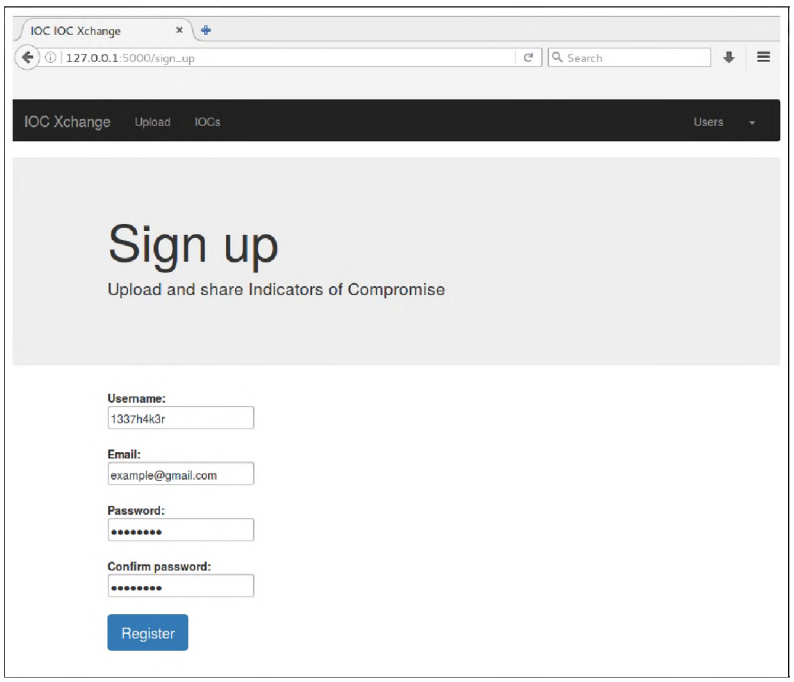

Figure 6.7: Sign up for IOC Xchange

This brought up the page in Figure 6.8, where the username and password of the new

user were entered and the Login button selected. If the two values were incorrect, an informative message would have been displayed.

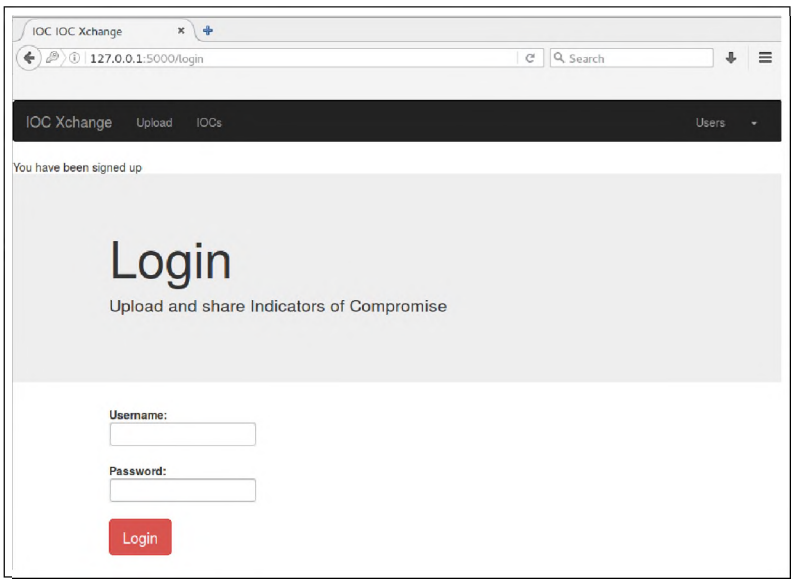

Figure 6.8: Login page after sign up IOC Xchange

Since the values were correct, the page in Figure 6.9 was loaded and the top bar had the Upload, IOCs, Users and Username options revealed.

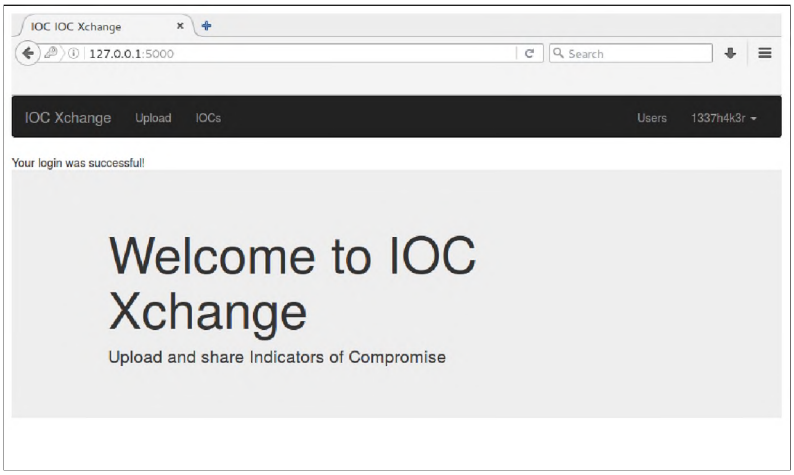

Figure 6.9: Logged in successfully to IOC Xchange

When the Upload tab was selected, the page shown in Figure 6.10 was displayed.

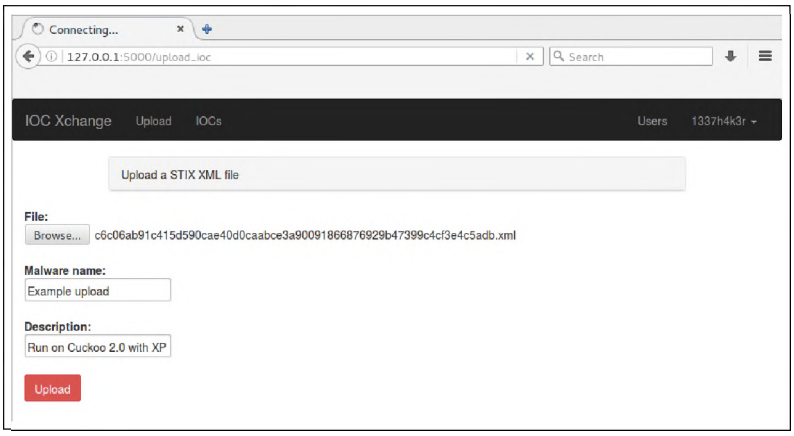

Figure 6.10: Upload a file of STIX indicators

An STIX XML file was selected from the local system and the name and description were input. Next the Upload button was pressed, which uploads the file and saves the details to the database. When the file had been uploaded, the IOCs page was displayed (shown in Figure 6.11), with the top row being the file that was uploaded.

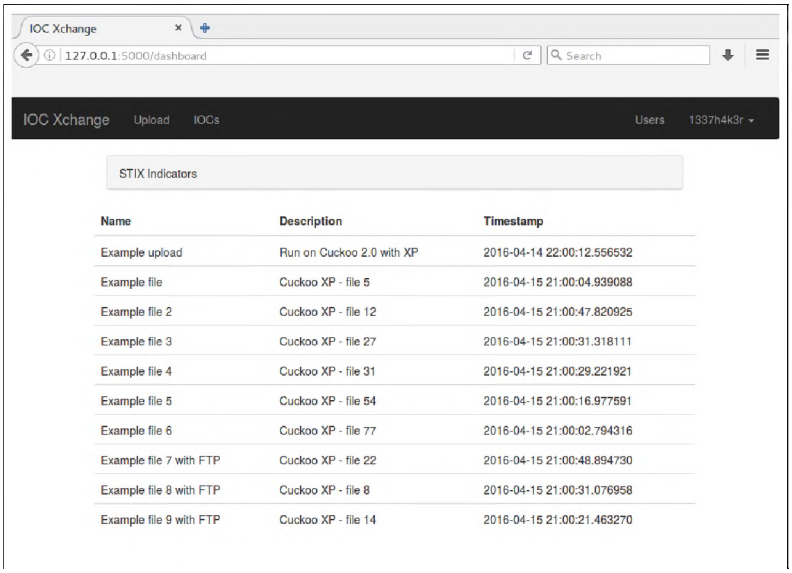

Figure 6.11: Clicked on the IOC tab to view other IOCs

The name, description and timestamp of the upload were displayed for each file and each row could be clicked to display the indicators from the file. The displayed files were uploaded by the other users of the site.

The top row was clicked and the information about the file that was just uploaded was displayed along with the indicators. As seen in Figure 6.12, the malware name (Example

upload), uploader's username (the green bar links to the user's profile), the description and upload date were displayed metadata for the uploaded file. Headings of the types of indicators found, in this case HTTP GET or POST and IP addresses were viewed as they appeared on the webpage. There were six IP addresses and one HTTP GET request shown in this case.

At the bottom of the page shown in Figure 6.12 were four buttons, Snort Rules, IPFW Rules, Iptables Rules and STIX XML.

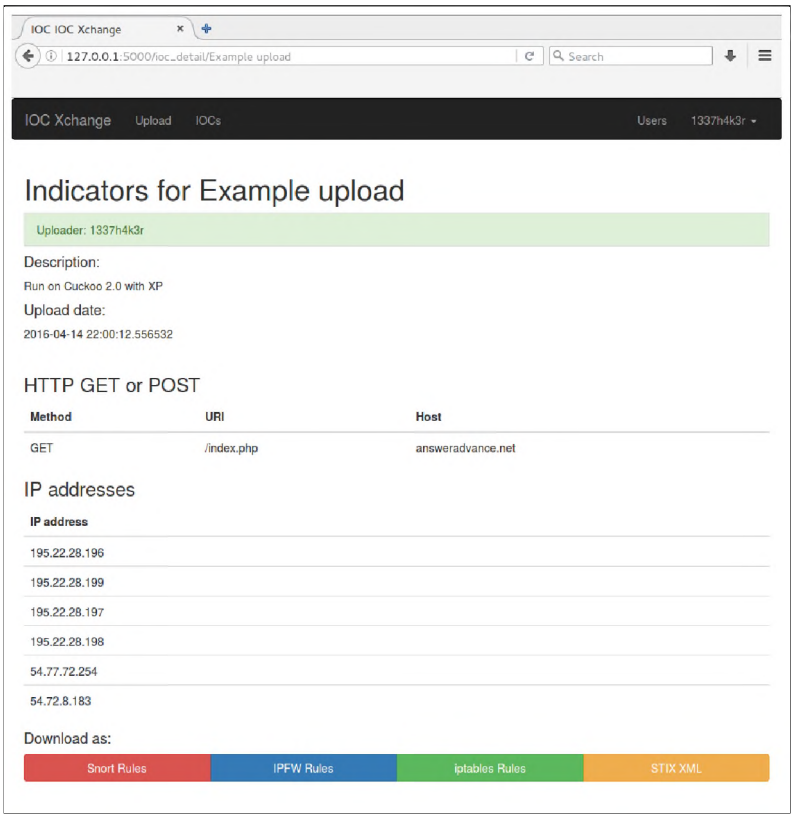

Figure 6.12: Viewing the uploaded file's IOCs

Each button is for downloading the indicators in a different format. The STIX XML button downloads the original uploaded file and the others are for converting the XML file to IDS or firewall rules. The IPFW Rules was pressed and the box shown in Figure 6.13 was shown which allows the file to be downloaded. This was the method used to generate the Snort and Iptables rules from Section 6.1.

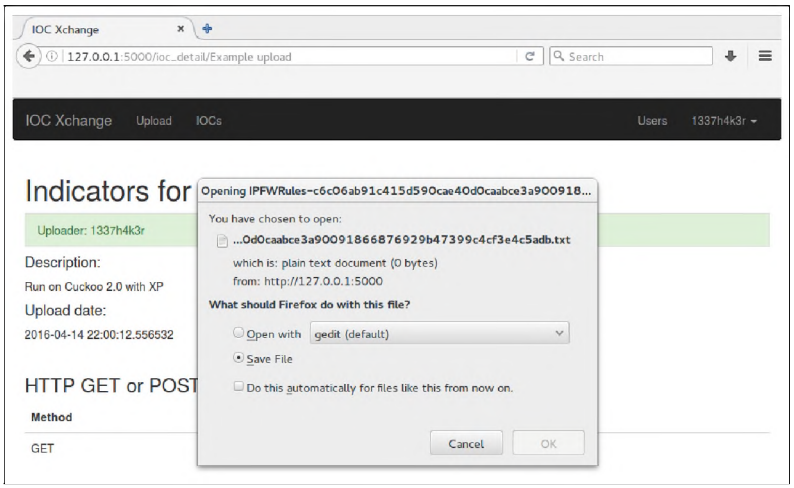

Figure 6.13: Clicked the IPFW Rules button

When the Users tab was selected, the page shown in Figure 6.14 was displayed showing all of the registered users of the site.

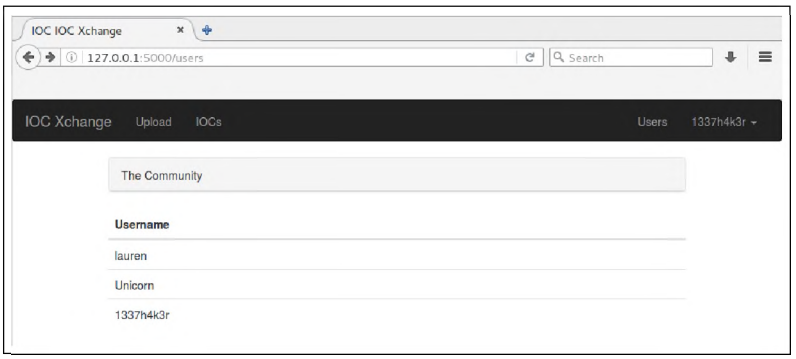

Figure 6.14: Viewing the other users

There were three registered users displayed and each row was selectable. When the 1337h4k3r profile was selected from the list, the Profile page in Figure 6.15 was displayed. The profile consisted of a list of files the user had uploaded. In this case it was the example file that was uploaded earlier. Each row was selectable and lead to the page, from Figure 6.12. A user can get to their own profile by clicking their name in the top right corner and selecting Profile, shown in Figure 6.15. They can also logout through this tab.

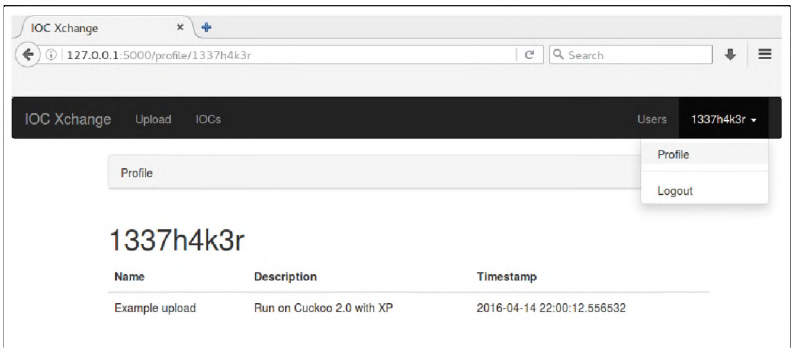

Figure 6.15: Example of user profile

Displaying each type of indicator

The sharing platform displays information about each indicator differently. For IP addresses, it shows an IP address on each row, which was shown in Figure 6.16.

Figure 6.16: Example of IP address indicators

For UDP indicators, the direction of the connection was shown in the third column, shown in Figure 6.17. If the direction was 'out', then the destination port and IP address are shown and visa versa.

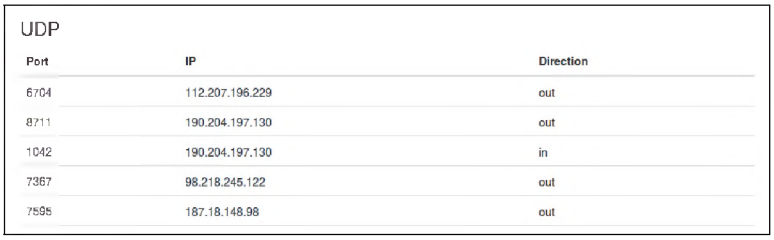

Figure 6.17: Example of UDP indicators

TCP indicators were represented by three values, shown in Figure 6.18, similar to UDP with the port IP address and direction.


TCP

Port	IP	Direction
21	188.165.230.79	out
50441	188.165.230.79	out
50676	188.165.230.79	out

Figure 6.18: Example of TCP indicators

HTTP GET or POST requests were represented through three values shown in Figure 6.19, the Method, URI and Host.


HTTP GET or POST

Method	URI	Host
GET	/index.php	orderflower.net
GET	/index.php	heavencorner.net

Figure 6.19: Example of HTTP indicators

DNS requests were displayed as domain names as shown in Figure 6.20. The domain names in Figure 6.20 most likely indicate a domain generation algorithm in the code of the malware sample. This is because the domains consist of two words, the first words seem to include 'return', 'various', 'forward' and 'degree' (the rest of the domains not displayed in Figure 6.20 justify this). An individual second word seems to be applied to two first words, which is an interesting find.


DNS

Domain
variouscorner.net
returncorner.net
degreeadvance.net
forwardadvance.net
degreestranger.net
forwardstranger.net
degreegoodbye.net
forwardgoodbye.net
degreefortieth.net
forwardfortieth.net

Figure 6.20: Example of DNS indicators

Four values were chosen to describe an FTP connection. These are the port, IP address, direction and description. Because CybOX does not have properties to house FTP usernames, passwords or response codes, they were placed in the indicator's `description` field. As seen in Figure 6.21, the description column includes a username and password, with

information that confirms that the two values still work. There was also information on a file that was retrieved from the FTP server, called '`InstallFramework_150063j.exe`'.

FTP

Port	IP	Direction	Description
21	188.165.230.79	in	Service ready for new user: ProFTPD 1.3.5 Server (Serveur FTP PC SOFT) [188.165.230.79]
21	188.165.230.79	out	Requested username: framework
21	188.165.230.79	out	Requested Password: framework
21	188.165.230.79	in	User logged in
21	188.165.230.79	in	Requested file action okay, completed.
21	188.165.230.79	out	Retrieve a copy of the file: InstallFramework_150063j.exe

Figure 6.21: Example of FTP indicators

API Upload and Download

Since the sharing platform has two simple API features, they were tested too. The upload feature was tested first using Curl[2], which is a tool to transfer data to or from a server using certain protocols. The code shown in Listing 26 in Appendix D was used for the upload. There were four variables set, `file` (the path of the file to upload), `malware_name`, `api_key` and `description`. Looking at Figure 6.22, where the code was run, it shows in the bottom line that there was a successful upload. Figure 6.23 shows the uploaded files on the site and the bottom line ("API Example") shows the file that was uploaded by the API upload function. The Curl script could be used to bulk upload files if needed, by using a simple bash script.

```
lauren@lauren-desktop:/media/lauren/SPACE$ curl -v -F "file=@loc.xml" -F "malware_name=API Example" -F "a
pi_key=65eab40bf1bcd5c82c6d9e02abea5ed3" -F "description=Uploaded from API" http://localhost:5000/api/upl
oad_ioc
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 5000 (#0)
> POST /api/upload_ioc HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.43.0
> Accept: */*
> Content-Length: 115934
> Expect: 100-continue
> Content-Type: multipart/form-data; boundary=------------------------8001f46190767718
>
< HTTP/1.1 100 Continue
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 19
< Server: Werkzeug/0.11.4 Python/2.7.10
< Date: Thu, 21 Apr 2016 16:44:05 GMT
<
* closing connection 0
All good, uploaded.lauren@lauren-desktop:/media/lauren/SPACE$
```

Figure 6.22: Successful API upload from terminal

[2]https://curl.haxx.se/docs/manpage.html

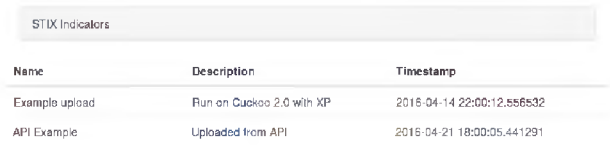

Figure 6.23: Uploaded by API upload

The Download function was also tested using Curl and the three variables shown in the download code in Listing 27 in Appendix D are `malware_name`, `api_key` and `convert_type`. The `convert_type` specifies that type of content that will be in the downloaded file. There are four options 'snort', 'ipfw', 'Iptables' and 'none' (downloads the original XML file). Figure 6.24 shows the Curl code that sends an HTTP POST request to the web application and in the second last line, the downloading of the file. The file downloaded was full of Snort rules created from the STIX indicators in the previously uploaded file. This Curl code can be used in a script that is used to bulk download files based on their name.

Figure 6.24: Successful API download from sharing platform

6.2.2 Summary

This Section described the functionality of the proof of concept sharing platform and showed how each type of STIX indicator would be represented in HTML. This included signing up to the application, logging in, uploading an STIX XML file, viewing the uploaded file and viewing other user's profiles. It was found that STIX data is easily represented using HTML and the layout was clear and simplistic. The downloading of indicators was successful too and could be converted into three types of rules, Snort, Ipt-

ables and IPFW. This is a useful part of the sharing platform as it makes the creation of defence mechanisms more efficient.

Since the sharing platform had two API methods, those were tested as well where a successful upload and download of files were performed.

6.3 Chapter Summary

The effectiveness of the automatically generated IDS and firewall rules was evaluated in Chapter 6.1. The automatically generated IDS and firewall rules have the greatest impact on malware samples that do not use DGAs and generate the same packets each time they execute. Ransomware, even from the same family, has different behaviours and variations of executions from Cuckoo and VMs and one sample in the Iptables case study was not able to execute, which may have been caused by blocked traffic, which suggests that IOCs were useful for detecting malicious traffic. Section 6.2 showcased the simple sharing platform and all the intended functionalities worked.

Chapter 7

Conclusion

This work set out to prove the feasibility of the automated generation of useful network based IOCs using a sandbox-based system. Chapter 1 began by detailing the problems involved with not sharing threat information, with the number of new malware discovered increasing rapidly each year. It exposed that there is a large amount of research on malware analysis, but a lack of research of reporting and sharing the results. The concept of dynamic malware analysis was introduced and how it usually focuses its results on the behaviour of the malware on the system and not network traffic. Finally the goals and scope of the research were discussed.

Chapter 2 provided an overview of the relevant literature in the fields of malware analysis, IOCs and sharing threat information. It described the dynamic malware analysis and the common types of analysis systems, together with some systems that perform traffic analysis after dynamic analysis. Various types of IOC formats were investigated, along with the types of systems designed to share them. This chapter ended with an examination of how IOCs can be used for network defence, focusing on IDS and firewall rules, and giving an overview of the common types.

The design of the system was presented in Chapter 3, beginning with a general overview before moving on to describing the two main system components: the Cuckoo sandbox setup and the virtual machine setup. The design of the proof of concept sharing platform was detailed at the end of this chapter, giving the implementation choices and logic of the system.

Chapter 4 focuses on the automated generation of IOCs and the design of the tool-chain that accomplished this. The design choices made, including the choice of using CybOX and

STIX, were justified after a comparison between OpenIOC and CybOX was presented. The logic contained within the traffic filter and generation of each type of indicator is explored in detail, ending with how the IOCs were converted to IDS and firewall rules for an evaluation in Section 6.1.

Finally Chapters 5 and 6 describe the various evaluations that were conducted to determine the types of traffic that is experienced when conducting malware analysis (in Sections 5.2 and 5.3) and its usefulness in the creation of IOCs that can eventually be shared and used for detecting and defending against malware on a network level (in Sections 6.1 and 6.2). A small evaluation was also carried out about the length of dynamic analysis and the effect on the amount of network indicators. The research ended with a showcase of the simple sharing platform.

7.1 Findings

The first two goals of this research were to find a way to automate the generation of comprehensive network level IOCs from malware samples and evaluate their usefulness for the detection and mitigation of malware. Both of these undertakings proved to be successful. Sections 5.2 and 5.3 demonstrated that a malware sample can be given to the system and comprehensive network indicators for UDP, TCP, DNS and HTTP can be generated. The sandbox environment proved effective together with dynamic analysis at generating traffic. The detailed look at traffic generated by the samples, found that many of the domains observed were compromised sites or generated by a DGA. DGAs are one weakness of the system as only a small portion of the domains that an algorithm can generate, are captured as IOCs. Some common domains, such as `wikipedia.org` and `torproject.org` were observed, which would trigger excessively if used in IDS or firewall rules. This shows the need for filtering of non-malicious common domains. Another weakness to the system, was the extraction of IP addresses from the binary file, by searching through the output from the Cuckoo Strings module. Most of the strings that the framework identified, were not IP addresses, but version numbers. This feature should be refined in the future.

Section 5.4 found that with the 325 samples used for testing, the optimal time for generation of IOCs is between 5 and 30 minutes and that analysis time is in proportion to the amount of IOCs generated. Because this was a simple evaluation, a larger dataset may need to be evaluated with more timings in future work to get more accurate results. However, it did find that when running the 325 samples in Windows XP, 7 and 8.1, Windows 7 generated the most IOCs.

The final research in Section 6.1 ably demonstrated that the IOCs for 20 ransomware samples were successfully converted into Snort IDS and Iptables firewall rules, combined with these rules being able to detect the samples on the network 17 out of 20 times. The Iptables rules, had the limitation of not picking up HTTP traffic, which may be because of the TCP SYN packets to the HTTP servers being dropped. This demonstrates that these rules may be irrelevant.

The showcased sharing platform in Section 6.2 was found to work as it was described, but is simple and needs a few improvements. One of the most useful parts of it proved to be the IOC to firewall and IDS rule download option, which made the previous evaluation easier to carry out.

In closing, this research has successfully demonstrated, using dynamic malware analysis, that network based IOCs can be automatically generated and can be used for the detection of malware. Some success was found in using IOCs for defence.

7.2 Future Work

During development and throughout the evaluations a number of potential core improvements were identified. These ranged from improvements to the analysis logic to new evaluations that can be conducted to better evaluate the framework.

1. Since the baseline traffic from the sandbox was filtered, the IOCs created were from traffic generated from the malware, however some of this traffic generated from the malware were common non-malicious domains. Future work for creating filters for these types of domains needs to be conducted. A suggestion, from Section 5.3, is the Alexa top one million websites list[1], which can be used to filter out domains that would trigger a lot, if made into IDS and firewall rules. If the domain is also seen being queried with HTTP, the full URL path with the domain can be used in the rules, instead of only using the domain alone.

2. Since each STIX indicator was created for one type of IOC, the IOCs could be combined with AND and OR operators to provide a more in depth description of the network behaviour.

[1] http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

3. An evaluation on the usefulness of the UDP, SSH and FTP indicators needs to be conducted as the Section 6.1, did not include samples with these types of indicators. Regardless, they still provide other information such as the SSH public key and FTP login details which can be used by a malware analyst for further testing and gathering of threat information.

4. A wider variety of firewall and IDS rules can be added to the IOC to firewall and IDS rules converter code in the sharing platform, to cater for a wider variety of operating systems. However, the sharing platform as a whole could be improved, by adding a more detailed user profile, with a reputation rating and user description. One disadvantage of the sharing platform is that there are only two API functions, upload and download. In the future it would therefore be useful to implement functions such as, search and edit.

References

Abrams, L. Cryptolocker ransomware information guide and faq. October 2013. [Accessed on: 13 October 2016].
URL https://www.bleepingcomputer.com/virus-removal/cryptolocker-ransomware-information

Albin, E. and Rowe, N. C. A realistic experimental comparison of the suricata and snort intrusion-detection systems. In *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, pages 122–127. IEEE, 2012.

AlienVault. Threat intelligence sharing & the government's role in it. 2015. [Accessed on: 3 April 2016].
URL https://www.alienvault.com/resource-center/white-papers/governments-role-in-threat-intelligence-sharing

AlienVault. Welcome to AlienVault Open Threat Exchange! 2016. [Accessed on: 3 April 2016].
URL https://www.alienvault.com/open-threat-exchange

Antsilevich, U. J., Kamp, P.-H., Nash, A., Cobbs, A., and Rizzo, L. ipfw: IP firewall and traffic shaper control program. *FreeBSD System Manager's Manual*, 2016.

AplusWebMaster. 'Changed Identification Numbers', 'Hilton Hotel' SPAM, Zombie 'Orkut' Phish ... Forum Post, July 2015. [Accessed on: 1 Novemeber 2015].
URL https://forums.spybot.info/showthread.php?23632-SPAM-frauds-fakes-and-other-MALWARE-deliveries/page75

AV-TEST. Malware. August 2016. [Accessed on: 13 October 2016].
URL https://www.av-test.org/en/statistics/malware/

Avast. A closer look at the locky ransomware. March 2016. [Accessed on: 13 October 2016].
URL `https://blog.avast.com/a-closer-look-at-the-locky-ransomware`

Aycock, J. Computer Viruses and Malware, volume 22. Springer Science & Business Media, 2006. Page 12-18.

Aydın, M. A., Zaim, A. H., and Ceylan, K. G. A hybrid intrusion detection system design for computer network security. *Computers & Electrical Engineering*, 35(3):517–526, 2009.

Barnum, S. Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX). Technical report, MITRE Corporation, February 2014. [Date Accessed: 27 April 2016].
URL `https://stixproject.github.io/getting-started/whitepaper/`

Bayer, U., Habibi, I., Balzarotti, D., Kirda, E., and Kruegel, C. A view on current malware behaviors. In *2nd Usenix Workshop on Large-scale Exploits and Emergent Threats*. 2009.

Bayer, U., Kruegel, C., and Kirda, E. TTAnalyze: A tool for analyzing malware. Ph.D. thesis, Technical University of Vienna, 2006.

Bhardwaj, A., Avasthi, V., Sastry, H., and Subrahmanyam, G. Ransomware digital extortion: A rising new age threat. *Indian Journal of Science and Technology*, 9:14, 2016.

Binsalleeh, H. Analysis of Malware and Domain Name System Traffic. Ph.D. thesis, Concordia University, 2014.

Bisson, D. The Dridex botnet ain't done yet, say researchers. News Article, October 2015. [Accessed on: 23 October 2015].
URL `https://grahamcluley.com/2015/10/dridex-botnet-dead/`

Bottomley, K. Tracking the footprints of ransomware. August 2015. [Accessed on: 13 October 2016].
URL `https://blog.opendns.com/2015/08/20/tracking-the-footprints-of-ransomware/`

Branco, R. R., Barbosa, G. N., and Neto, P. D. Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies. *Black*

Hat, 2012. [Accessed on: 27 April 2016].
URL http://www.kernelhacking.com/rodrigo/docs/blackhat2012-paper.pdf

Casey, E., Back, G., and Barnum, S. Leveraging cybox to standardize representation and exchange of digital forensic information. *Digital Investigation*, 12:102–110, 2015.

Chen, P., Huygens, C., Desmet, L., and Joosen, W. Advanced or not? a comparative study of the use of anti-debugging and anti-vm techniques in generic and targeted malware. In *IFIP International Information Security and Privacy Conference*, pages 323–336. Springer, 2016.

Chen, X., Andersen, J., Mao, Z. M., Bailey, M., and Nazario, J. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 177–186. IEEE, 2008.

Chismon, D. and Ruks, M. Threat intelligence: Collecting, analysing, evaluating. Technical report, MWR InfoSecurity, 2015.

Cover, R. Incident Object Description and Exchange Format (IODEF). 2008. [Accessed on: 13 June 2016].
URL http://xml.coverpages.org/iodef.html

CSIRT Gadgets Foundation. Collective intelligence framework. 2015. [Accessed on: 16 July 2016].
URL http://csirtgadgets.org/collective-intelligence-framework/

Cuckoo Foundation. Processing modules. 2015a. [Date accessed: 3 March 2016].
URL http://docs.cuckoosandbox.org/en/latest/customization/processing/

Cuckoo Foundation. Reporting modules. 2015b. [Date accessed: 3 March 2016].
URL http://docs.cuckoosandbox.org/en/latest/customization/reporting/

Dandurand, L. Cyber security information exchange. 2013. [Accessed on: 16 July 2016].
URL https://www.rsaconference.com/writable/presentations/file_upload/sect-t08-cyber-security-information-exchange.pdf

Danyliw, R., Meijer, J., and Demchenko, Y. Rfc 5070: The incident object description exchange format. 2007. [Accessed on: 9 July 2016].
URL https://www.ietf.org/rfc/rfc5070.txt

Davidson, M. and Schmidt, C. Taxii–overview. Technical report, The Mitre Corporation, January 2014. [Accessed on: 28 April 2016].
URL `https://taxiiproject.github.io/releases/1.1/TAXII_Overview.pdf`

Dela Paz, R. CryptoWall, TeslaCrypt and Locky: A Statistical Perspective. Blog Post, March 2016. [Accessed on: 3 June 2016].
URL `https://blog.fortinet.com/2016/03/08/cryptowall-teslacrypt-and-locky-a-statistical-perspective`

Di Pietro, R. and Mancini, L. V. Intrusion Detection Systems, volume 38. Springer Science & Business Media, 2008.

Doniec, A. DMA Locker: New Ransomware, But No Reason To Pani. February 2016. [Accessed on: 5 May 2016].
URL `https://blog.malwarebytes.com/threat-analysis/2016/02/dma-locker-a-new-ransomware-but-no-reason-to-panic/`

Duncan, B. 2015-02-02 - malspam run pushes chanitor - subject: Logmein promo code - get 50February 2015a. [Accessed on: 1 Novemeber 2015].
URL `http://www.malware-traffic-analysis.net/2015/02/02/index.html`

Duncan, B. Upatre/Dyre - the daily grind of botnet-based malspam. Forum Post, 2015b. [Accessed on: 1 November 2015].
URL `https://isc.sans.edu/forums/diary/UpatreDyre%20the%20daily%20grind%20of%20botnetbased%20malspam/19657/`

Egele, M., Scholte, T., Kirda, E., and Kruegel, C. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR)*, 44(2):6, 2012.

ENISA. Share, protect-solutions for improving threat data exchange among certs, 2013. Technical report, European Union Agency for Network and Information Security, November 2013. [Accessed on: 16 July 2016].
URL `https://www.enisa.europa.eu/publications/detect-share-protect-solutions-for-improving-threat-data-exchange-among-certs`

Farnham, G. and Leune, K. Tools and standards for cyber threat intelligence projects. Technical report, 2013. [Accessed on: 15 May 2016].
URL `https://www.sans.org/reading-room/whitepapers/warfare/tools-standards-cyber-threat-intelligence-projects-34375`

Fransen, F., Smulders, A., and Kerkdijk, R. Cyber security information exchange to gain insight into the effects of cyber threats and incidents. *e & i Elektrotechnik und Informationstechnik*, 132(2):106–112, 2015.

Frink, L. Dridex botnet distributor now serves avira. Blog post, February 2016. [Accessed on: 29 April 2016].
URL `http://blog.avira.com/dridex_serves_avira/`

Fujiwara, K., Sato, A., and Yoshida, K. DNS Traffic Analysis CDN and the World IPv6 Launch. *Information and Media Technologies*, 8(3):833–842, 2013.

Gadhiya, S. and Bhavsar, K. Techniques for malware analysis. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(4):972–975, 2013.

Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., and Vázquez, E. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1):18–28, 2009.

Geyer, C. OASIS Advances Automated Cyber Threat Intelligence Sharing with STIX, TAXII, CybOX. Blog Post, July 2015. [Accessed on: 29 November 2015].
URL `https://www.oasis-open.org/news/pr/oasis-advances-automated-cyber-threat-intelligence-sharing-with-stix-taxii-cybox`

Harrington, C. Sharing indicators of compromise: An overview of standards and formats. Conference Presentation, November 2013. [Accessed on: 14 April 2016].
URL `https://www.rsaconference.com/writable/presentations/file_upload/dsp-w25a.pdf`

Holz, T., Gorecki, C., Rieck, K., and Freiling, F. C. Measuring and detecting fast-flux service networks. In *15th Network and Distributed System Security Symposium*. 2008. [Accessed on: 22 July 2016].
URL `http://ei.rub.de/media/emma/veroeffentlichungen/2012/08/07/FastFlux-NDSS08.pdf`

Hun-Ya Lock, A. K. Using IOC (indicators of compromise) in malware forensics. Technical report, SANS Institute, 2013. [Accessed on: 22 April 2016].
URL `https://www.sans.org/reading-room/whitepapers/forensics/ioc-indicators-compromise-malware-forensics-34200`

Inoue, D., Yoshioka, K., Eto, M., Hoshizawa, Y., and Nakao, K. Malware behavior analysis in isolated miniature network for revealing malware's network activity. In *2008 IEEE International Conference on Communications*, pages 1715–1721. IEEE, 2008.

Jezorek, M. and Kuntz, D. 2013. [Accessed on: 5 July 2016].
URL `http://www.irongeek.com/i.php?page=videos/derbycon3/2102-iocaware-actively-collect-compromise-indicators-and-test-your-entire-enterprise-matt-jezorek-dennis-kuntz`

Johnson, C., Badger, L., and Waltermire, D. NIST Special Publication 800-150 (Draft) Guide to Cyber Threat Information Sharing (Draft), October 2014.

Kaspersky Lab. Kaspersky labs new malware count falls by 15,000 a day in 2015, as cybercriminals look to save money. December 2015.
URL `http://www.kaspersky.com/about/news/virus/2015/Kaspersky-Labs-New-Malware-Count-Falls-by-15000-a-Day-in-2015-as-Cybercriminals-Look-to-Save-Money`

Katsamakis, N. A Comparison of Dynamic Malware Analysis Systems and Security Information and Event Management systems for Malware Analysis. Master's thesis, Edinburgh Napier University, 2014.

Kendall, K. and McMillan, C. Practical malware analysis. In *Black Hat Conference, USA*. 2007. [Accessed on: 15 July 2015].
URL `https://www.blackhat.com/presentations/bh-dc-07/Kendall_McMillan/Paper/bh-dc-07-Kendall_McMillan-WP.pdf`

Kirk, R. Threat sharing–a neighbourhood watch for security practitioners. *Network Security*, 2015(12):5–7, 2015.

Lee, J., Jeong, K., and Lee, H. Detecting metamorphic malwares using code graphs. In *Proceedings of the 2010 ACM symposium on applied computing*, pages 1970–1977. ACM, 2010.

Lengyel, T. K., Maresca, S., Payne, B. D., Webster, G. D., Vogl, S., and Kiayias, A. Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 386–395. ACM, 2014.

Mandiant. Sophisticated Indicators for the Modern Threat Landscape: An Introduction to OpenIOC. Technical report, 2012. [Accessed on: 3 March 2016].
URL http://openioc.org/resources/An_Introduction_to_OpenIOC.pdf

Marie, F. New netfilter matches. April 2005.
URL http://www.netfilter.org/documentation/HOWTO/netfilter-extensions-HOWTO-3.html

Mehra, P. A brief study and comparison of snort and bro open source network intrusion detection systems. *International Journal of Advanced Research in Computer and Communication Engineering*, 1(6):383–386, 2012.

Microsoft. Ransomware. 2015. [Accessed on: 17 May 2016].
URL https://www.microsoft.com/security/portal/mmpc/shared/ransomware.aspx

MITRE Corporation. The MAEC Language. 2014. [Accessed on: 3 April 2016].
URL https://maec.mitre.org/

MITRE Corporation. Observable Type CYBOX CORE SCHEMA. 2015. [Acessed on: 1 November 2015].
URL http://stixproject.github.io/data-model/1.2/cybox/ObservableType/

MITRE Corporation. About CybOX. 2016a. [Accessed on: 3 April 2016].
URL http://cyboxproject.github.io/about/

MITRE Corporation. About TAXII. 2016b. [Accessed on: 3 April 2016].
URL http://taxiiproject.github.io/about/

MITRE Corporation. Regular expression support. 2016c. [Accessed on: 8 June 2016].
URL http://cyboxproject.github.io/documentation/regex-support/

MITRE Corporation. Use cases. 2016d. [Accessed on: 8 March 2016].
URL http://stixproject.github.io/usecases/

Mockapetris, P. Rfc 1035 domain names implementation and specification, november 1987. 1987. [Accessed on: 19 May 2016].
URL http://www.ietf.org/rfc/rfc1035.txt

NCI Agency. Malware information sharing platform. Technical report, NCI Agency, 2013. [Accessed on: 3 April 2016].

URL https://www.ncia.nato.int/Documents/Agency%20publications/Malware%20Information%20Sharing%20Platform%20(MISP).pdf

Newman, D. Benchmarking terminology for firewall performance. 1999. [Accessed on: 19 May 2016].
URL https://tools.ietf.org/html/rfc2647

O'Brien, D. Dridex: Tidal waves of spam pushing dangerous financial Trojan. White paper, Symantec, February 2016. [Accessed on: 3 August 2016].
URL http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/dridex-financial-trojan.pdf

Obrst, L., Chase, P., and Markeloff, R. Developing an ontology of the cyber security domain. In *Semantic Technology for Intelligence, Defense, and Security (STIDS)*, pages 49–56. 2012.

Oktavianto, D. and Muhardianto, I. Cuckoo Malware Analysis. 9781782169246-029. Packt Publishing Ltd, 2013.

Packard, H. Threat Central. June 2016. [Accessed on: 15 July 2016].
URL http://www8.hp.com/us/en/software-solutions/cyber-threat-analysis/

Panda Security. Pandalabs neutralized 75 million new malware samples in 2014, twice as many as in 2013. March 2015.
URL http://www.pandasecurity.com/mediacenter/press-releases/pandalabs-neutralized-75-million-new-malware-samples-2014-twice-many-2013/

Postel, J. and Reynolds, J. Rfc 959: File transfer protocol. 1985.

Provataki, A. and Katos, V. Differential malware forensics. *Digital Investigation*, 10(4):311–322, 2013.

Purdy, G. N. Linux iptables pocket reference. O'Reilly Media, Inc, 2004.

Qiao, Y., Yang, Y., He, J., Tang, C., and Liu, Z. Cbm: free, automatic malware analysis framework using api call sequences. In *Knowledge Engineering and Management*, volume 214, pages 225–236. Springer, 2014.

Raber, J. and Laspe, E. Deobfuscator: An automated approach to the identification and removal of code obfuscation. In *14th Working Conference on Reverse Engineering (WCRE 2007)*, pages 275–276. IEEE, 2007.

Ranjan, S. Detecting DNS fast-flux anomalies. September 4 2012. US Patent 8,260,914.

Roesch, M. Snort: Lightweight intrusion detection for networks. In *LISA '99 Proceedings of the 13th USENIX conference on System administration*, volume 99, pages 229–238. 1999.

Rossow, C., Dietrich, C. J., Bos, H., Cavallaro, L., Van Steen, M., Freiling, F. C., and Pohlmann, N. Sandnet: Network traffic analysis of malicious software. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pages 78–88. ACM, 2011.

Rudman, L. and Irwin, B. Dridex: analysis of the traffic and automatic generation of IOCs. In *ISSA 2016:15th International Information Security for South Africa Conference*, volume 15. ISSA, 2016a.

Rudman, L. and Irwin, B. A sharing platform for indicators of compromise. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC) 2016*, pages 266–271. SATNAC, September 2016b.

Saidi, H., Porras, P., and Yegneswaran, V. Experiences in malware binary deobfuscation. The 20th Virus Bulletin International Conference, 2010.

Sanghavi, M. DRIDEX and how to overcome it. Blog Post, March 2015. [Accessed on: 23 October 2015].
URL `http://www.symantec.com/connect/blogs/dridex-and-how-overcome-it`

Santos, I., Penya, Y. K., Devesa, J., and Bringas, P. G. N-grams-based file signatures for malware detection. *International Conference on Enterprise Information Systems (ICEIS)*, 9:317–320, 2009.

Savage, K., Coogen, P., and Lau, H. The evolution of ransomware. Technical report, Symantec, August 2015. [Accessed on: 19 May 2016].
URL `http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-evolution-of-ransomware.pdf`

SecurityWeek News. Dridex botnet spreading locky ransomware via javascript attachments. News Article, March 2016. [Accessed on: 29 April 2016].
URL `http://www.securityweek.com/dridex-botnet-spreading-locky-ransomware-javascript-attachments`

Segura, J. Deceiving cpanel account suspended page serves exploits. February 2015. [Accessed on: 20 July 2016].
URL https://blog.malwarebytes.com/threat-analysis/2015/02/deceiving-cpanel-account-suspended-page-serves-exploits/

Seok, J., Choi, M., Kim, J., and Park, J. A Comparative Study on Performance of Open Source IDS/IPS Snort and Suricata. *Journal of the Korea Society of Digital Industry and Information Management*, 12(1):89–95, 2016.

Sharif, M., Yegneswaran, V., Saidi, H., Porras, P., and Lee, W. Eureka: A framework for enabling static malware analysis. In *Computer security-ESORICS 2008*, pages 481–500. Springer, 2008.

Sinitsyn, F. Teslacrypt 2.0 disguised as cryptowall. July 2015. [Accessed on: 13 October 2016].
URL https://securelist.com/blog/research/71371/teslacrypt-2-0-disguised-as-cryptowall/

Smith, E. Snort vs Suricata. 2015. [Accessed on: 23 March 2016].
URL http://wiki.aanval.com/wiki/Snort_vs_Suricata

Speed Guide. Ports database. 2016. [Acessed on: 19 July 2016].
URL http://www.speedguide.net/ports.php

Stalmans, E. and Irwin, B. A framework for DNS based detection and mitigation of malware infections on a network. In *2011 Information Security for South Africa*, pages 1–8. IEEE, 2011.

Struse, R. OASIS Cyber Threat Intelligence (CTI) TC. 2016. [Accessed on: 16 July 2016].
URL https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cti

Suricata. Features. 2015a. [Accessed on: 23 March 2016].
URL http://suricata-ids.org/features/

Suricata. Suricata. 2015b. [Accessed on: 23 March 2016].
URL http://suricata-ids.org/

Svajcer, V. Building a malware lab in the age of big data. October 2015.

Sykosch, A. and Wubbeling, M. STIX 2 IDS. *Internet Architecture Board*, 2015.

Teo, L. Learning from the Dridex Malware - Adopting a Effective Strategy. White paper, SANS Institute, October 2015. [Accessed on: 4 April 2016].
URL https://www.sans.org/reading-room/whitepapers/detection/learning-dridex-malware-adopting-effective-strategy-36397

The Bro Project. The bro network security monitor. 2014. [Accessed on: 4 April 2016].
URL https://www.bro.org/

The Bro Project. Bro logging. 2016a. [Accessed on: 4 April 2016].
URL https://www.bro.org/sphinx/logs/index.html

The Bro Project. Signature framework. 2016b. [Accessed on: 4 April 2016].
URL https://www.bro.org/sphinx/frameworks/signatures.html

ThreatConnect. Threatconnect takes signature management to the next level. 2013. [Accessed on: 4 April 2016].
URL https://www.threatconnect.com/threatconnect-takes-signature-management-next-level/

Trend Micro. The history of ransomware: From CryptoLocker to Onion. August 2014. [Accessed on: 17 May 2016].
URL http://blog.trendmicro.com/the-history-of-ransomware-from-cryptolocker-to-onion/

Trend Micro. FBI, Security Vendors Partner for DRIDEX Takedown. Blog Post, October 2015. [Accessed on: 23 October 2015].
URL http://blog.trendmicro.com/trendlabs-security-intelligence/us-law-enforcement-takedown-dridex-botnet/

Trend Micro. Ransomware. 2016. [Accessed on: 17 May 2016].
URL http://www.trendmicro.com/vinfo/us/security/definition/Ransomware

ul-hassan Shirazi, N., Schaeffer-Filho, A., and Hutchison, D. Attack pattern recognition through correlating cyber situational awareness in computer networks. In *Cyberpatterns*, pages 125–134. Springer, 2014.

UpGuard. Top Free Network-Based Intrusion Detection Systems (IDS) for the Enterprise. 2015. [Accessed on: 15 July 2016].
URL https://www.upguard.com/articles/top-free-network-based-intrusion-detection-systems-ids-for-the-enterprise

US-CERT. Alert (TA15-286A) Dridex P2P Malware. Online Article, October 2015. [Accessed on: 23 October 2015].
URL `https://www.us-cert.gov/ncas/alerts/TA15-286A`

van Zyl, I., Rudman, L., and Irwin, B. A review of current DNS TTL practices. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC) 2015*, pages 131–136. 2015.

West, A. G. and Mohaisen, A. Metadata-driven threat classification of network endpoints appearing in malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, 978-3-642-22424-9, pages 152–171. Springer, 2014.

Willems, C., Holz, T., and Freiling, F. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, (2):32–39, 2007.

Zheng, D. E. and Lewis, J. A. Cyber threat information sharing recommendations for congress and the administration. March 2015. [Accessed on: 17 May 2016].
URL `http://csis.org/files/publication/150310_cyberthreatinfosharing.pdf`

Zorz, Z. Dridex botnet alive and well, now also spreading ransomware. Online Article, February 2016a. [Accessed on: 29 April 2016].
URL `https://www.helpnetsecurity.com/2016/02/17/dridex-botnet-alive-and-well-now-also-spreading-ransomware/`

Zorz, Z. Dridex botnet hacked, delivers dummy file. Online Article, May 2016b. [Accessed on: 6 May 2016].
URL `https://www.helpnetsecurity.com/2016/05/05/dridex-botnet-hacked/`

Appendix A

System Configuration

This appendix contains code listings of the changes needed to configure the Cuckoo Sandbox in Chapter 3.

Listing 1 auxilary.conf

```
[sniffer]
enabled = yes
tcpdump = /mnt/datapool/lauren/tcpdump
```

Listing 2 cuckoo.conf

```
[cuckoo]
machinery = virtualbox

[routing]
route = internet
internet = eth0

[resultserver]
ip = 146.231.133.164

[timeouts]
default = 1200
critical = 1230
vm_state = 60
```

Listing 3 processing.conf

```
[netwioc-proc]
enabled=yes
```

Listing 4 reporting.conf

```
[netwioc-rep]
enabled=yes
```

Listing 5 virtualbox.conf

```
[virtualbox]
machines = Windows7USP1

[Windows7USP1]
label = Windows7USP1
ip = 146.231.133.174
snapshot = SnapWithFiles
interface = eth0
```

Appendix B

IOC Generation and Sharing

This appendix contains a listing and table for Chapter 4.

Listing 6 Example of a tshark filter automatically generated for the custom Cuckoo processing module

```
#!/bin/bash
tshark -r $1 -w $2 -F pcap -Y "dns or http or tcp or icmp or
smtp or udp and not (ip.src == 146.231.133.190) and not
(ip.dst == 146.231.133.190) and not
(ip.src == 146.231.133.255) and not
(ip.dst == 146.231.133.255) and not
(ip.src == 146.231.133.164) and not
(ip.dst == 146.231.133.164) and not
(ip.src == 146.231.133.174) and not
(ip.dst == 146.231.133.174)"
```

Table B.1: OpenIOC Email vs. CybOX EmailMessageObj

OpenIOC Email	CybOX EmailMessageObj
Attachment/Name	Attachments/AttachmentsType/FAttachmentReferenceType->FileObjectType/File_Name
Attachment/SizeInBytes	Attachments/AttachmentsTypeAttachmentReferenceType->FileObjectType/Size_In_Bytes
BCC	Header/EmailHeaderType/BCC
Body	Raw_Body
CC	Header/EmailHeaderType/CC
Content-Type	Header/EmailHeaderType/Content_Type
Date	Header/EmailHeaderType/Date
From	Header/EmailHeaderType/From
In-Reply-To	Header/EmailHeaderType/In_Reply_To
MIME-Version	Header/EmailHeaderType/MIME_Version
Received	EmailReceivedLineType/Timestamp
ReceivedFromHost	EmailReceivedLineType/From
ReceivedFromIP	EmailReceivedLineType/From
Subject	Header/EmailHeaderType/Subject
To	Header/EmailHeaderType/To
	Email_Server
	Raw_Header
	Links
	AttachmentsType/File
	Header/EmailHeaderType/Message_ID
	Header/EmailHeaderType/Recieved_Lines
	Header/EmailHeaderType/Sender
	Header/EmailHeaderType/Reply_To
	Header/EmailHeaderType/Errors_To
	Header/EmailHeaderType/Boundary
	Header/EmailHeaderType/Precedence
	Header/EmailHeaderType/User_Agent
	Header/EmailHeaderType/X_Mailer
	Header/EmailHeaderType/X_Originating_IP
	Header/EmailHeaderType/X_Priority
	Header/EmailRecipientsType/Recipient
	LinksType/Link
	EmailReceivedLineType/By
	EmailReceivedLineType/Via
	EmailReceivedLineType/With
	EmailReceivedLineType/For
	EmailReceivedLineType/ID
	EmailReceivedLineListType/Recieved
	Attachments/AttachmentReferenceType/Qname
	LinkReferenceType/Qname
Attachment/MIMEType	
Attachment/AttachmentCount	
References	
Return-Path	
Thread-Index	
Thread-Topic	
X-MS-Has-Attach	
X-filenames	
X-filesizes	
X-filetypes	

Appendix C

Traffic Filters

C.1 Traffic Filtering code using `tshark`

Listing 7 Filter used to get resolved IP addresses

```
os.system("tshark -r "+folderPath+"/cut-byprocessingmodule.pcap -Y dns.flags.response==1
    -T fields -e dns.qry.name -e dns.a -E separator=, > "+folderPath+"/domains-SUS-IPs.
    csv")
```

Listing 8 Filter used to get ICMP packet information

```
os.system('tshark -r '+folderPath+'/cut-byprocessingmodule.pcap -Y icmp -T fields -e icmp
    .type -e ip.src -e ip.dst -E separator=, > '+folderPath+'/ICMPInfo.csv')
```

Listing 9 Filter used to get TCP SYN packet information

```
os.system("tshark -r "+folderPath+"/cut-byprocessingmodule.pcap -Y 'tcp.flags.syn==1 and
    tcp.flags.ack==0 and tcp.flags.cwr==0 and tcp.flags.ecn==0 and tcp.flags.fin==0 and
    tcp.flags.ns==0 and tcp.flags.push==0 and tcp.flags.res==0 and tcp.flags.reset==0 and
     tcp.flags.urg==0' -T fields -e ip.src -e ip.dst -e tcp.srcport -e tcp.dstport -E
    separator=, > "+folderPath+"/SYNConn.csv")
```

Listing 10 Filter used to get UDP connection information

```
os.system('tshark -r '+folderPath+'/cut-byprocessingmodule.pcap -Y udp -T fields -e udp.
    srcport -e udp.dstport -e ip.dst -e ip.src -E separator=, > '+folderPath+'/UDPInfo.
    csv')
```

Listing 11 Filter to get HTTP GET information

```
os.system('tshark -r '+folderPath+'/cut-byprocessingmodule.pcap -Y http.request.method=="
    GET" -T fields -e http.request.method -e http.request.uri -e http.request.version -e
    tcp.dstport -e http.accept -e http.accept_language -e http.accept_encoding -e http.
    authorization -e http.cache_control -e http.connection -e http.cookie -e http.
    content_length -e http.content_type -e http.date -e http.host -e http.
    proxy_authorization -E separator=~ > '+folderPath+'/HTTPFullGET.csv')
```

Listing 12 Filter to get DNS request information

```
os.system('tshark -r'+folderPath+'/cut-byprocessingmodule.pcap -Y dns.flags.response==0 -
    T fields -e ip.src -e udp.srcport -e ip.dst -e udp.dstport -e dns.qry.name -e dns.qry
    .type -e dns.flags.response -E separator=~ > '+folderPath+'/DNSInfo.csv')
```

Listing 13 Filter to get FTP information

```
os.system('tshark -r '+folderPath+'/cut-byprocessingmodule.pcap -Y ftp -T fields -e ip.
    src -e tcp.srcport -e ip.dst -e tcp.dstport -e ftp.response.code -e ftp.request.
    command -e ftp.request.arg -e ftp.response.arg -E separator=, > '+folderPath+'/
    FTPInfo.csv')
```

Listing 14 Filter used to get SSH information

```
os.system('tshark -r '+folderPath+'/cut-byprocessingmodule.pcap \
-Y ssh -T fields -e ip.src -e tcp.srcport -e ip.dst -e tcp.dstport \
-e tcp.flags.syn -e tcp.flags.ack -e ssh.kexdh.host_key \
-E separator=, > '+folderPath+'/SSHInfo.csv')
```

Appendix D

Evaluations

Listing 18 Python function extracted from the script used for STIX to Iptables rules

```
1  def TCPtoIptables(tcplist):
2      out = tcplist[2] == "out"
3      if out:
4          return 'Iptables -A FORWARD -j LOG_AND_REJ -p tcp {0} {1} {2} {3} \n'.format("--dport" if out else "--sport",
               tcplist[0], "-d" if out else "-s", tcplist[1])
5      else:
6          return 'Iptables -A FORWARD -j LOG_AND_REJ -p tcp {0} {1} {2} {3} \n'.format("--dport" if out else "--sport",
               tcplist[0], "-d" if out else "-s", tcplist[1])
7
8  def HTTPReqtoIptables(httplist):
9      return 'Iptables -A FORWARD -p tcp --dport {0} -d {2} -m string --algo bm --string "{1}" -j LOG_AND_REJ\n'.format(
           httplist[6], httplist[1], httplist[5]) #LOG --log-prefix "Suspicious HTTP requests"\n'.format(httplist[6],
           httplist[5]+httplist[1])
10
11 def UDPtoIptables(udplist):
12     out = udplist[2] == "out"
13     if out:
14         return 'Iptables -A FORWARD -j LOG_AND_REJ -p udp {0} {1} {2} {3} \n'.format("--dport" if out else "--sport",
               udplist[0], "-d" if out else "-s", udplist[1])
15     else:
16         return 'Iptables -A FORWARD -j LOG_AND_REJ -p udp {0} {1} {2} {3} \n'.format("--dport" if out else "--sport",
               udplist[0], "-d" if out else "-s", udplist[1])
17
18 def DNStoIptables(dnslist):
19     dns = dnslist[0].split('.')
20     dns = dns[-2]+"|0"+str(len(dns[-1]))+"|"+dns[-1]+"|00|"
21     return 'Iptables -A FORWARD -p udp -m udp --dport 53 -m u32 --u32 "28 & 0xF8 = 0" -m string --algo bm --from 40 --
           hex-string "{1}" -j LOG_AND_REJ\n'.format(dnslist[1], dns)
```

Listing 15 A DNS Request STIX indicator for Dridex

```
<stix:Indicator id="example:indicator-c01943e2-4ae8-4399-820d-f202da452e9b" timestamp="2016-05-04T17:11:21.551479+00:00" xsi:type
    ='indicator:IndicatorType'>
 <indicator:Title>DNS Request</indicator:Title>
 <indicator:Description>An indicator containing information about a DNS Request</indicator:Description>
  <indicator:Observable id="example:Observable-e7aa842a-1ed7-44cf-97a9-a564d06a5455">
   <cybox:Object id="example:NetworkConnection-6434a124-db6e-4d84-8e02-9f4daae6692f">
    <cybox:Properties xsi:type="NetworkConnectionObj:NetworkConnectionObjectType">
     <NetworkConnectionObj:Layer3_Protocol>IPv4</NetworkConnectionObj:Layer3_Protocol>
     <NetworkConnectionObj:Layer4_Protocol>UDP</NetworkConnectionObj:Layer4_Protocol>
     <NetworkConnectionObj:Layer7_Protocol>DNS</NetworkConnectionObj:Layer7_Protocol>
     <NetworkConnectionObj:Destination_Socket_Address xsi:type="SocketAddressObj:SocketAddressObjectType">
      <SocketAddressObj:IP_Address xsi:type="AddressObj:AddressObjectType">
         <AddressObj:Address_Value>146.231.129.97</AddressObj:Address_Value>
      </SocketAddressObj:IP_Address>
      <SocketAddressObj:Port xsi:type="PortObj:PortObjectType">
         <PortObj:Port_Value>53</PortObj:Port_Value>
         <PortObj:Layer4_Protocol>UDP</PortObj:Layer4_Protocol>
      </SocketAddressObj:Port>
     </NetworkConnectionObj:Destination_Socket_Address>
     <NetworkConnectionObj:Layer7_Connections>
      <NetworkConnectionObj:DNS_Query xsi:type="DNSQueryObj:DNSQueryObjectType">
       <DNSQueryObj:Question>
        <DNSQueryObj:QName xsi:type="URIObj:URIObjectType">
         <URIObj:Value>ho7rcj6wucosa5bu.tor2web.org</URIObj:Value>
        </DNSQueryObj:QName>
        <DNSQueryObj:QType>A</DNSQueryObj:QType>
       </DNSQueryObj:Question>
      </NetworkConnectionObj:DNS_Query>
     </NetworkConnectionObj:Layer7_Connections>
    </cybox:Properties>
   </cybox:Object>
  </indicator:Observable>
 <indicator:Producer>
  <stixCommon:Time>
   <cyboxCommon:Produced_Time>2016-05-04T17:11:21.555631+00:00</cyboxCommon:Produced_Time>
  </stixCommon:Time>
 </indicator:Producer>
 </stix:Indicator>
```

Listing 16 A HTTP Request STIX indicator for Dridex

```
<stix:Indicator id="example:indicator-31adc014-e3d0-4893-8299-c6a407b418d1" timestamp="2016-05-05T08:15:24.464132+00:00" xsi:type
    ='indicator:IndicatorType'>
 <indicator:Title>HTTP request</indicator:Title>
 <indicator:Description>An indicator containing information about a HTTP request</indicator:Description>
 <indicator:Observable id="example:Observable-c01e5925-4c33-451e-b179-bde7e577cc6f">
  <cybox:Object id="example:NetworkConnection-1ff8499c-6f90-49af-b876-f9615be8d5dd">
   <cybox:Properties xsi:type="NetworkConnectionObj:NetworkConnectionObjectType">
    <NetworkConnectionObj:Layer3_Protocol>IPv4</NetworkConnectionObj:Layer3_Protocol>
    <NetworkConnectionObj:Layer4_Protocol>TCP</NetworkConnectionObj:Layer4_Protocol>
    <NetworkConnectionObj:Layer7_Protocol>HTTP</NetworkConnectionObj:Layer7_Protocol>
    <NetworkConnectionObj:Layer7_Connections>
     <NetworkConnectionObj:HTTP_Session xsi:type="HTTPSessionObj:HTTPSessionObjectType">
      <HTTPSessionObj:HTTP_Request_Response>
       <HTTPSessionObj:HTTP_Client_Request>
        <HTTPSessionObj:HTTP_Request_Line>
         <HTTPSessionObj:HTTP_Method>GET</HTTPSessionObj:HTTP_Method>
         <HTTPSessionObj:Value>/wp-content/plugins/cached_data/k1.exe</HTTPSessionObj:Value>
         <HTTPSessionObj:Version>HTTP/1.0</HTTPSessionObj:Version>
        </HTTPSessionObj:HTTP_Request_Line>
        <HTTPSessionObj:HTTP_Request_Header>
         <HTTPSessionObj:Parsed_Header>
          <HTTPSessionObj:Accept>*/*</HTTPSessionObj:Accept>
          <HTTPSessionObj:Accept_Language>en-US</HTTPSessionObj:Accept_Language>
          <HTTPSessionObj:Accept_Encoding>identity, *;q=0</HTTPSessionObj:Accept_Encoding>
          <HTTPSessionObj:Connection>close</HTTPSessionObj:Connection>
          <HTTPSessionObj:Host>
           <HTTPSessionObj:Domain_Name xsi:type="URIObj:URIObjectType">
            <URIObj:Value>nerdmeetsgirl.com</URIObj:Value>
           </HTTPSessionObj:Domain_Name>
           <HTTPSessionObj:Port xsi:type="PortObj:PortObjectType">
            <PortObj:Port_Value>80</PortObj:Port_Value>
           </HTTPSessionObj:Port>
          </HTTPSessionObj:Host>
         </HTTPSessionObj:Parsed_Header>
        </HTTPSessionObj:HTTP_Request_Header>
       </HTTPSessionObj:HTTP_Client_Request>
      </HTTPSessionObj:HTTP_Request_Response>
     </NetworkConnectionObj:HTTP_Session>
    </NetworkConnectionObj:Layer7_Connections>
   </cybox:Properties>
  </cybox:Object>
 </indicator:Observable>
 <indicator:Producer>
  <stixCommon:Time>
   <cyboxCommon:Produced_Time>2016-05-05T08:15:24.464597+00:00</cyboxCommon:Produced_Time>
  </stixCommon:Time>
 </indicator:Producer>
</stix:Indicator>
```

Listing 17 A TCP STIX indicator for Dridex

```
<stix:Indicator id="example:indicator-d3aaf099-5cb7-4f2d-8b6e-8572738d0a3b" timestamp="2016-05-05T08:15:24.459987+00:00" xsi:type='indicator:Indicat
 <indicator:Title>TCP Connection Established</indicator:Title>
 <indicator:Description>An indicator containing information about a successful TCP hand shake</indicator:Description>
 <indicator:Observable id="example:Observable-2b2c557e-86b9-4992-a07c-e2982e45baad">
  <cybox:Object id="example:NetworkConnection-cc12d0be-2d9b-485d-a479-e13551d3b369">
   <cybox:Properties xsi:type="NetworkConnectionObj:NetworkConnectionObjectType">
    <NetworkConnectionObj:Layer3_Protocol>IPv4</NetworkConnectionObj:Layer3_Protocol>
    <NetworkConnectionObj:Layer4_Protocol>TCP</NetworkConnectionObj:Layer4_Protocol>
    <NetworkConnectionObj:Source_TCP_State>ESTABLISHED</NetworkConnectionObj:Source_TCP_State>
    <NetworkConnectionObj:Destination_Socket_Address xsi:type="SocketAddressObj:SocketAddressObjectType">
     <SocketAddressObj:IP_Address xsi:type="AddressObj:AddressObjectType">
      <AddressObj:Address_Value>166.62.114.65</AddressObj:Address_Value>
     </SocketAddressObj:IP_Address>
     <SocketAddressObj:Port xsi:type="PortObj:PortObjectType">
      <PortObj:Port_Value>80</PortObj:Port_Value>
      <PortObj:Layer4_Protocol>TCP</PortObj:Layer4_Protocol>
     </SocketAddressObj:Port>
    </NetworkConnectionObj:Destination_Socket_Address>
   </cybox:Properties>
  </cybox:Object>
 </indicator:Observable>
 <indicator:Producer>
  <stixCommon:Time>
   <cyboxCommon:Produced_Time>2016-05-05T08:15:24.460507+00:00</cyboxCommon:Produced_Time>
  </stixCommon:Time>
 </indicator:Producer>
</stix:Indicator>
```

Listing 19 Python function extracted from the script used for STIX to Snort rules

```
def IPtoSNORT(iplist, sid, malware_name):
        return "alert IP $HOME_NET any -> "+iplist+' any (msg:"Suspicious IP address seen"; logto:"RulesFromSTIX.log";
             sid:'+str(sid)+';)\n'

def TCPPortIPtoSNORT(portIP, sid, protocol, malware_name):
        out = portIP[2] == "out"
        return "alert TCP  {0} (msg:\"Suspicious {1} {4} connection {2}\"; classtype:bad-unknown; sid:{3};)\n".format("
             $HOME_NET any -> "+portIP[1]+" "+str(portIP[0]) if out else portIP[1]+" "+str(portIP[0])+" -> $HOME_NET
             any", "outgoing" if out else "incoming", malware_name, str(sid), protocol)

def UDPPortIPtoSNORT(portIP, sid, protocol, malware_name):
        out = portIP[2] == "out"
        return "alert UDP {0} (msg:\"Suspicious {1} {4} connection {2}\"; classtype:bad-unknown; sid:{3};)\n".format("
             $HOME_NET any -> "+portIP[1]+" "+str(portIP[0]) if out else portIP[1]+" "+str(portIP[0])+" -> $HOME_NET
             any", "outgoing" if out else "incoming", malware_name, str(sid), protocol)

def DNStoSNORT(dnslist, sid, malware_name):
        dns = dnslist[0].split('.')
        dns = dns[-2]+"|0"+str(len(dns[-1]))+"|"+dns[-1]+"|00|"
        return "alert UDP $HOME_NET any -> any {2} (msg:\"Suspicious domain name request {3}\"; byte_test:1,!&,0xF8,2;
             content:\"{0}\"; fast_pattern:only; classtype:bad-unknown; sid:{1};)\n".format(dns, sid, dnslist[1],
             malware_name)


def HTTPReqtoSNORT(httplist, sid, malware_name):
        return "alert TCP $HOME_NET any -> any {0} (msg:\"Malicious {1} detected {2}\"; content:\"{5}\"; http_header;
             content:\"{3}\"; http_uri; nocase; sid:{4};)\n".format(httplist[6], "HTTP "+httplist[0]+" request",
             malware_name, httplist[1], sid, httplist[5])
```

Listing 20 TCP Snort Rule example

```
alert TCP $HOME_NET any -> 184.106.112.172 9001 (msg:"Suspicious outgoing TCP connection
    "; classtype:bad-unknown; sid:234501;)
```

Listing 21 HTTP Snort Rule example

```
alert TCP $HOME_NET any -> any 80 (msg:"Malicious HTTP POST request detected"; content:"
    chemes.eu"; http_header; content:"/wp-content/themes/decoy2/redux-framework/ReduxCore
    /inc/fields/info/2.php?c=68yccs1xca0m7tc"; http_uri; nocase; sid:234781;)
```

Listing 22 DNS Snort Rule example

```
alert UDP $HOME_NET any -> any 53 (msg:"Suspicious domain name request"; byte_test:1,!&,0
    xF8,2; content:"ip-addr|02|es|00|"; classtype:bad-unknown; sid:234529;)
```

Listing 23 TCP Iptables Rule example

```
sudo Iptables -A FORWARD -j LOG_AND_REJECT -p tcp --dport 9001 -d 184.106.112.172
```

Listing 24 HTTP Iptables Rule example

```
sudo Iptables -A FORWARD -p tcp --dport 80 -d xsso.sisttwtdogpyjlcc.info -m string --algo
     bm --string "/c8619ce03b36001f0d7c4258684707e5" -j LOG_AND_REJ
```

Listing 25 DNS Iptables Rule example

```
sudo Iptables -A FORWARD -p udp -m udp --dport 53 -m u32 --u32 "28 & 0xF8 = 0" -m string
    --algo bm --from 40 --hex-string "vsjlyjkxdewy|02|ru|00|" -j LOG_AND_REJ
```

Listing 26 API Upload to sharing platform with Curl

```
curl -v -F "file=@ioc.xml" -F "malware_name=API Example"
-F "api_key=65eab40bf1bcd5c82c6d9e02abea5ed3"
-F "description=Uploaded from API" http://localhost:5000/api/upload_ioc
```

Listing 27 API Download from sharing platform using Curl

```
curl -v -F "malware_name=Example upload"
-F "api_key=65eab40bf1bcd5c82c6d9e02abea5ed3"
-F "convert_type=snort" http://localhost:5000/api/download_ioc
> Example download.xml
```