UNIVERSITY OF TARTU
Institute of Computer Science
Cyber Security Curriculum

Ivo Pure

An automated methodology for validating web related cyber threat intelligence by implementing a honeyclient

Master's Thesis (30 ECTS)

Supervisor(s): Risto Vaarandi, phd
Raimundas Matulevicius, phd

Tartu 2019

An automated methodology for validating web related cyber threat intelligence by implementing a honeyclient

Abstract:

This paper is contributing to the open source cybersecurity community by providing an alternative methodology for analyzing web related cyber threat intelligence. Websites are used commonly as an attack vector to spread malicious content crafted by any malicious party. These websites become threat intelligence which can be stored and collected into corresponding databases. Eventually these cyber threat databases become obsolete and can lead to false positive investigations in cyber incident response. The solution is to keep the threat indicator entries valid by verifying their content and this process can be fully automated to keep the process less time consuming. The proposed technical solution is a low interaction honeyclient regularly tasked to verify the content of the web based threat indicators. Due to the huge amount of database entries, this way most of the web based threat indicators can be automatically validated with less time consumption and they can be kept relevant for monitoring purposes and eventually can lead to avoiding false positives in an incident response processes.

Keywords:

Cyber threat intelligence validation and automation, honeyclient, YARA, Javascript sandbox

CERCS: P170

Automatiseeritud metoodika veebipõhiste ohuteadmus indikaatorite valideerimiseks kasutades aktiivset meepurki

Lühikokkuvõte:

Loodud töö panustab küberkaitse valdkonda pakkudes alternetiivse viisi, kuidas hoida ohuteadmus andmebaas uuendatuna. Veebilehti kasutatakse ära viisina toimetada pahatahtlik kood ohvrini. Peale veebilehe klassifitseerimist pahaloomuliseks lisatakse see ohuteadmus andmebaasi kui pahaloomulise indikaatorina. Lõppkokkuvõtteks muutuvad sellised andmbaasid mahukaks ja sisaldavad aegunud kirjeid. Lahendus on automatiseerida aegunud kirjete kontrollimist klient-meepott tarkvaraga ning kogu protsess on täielikult automatiseeritav eesmärgiga hoida kokku aega. Jahtides kontrollitud ja kinnitatud indikaatoreid aitab see vältida valedel alustel küberturbe intsidentide menetlemist.

Võtmesõnad:

Ohuteadmus andmebaasi kirjete automaatne uuendamine, klient-meepott, YARA, Javascript liivakast

CERCS: P170

Table of Contents

Dictionary

Terms:

Honeyclient – Headless programmable browser modified to detect malicious web content

CTI – Cyber Threat Intelligence, a set of IOC's with context, also with correlation, aggregation capablilities

IOC – Indicator of compromise, threat indicator – indicates an attack or a compromise of some kind

Threat actors – An individual or a group of individuals targeting individuals or an organization with cyber threats

TTP - Tactics, Techniques and Procedures. Patterns of activities and methods incorporated with threat actors.

Web Proxy – An intercepting man in the middle solution, a gateway for outbound HTTP requests. Purpose: Privacy and security.

EK – Exploit Kit. An automated tool for automatically probing for and exploiting browser vulnerabilities.

Malvertisement – malware injected into the advertisement network and is spreaded through it.

Watering hole attacks – Only group of interest is targeted for infection, e.g certain bank costumers

CaaS – Cybercrime as a service

RaaS – Ransomware as a service

Threat Actor – Threat agent

OPSEC – Operation security

Acronyms:

HTTP - Hypertext Transfer Protocol

DB – Database

FW – Firewall

PCAP – Packet capture

DOM – Document object model

NGFW – Next generation firewall

GWAV – Gateway anti-malvare, i.e AV in Web proxy or Email proxy

SWG - Secure web gateway

CMS – Content management system

Zero-day attack – A type of attack which is not known for the public and there is not defense against it.

XSS – Cross site scripting attack

SOC - Security Operations Center

NOC – Network Operations Center

IDS – Intrusion detection system

IPS – Intrusion prevention system

URL - uniform resource locator

DGA – Domain generation algorithm

CERT - Computer Emergency Response Team

MISP – Malware Information Sharing platform

ATD – Advanced threat detection

VBA Script - Microsoft Visual Basic Scripting

C&C – Command and control server

MITM – Man in the middle attack

APT – Advanced persistent threat. Usually associated with governments

API - Application programming interface

RFC – Request for comments, Internet Engineering Task Force (IETF)

TLD – Top level domain

List of Figures

List of Tables

List of code snippets

1 Introduction

This thesis aims to design a methodology to validate known cyber threat intelligence database entries with a fast working honeyclient. Firstly, low interaction and high interaction already known solutions are studied and compared. Secondly, the author proposes a methodology with proposed principles to automate the entire process. Next, the methodology is implemented to a low interaction honeyclient. The honeyclient composes of already developed open source components and the authors contribution is the automation code. Lastly, for validation a large dataset is inspected to validate the detection principles and the time factor.

This paragraph describes the general background of malicious websites. Furthermore, it emphasizes the importance of taking in account that websites are one of the main attack vectors. Lastly, the problem statement is described and the general outline of the work is presented.

1.1 General background

The world wide web has become the most used internet service with the purpose of exchanging data between a user and server. There are currently 1,518,207,412 publicly hosted domains which actively respond to any client connecting to the server over the HTTP protocol [1]. Eventually, all of the world's web content is accessible over any internet friendly IoT[1] device, boosting the growth rate of the web even more.

Nowadays it is fairly easy to set up a website and to present the desired content. There are many CMS platforms, which require almost no need for programming skills and no special system administration knowledge. Accordingly, 44,8% of the publicly accessible websites use no CMS and the most popular CMS is Wordpress which is used by 33,2% of publicly accessible websites. Furthermore, Wordpess owns 60.1% of the entire CMS market share [2,4]. In conclusion, this increases the risk of poorly configured webservers, websites and the lack of security knowledge leaves these websites unpatched and vulnerable to exploitation.

Websites have become of the main attack vectors for delivering malicious payloads to the victim. They are also widely used in phishing campaigns and malvertising[2]. Generally, there are two types of malicious websites. Firstly, the ones that have been set up by adversaries directly. Secondly, unpatched websites taken over by adversaries exploiting a certain vulnerability. Furthermore, threats residing in benign websites are becoming more complex and the cloaking methods are constantly improving to avoid detection. In addition, the attack rate seems not to decrease, because the ratio between legitimate and malicious websites is increasing in favor of the benign websites. Hence, the ratio of websites found to be malicious in year 2016 was one of 13 and in year 2017 it was one of 20 [3]. Thus, websites are a very common attack surface with many possible attack vectors, making them an important concern amongst cyber security personnel.

In the past, exploit kits dominated the threat landscape amongst websites. Currently exploit kits are diminishing due to the fact that browsers and its add-ons are harder to exploit. For example, Google removed flash, java and adobe reader support, because they were the main components in an exploitation process to make it possible for an adversary to run benign

[1] https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT
[2] https://blog.malwarebytes.com/101/2015/02/what-is-malvertising/

code. Nevertheless, EK-s are still a common way to exploit a vulnerability in a web browser component, especially in IE where the previously mentioned plugins are still usable [4].

The focus of the attackers has shifted to E-mails as an attack vector. Statistics show that 92.4% of the malware is delivered via emails [4]. There are several ways to infect the victim with malware and each category is having their own tactics. The easiest way is just to send the payload itself in hopes that the user will open it. These kind of attacks are fairly easy detected by e-mail gateways and endpoints security solutions. Secondly, a specially crafted document, mostly Microsoft office Word has the ability to run VBA scripts. The most common way is to obfuscate the VBA code to avoid detection. Thereafter, in many cases the Trojan downloader enumerates the OS to gain information which exploit or payload it should download. Also, these kind of downloaders use most likely DGAs and simple URL blacklists are not sufficient.

Malvertising has become an important concern. The ad networks are huge and complex and this makes it hard to contain malicious activity. Many websites just embed the ad networks sources and if an attacker manages to advertise for example an exploit kit and embed a drive by download, the visiting victim of the websites is unaware of being exploited. Adversaries also use scaring tactics, scam and click baiting to initiate the execution of malicious code [4].

Directly or indirectly websites are part of an attacking process. To detect these kind of attack attempts, security solutions such as IDS, Web proxies with SSL interception capabilities, automatic sandboxing features and various monitoring solutions are used. To express it as a chain of activities, then this approach is limited in the cyber kill chain only on the intrusion parts. There after most commonly an incident response is triggered. However, to take a step further, the next phase is called active defense. This refers to the process of monitoring for and learning from malicious activity. The outcome of the learning process is stored in a CTI database. This last step before offense is also referred as active defense CTI gathering and validation, whereas the collected data is converted into re-usable information to detect similar attacks from the same adversaries [5]. Lastly, these databases need to be revalidated constantly to avoid any threat hunting or cyber threat incidence response based on false positive outdated threat intelligence. A low level rapid and automated solution can give a much quicker initial verdict than any other high level computer resource consuming ATD system.

1.2 Motivation

The motivation of this work is based on several key activities of a SOC analyst, how one collects threat information, exchanges it, applies it and revalidates the database.

Firstly, in different organizations and companies the SOC team is divided into different tiers and teams, based on the size and the need of it. Larger organizations usually follow general frameworks that have been approved by the larger community and they adopt them to their needs. One of the largest CERT[3] community is FIRST[4], which has a common goal to improve cybersecurity, share intelligence of how to detect malicious activities and help to increase the efficiency of incident response. Currently one of the focuses of SOC teams is to collect and share cyber threat intelligence [6]. The CTI databases are usually very vast and most of the IOC’s exchanged are outdated and invalid. An automated solution is required to actively re-validate value of the IOC’s that are stored in the CTI in the light of an incident

[3] https://www.ria.ee/en/cyber-security/cert-ee.html
[4] https://www.first.org/

response. The purpose is not to delete this precious information but to tag it with a timestamp and reference that the risk is mitigated or the threat does not exist any more.

Secondly, the most common approach to deal with malicious IOC’s is to create blacklists from them, with the purpose of preventing any kind of connection to or originating from them. With the amount of benign IOC’s growing daily their validity becomes questionable, since most of the malicious actors TTP’s define, that if compromised one needs to move on to new resources [8]. These resources are DGA generated domains or exploited valid websites. The average lifespan of an EK is twenty-four hours [7]. The speed of cleaning legitimate websites from malicious content varies, mostly depending on the service provider and website administrator. These websites are usually exploited and taken over by adversaries due to unpatched or not mitigated vulnerabilities or by malvertising. Nevertheless, these websites need to be accessed after the malicious content is removed and an automated solution can proactively probe and verify that.

In conclusion, the motivation is to automate the common steps of a SOC analyst in the scope of malicious activity delivered over the HTTP(s) protocol by keeping the CTI database up to date.

1.3 Problem statement

To emphasize the contribution of this work, the problem needs background information. Websites are not just attack surfaces, they are part of complex hacker TTP-s. They are used in different steps of the attack technique [8].

The goal of the study is to automate the analysis process of malicious actions and content delivered via the HTTP protocol. The outcome is meant to be a method which could be implemented in any high level programming language to create the automated software with the goal to assist the SOC analyst in revalidating IOC’s. In any SOC team nowadays gathering CTI is a must and exchanging it with regional partners gives insight what hacker groups are working in the area and which TTP-s they are using. Exchanging outdated and invalid CTI entries must be avoided. Furthermore, keeping the size of blacklists minimal and providing valid CTI database information is an important factor, since in incident response chasing adversaries based on outdated information can lead to chasing ghosts[5] in an incident response process.

The focus of the study are client side attacks, which require end user actions to trigger them. To achieve an acceptable ratio between the detection rate and web crawling speed, a crawling mechanism is needed with a minimal amount of signatures, which are meant to detect the common steps of an adversaries TTP. To further enhance the detection rates a weight mechanism with a scoring system could be introduced.

There are also several limitations to take into account. There are three types of attacks, which are not detectable by the proposed methodology:

- Geographic and location based
- Watering hole
- APT

[5] A term which referrers that one can search for nothing due to invalid indicators. https://www.fireeye.com/blog/executive-perspective/2014/05/ghost-hunting-with-anti-virus.html

The main constraint of using this methodology is to have a well maintained CTI database where threat entries have additional context. The goal is not to create a tool, which detects malicious websites but rather automatically validates them based on the malicious characteristics defined alongside the CTI entry. **Secondly**, CTI entries can be re-validated only if accessing the web-based indicator despite the originating IP address will not compromise the company's operation security concepts. **Thirdly**, to effectively create signatures based on the CTI entry context, network traffic capture of similar malicious events is required to validate the signatures detection capabilities.

Websites are constantly exploited and temporarily created for benign activities, therefore they are never fully secure and trustworthy. A well maintained CTI DB can reduce the rate of actions triggered by false positive events. An automated method based on the context of the CTI yields more effective results than any other alternative automated tool which does not know the context.

RQ1: How does proactive defense, particularly using client side honeypots, help to detect malicious characteristics from websites?

RQ2: How to validate cyber threat intelligence by adopting the honeyclient principles with the goal of keeping the web based indicators of compromise relevant?

RQ3: How to implement the methodology using open source components?

RQ4: How effective is a context oriented fully automated CTI validating honeyclient?

1.4 Outline

The thesis consists of **6** main chapters:

1. Introduction
2. Background and related work – describes different low interaction and high interaction honeyclients
3. A methodology for revalidating cyber threat intelligence – defines the process of automatically validating CTI
4. Implementing the methodology to a honeyclient – proposes a technical architecture composing of open source components
5. Analysis of the results and validation – Introduces context oriented playbooks tested against malicious data sources
6. Conclusion and future work – possible features added to the honeyclient software to furthermore boost its value

2 Background and related work

This subject explains the general Cybercrime landscape, emphasizes to danger lurking from websites and gives insight how honeyclients operate on different levels. The answer the following research question is discussed in this subject: How does proactive defense, particularly using client side honeypots, help to detect malicious characteristics from websites?

2.1 Cybercrime attack trends shifting

Before the boom of cryptocurrencies, the economy of cybercrime was focused on fraud, such as credit card information stealing. Thanks to anonymous ways to gain profit the criminal market got a huge boost. The payloads which attackers started to deliver were Ransomware variants with the single goal of encrypting the victim's data and demanding ransom. All this started in year 2013 and reached their height in 2016. Furthermore, in 2017 the emergence of Wannacry and NotPetya shuttered the digital world. It is defined by Symantec Internet security report, that they were actually a masquerade of ransomware and the goal of these targeted attacks was total destruction of data. The family of this destructive payload is called Cryptowall. It does not encrypt any files, instead overwrites all files by random data. Hence, making Cryptowall a total disk wiper, disguised as ransomware. The true purpose was to steal all data and thereafter wipe everything, including himself and the digital footprint of this benign action. For example, Phoneywall disguised itself as ransomware but was silently a data exfiltration tool. Eventually the data got stolen and organizations stopped investigations due to not knowing the real objective of the payload. Currently, the boom has stopped and Ransomware holds a steady line. Furthermore, the new trend is coinmining, which essentially means using the computing power of others to earn virtual money. Commonly they are implanted to exploited servers or injected into a websites code so that the visiting user's computer would start mining for the coins [3].

Web browser developing companies have put in serious countermeasures and mitigations for EK's, therefore the usage of them has drastically decreased. For example, Chrome removed Java and disabled Flash plugins, which are most commonly exploited. Nonetheless, this does not mean EK's are gone, as always in the cyber economy they are rearranging themselves to adopt to the new landscape [4].

Nowadays it is possible for every average user to become a cybercriminal, due to the fact that cybercrime as a service (CaaS) has rapidly evolved. In dark market forums it is possible to buy ransomware as a service (RaaS) [12].

To sum it up, irrespectively to which payloads are delivered, the motives of attackers and the attack vectors remain the same. The ultimate goal of the threat actor is to earn profit at the expense of someone's business costs.

2.2 Websites as threat surface

The threat surface for an attacker is any vulnerable web browser visiting the malicious website, hence the websites themselves become part of the attack vector. Generally, there are three[6] categories of websites which are used in attack campaigns. The first group of websites

[6] Three malicious website categories defined by Trendmicro. https://www.trendmicro.com/vinfo/us/security/definition/exploit-kit

are temporarily created by attackers, the second group contains websites which have been taken over because of a successful exploitation and thirdly websites which malvertise unwillingly. Nevertheless, all three types are used in an attack process, mostly according to the attackers TTP.

The attack vectors of a malicious actor have not changed for a decade [9]. For example, persistent XSS and command injection flaws were actively abused in the past. The abusing of these vulnerabilities has not disappeared, instead the payload served after a successful attack and the way it is distributed to the victim, has become drastically more complex to detect and analyze. More precisely, cloaking methods such as multi-layer obfuscation and Javascript packers are being constantly improved to avoid detection [13]. OWASP[7] Top 10 frequently keeps track of the most exposed security risks found on web applications. It is possible to generalize the security issues:

- Command and code injection flaws due to poor input validation or sanitization
- Vulnerable web applications[8]
- Access and security control flaws

Conclusively, taking over websites is the most desired way for a threat actor due to it is trusted and visited by many potential victims. It also makes these URL’s trustworthy when spamming them over mass emails and this way it also increases the success rate of the attacks [4, 14].

2.2.1 Drive-by downloads

Drive-by download is a technique used by malicious actors as an attack vector to deliver malicious code to the victim’s computer. There are two strategies behind this technique. Firstly, using exploit kits targeting specific vulnerabilities. These vulnerabilities can be browser memory overflow or heap spray vulnerabilities or alternatively the vulnerabilities of different API calls in plugins can be abused due to improper parameter handling [11, 15].

Essentially, there are two general ways how malware is served and delivered over the websites. Firstly, the victim visits a malicious website controlled by the attacker. Malicious actors in cyberspace exploit vulnerable websites and CMS systems to gain control over them or to plant their malicious code. These websites usually have code injection vulnerabilities or poor security and access control settings. For example, one can just upload and executable to a picture gallery and then after trick the user to download and execute it. Nevertheless, the malicious payload may be served directly over the exploited website. But, most commonly a malicious code is injected to redirect the attacker over a redirect chain to the landing page. This kind of redirection can occur due to hidden iframes in the background of the hacked website or through malvertising a malicious ad. However, the optional landing page is designed to profile the victim and it also performs several checks that it is really a human. In most cases, the landing page also checks the version of the browsers, OS and browser plugins including their versions. Afterwards, the victim is redirected to the website where the actual exploitation process takes place. All this happens in the background silently and the victim has no control [4, 11].

[7] OWASP top 10 topical security risks. https://www.owasp.org/index.php/Top_10-2017_Top_10
[8] Persistent & reflected XSS, DOM-Based XSS. https://excess-xss.com/

Secondly, cybercriminals use manipulative and deceiving ways to trick the users to click on malicious URL addresses. These links are mostly distributed via email spam, social networks and chat programs such as Skype. Most often decoys of authentic platforms are used, to trick the user clicking on the malicious link or a picture with a hidden link. These authentic platforms can be social media or any other popular web platforms. For example, an attacker crafts an identical message, such as the authentic platforms would send and using any kind of manipulative tactic to make the user click the URL in the spam email or the attachment containing a malicious URL. Furthermore, the links embedded in the attached documents are usually short URL's, which is an ideal tactic to bypass static URL whitelists. [3, 4, 11].

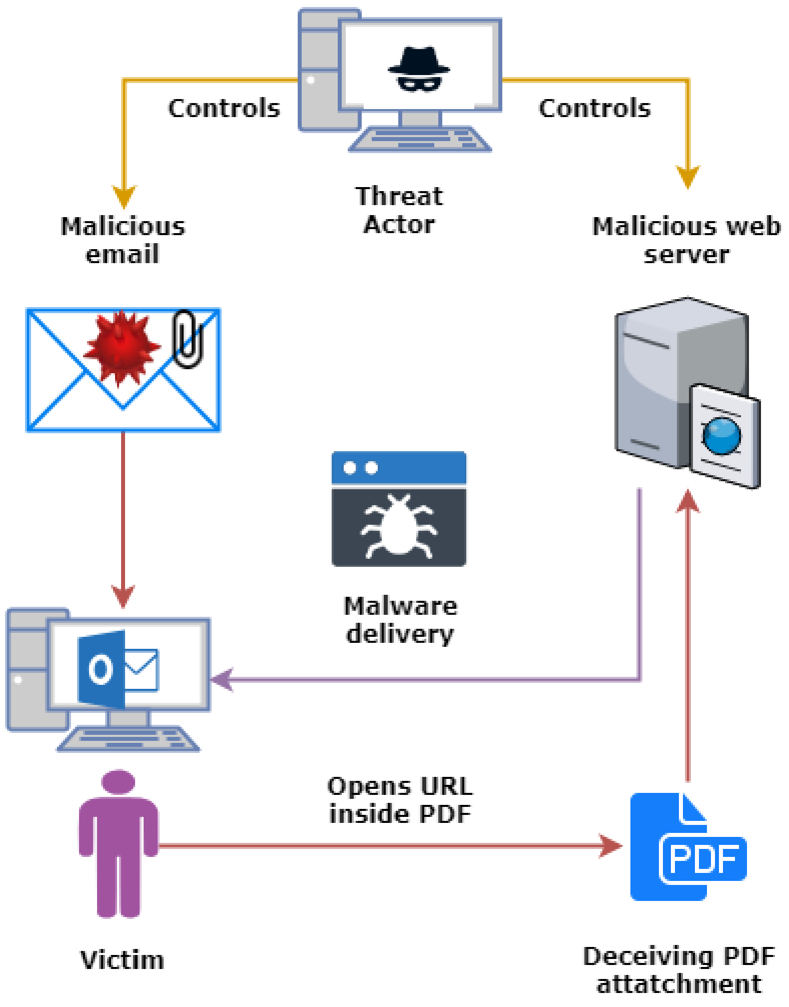


Figure 1. Example case of a drive-by-download.

In the **Figure 1**, a threat actor sends a deceiving PDF as an email attachment. This file contains a malicious short URL, which the user is tricked into clicking. Afterwards a Microsoft Office document is downloaded containing malicious VBA script. This script can contain exploit code to exploit MS Word vulnerabilities or download and execute further payloads. Alternatively, the short URL could also redirect the user to a browser EK landing page.

2.2.2 Exploit Kits

Exploiting browser vulnerabilities has been an issue for a decade. In fact, exploiting these vulnerabilities has a become part of automated tools, known as Exploit Kits. These kits exploit the victim's browser or its components vulnerabilities automatically. Most commonly exploited web technologies are: IE ActiveX, embedded Java and Flash objects and Javascript. In the **Figure 2**, firstly, the victim is redirected to a landing site via an HTML iframe tag, PHP code or Javascript redirect, which has been injected to the compromised website. In the first case, this HTML tag usually has the property "*visibility: none"* which makes the frame hidden enabling behind the scene redirections. Secondly, the victim's browser is probed for vulnerabilities by fingerprinting the browser itself, its version and the versions of the browser plugins. Secondly, if a vulnerability has been detected and exploited the victim's browser is redirected to the delivery server. And thirdly, the shellcode is delivered to the victim's browser for rendering.

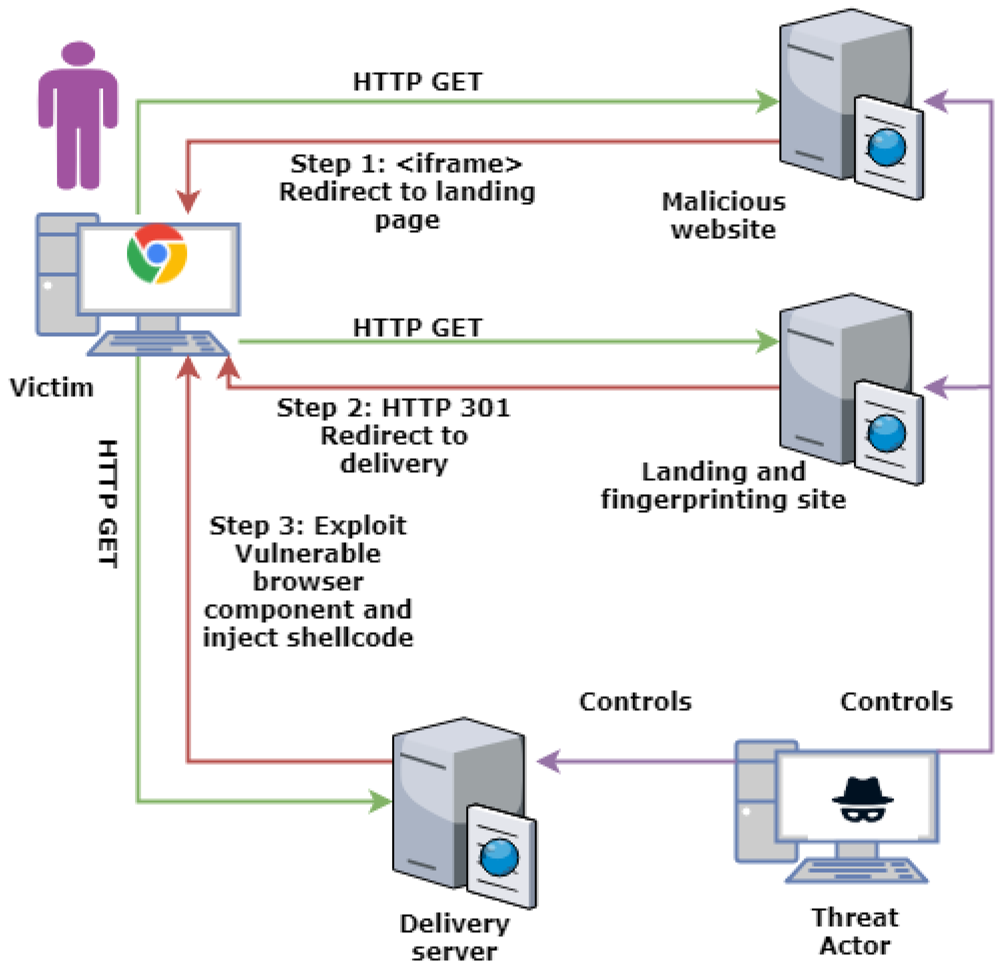

HTTP GET
Step 1: <iframe> Redirect to landing page
Malicious website
Victim
HTTP GET
Step 2: HTTP 301 Redirect to delivery
Landing and fingerprinting site
HTTP GET
Step 3: Exploit Vulnerable browser component and inject shellcode
Controls
Controls
Delivery server
Threat Actor

Figure 2. General example of the process of a victim visiting a malicious website and being exploited.

It is important to emphasize, that the EK will distribute a payload only if a certain vulnerability in a web browser component has been identified (**Figure 2 – Fingerprinting site**). Then a corresponding payload is sent to exploit the vulnerability. Furthermore, EK's are usually armed with several malicious payloads to increase the chance of a successful infiltration. The purpose of the initial payload is to deliver and execute shellcode. Firstly, after the browser vulnerability is detected a piece of shellcode will be written in the victim's computer memory. This can be achieved in two ways:

1. Heap based buffer overflow[9] – targeted insertion of malicious code to computer memories heap memory space [18]
2. Stack based buffer overflow[10] – writing to adjacent memory locations of the memory allocated to the program itself [20]

After spraying the heap or overflowing the program memory buffer, the injected payload code will be executed. The goal of the shellcode is to divert the program flow to it so that the attacker has control over the targeted machine over a remote shell. This remote shell is used to inject further malware to the victim's computer to maintain persistence [7, 11, 15].

2.3 Honeypots

One group of cyber security tools out there are honeypots. Generally, they are used to lure attacks against them or are tasked to probe a specific target themselves, therefore some of them can be classified as passive protection systems and the others proactive protection systems. To observe concrete threat actors targeting one's organizations gives valuable insight of their goals and in summation using all that information can improve security. Additionally, different honeypots have distinct functional and operational differences. Classical honeypots, also referred as server side honeypots, are used to investigate server side attacks and client side honeypots are used to investigate the client side attacks [14, 16].

2.3.1 Server side honeypots

Server side honeypots can be divided into two main subgroups: Low-interaction and high-interaction. Low-interaction server side honeypots are fairly static and provide a limited amount of services with limited functionality. In the **Figure 3**, they emulate some of the functionalities of the emulated services such as SSH and Telnet by letting the attacker insert commands. These commands do not execute anything and sometimes provide a confusing error message. On the contrary, high interaction server side honeypots serve an entire operating system, with its own legitimate software services and any other third party service to make the system look as real as possible. Usually, the main goal here is to let the attacker get root or system level privileges on the operating system. In return a great deal of evidence is gathered and possible new exploits or methods how an attacker managed to elevate his privileges.

[9] Heap Spraying Buffer Overflow. https://tools.cisco.com/security/center/viewAlert.x?alertId=21136
[10] Buffer overflow. https://www.owasp.org/index.php/Buffer_Overflow

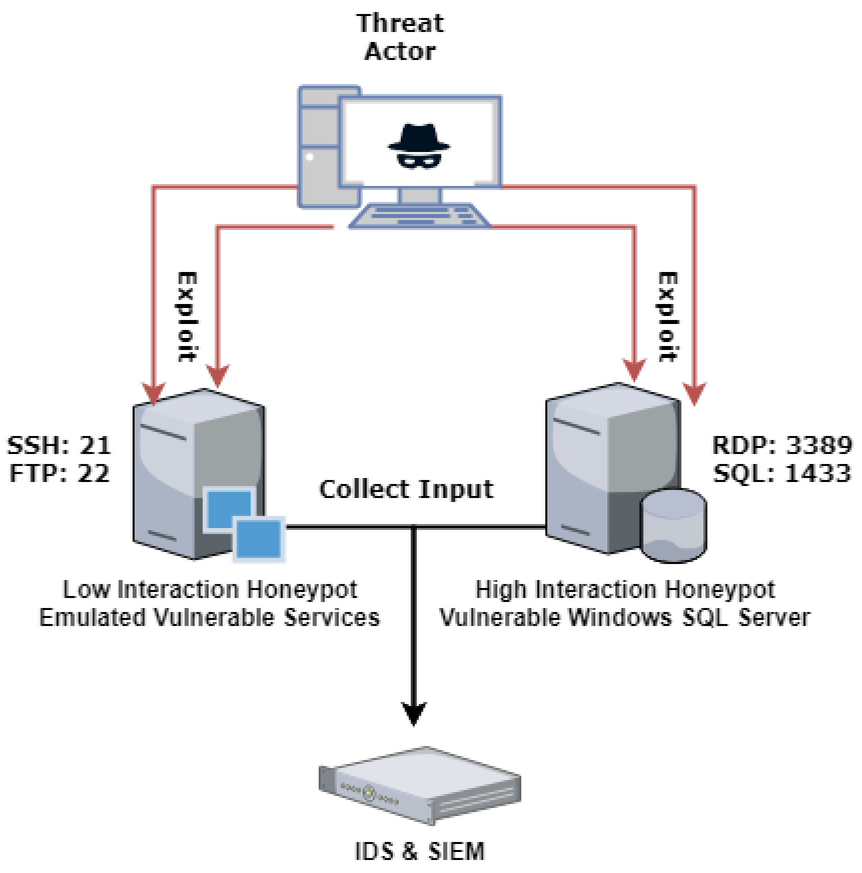


Figure 3. Honeypots enriching passive defense cyber security systems

The collected input is valuable CTI and it can be used to harden the real system, supplement any static defense tool with signatures and to monitor for events (**Figure 3 - IDS**). Commonly to all the server side honeypots, all the malicious activity is being logged and in most cases gives valuable insight of the attackers purposes and methods [14, 15, 16].

2.3.2 Client side honeypots

Client side honeypots are called honeyclients. Similar to server side honeypots, honeyclients can also be clearly distinguished by their interaction level. High-interaction honeyclients operate on a real operating system by using a real browser to navigate through websites. This browser is also called a headful[11] browser and it is a fully-fledged browser driven over a API or a driver. Furthermore, the detection is performed by state changes of the Operation system. For example, it monitors whether a file was created or downloaded, a registry entry was added or modified or a new service was created. Because of running real systems, they also are resource demanding. Low-interaction honeyclients on the other hand emulate the browser and mimic user behavior by interacting directly with the website on a programming level. The common tool for all honeyclients used to interact with the website is called a headless browser. In short, this type of browser has full features like a common user browser but it does not have a graphical user interface and is completely driven by program code.

[11] Headful browser definition. http://www.open.ou.nl/hjo/supervision/2018-g.vlot-msc-thesis.pdf

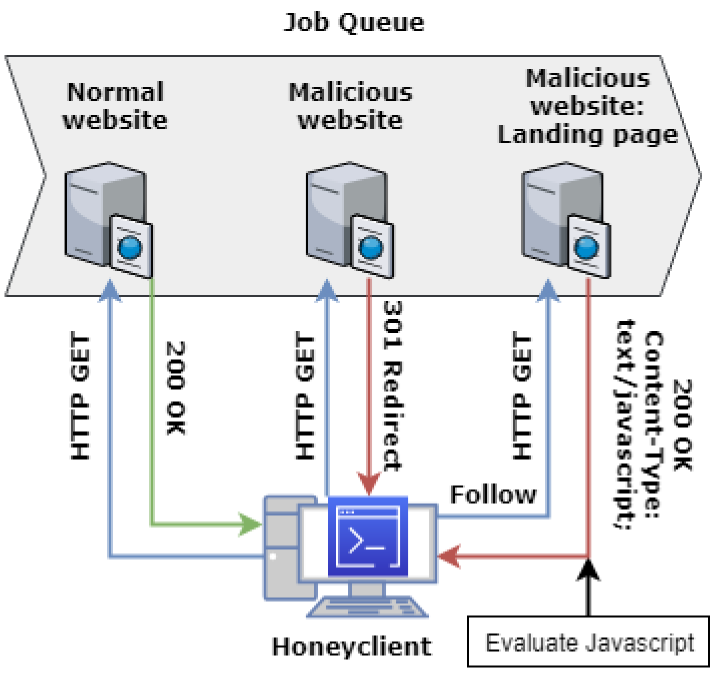


Figure 4. General logic of a honeyclient crawling and evaluating websites.

The most common approach used in low interaction honeyclients is to fetch the content of the website and match it against security patterns which match for known malware or exploit code. A signature is essentially a specific set of strings in a specific order and sometimes the string is expressed by some logical statements. In the **Figure 4**, the scanning queue consists of two websites. The normal website (**Figure 4**) is validated and the next website is malicious, which sends a HTTP 301 redirect. The new page redirected to is added to the scanning queue but the entire process from the malicious website (**Figure 4**) to the landing page is evaluated together. Therefore, honeyclients work fast and require few system resources. However, there is a downside to the signature based approach which leads to failure in detecting unknown, heavily obfuscated and dynamic threats. To tackle the problem with dynamic Javascript components, low interaction honeyclients use compiled javascript engines to emulate the browser attributes on top of it. [14, 15, 16].

Honeyclients are less resource demanding compared to server side honeypots. Server side honeypots also require a lot of time for an attacker to get interested. On the other hand, client side honeypots are more fruitful since they literally crawl the malicious website and find indicators of compromise and are part of proactive defense. The targets of interest for them are usually websites or web applications. Depending of the architecture and design of the honeyclient, the main goal is to identify malicious content from a hosted website, extract IOC's or determine whether the website is malicious. The input for crawling is retrieved from individuals who stumbled on suspicious websites or received deceptive emails with URL addresses. Therefore, the role of client side honeypot is to visit the website to trigger and identify the potentially malicious code itself [14, 15, 16].

2.4 Low interaction honeyclients

There several proof of concepts, honeyclients and methods for detecting EK's and malware from websites. Therefore, the related work focuses specifically on honeyclients or methods in the same category, which use the partly the same overlapping technology, tools and approaches.

2.4.1 Thug

Thug is one of the most popular honeyclients currently in active development and is mostly used in science projects as a reliable tool for extracting malicious content from websites. Thug has extra abilities and features which make it a hybrid client side honeypot, being between low and high interaction. The first major difference is that it should be ran in a virtual environment, since it tries to emulate OS environments, where code and files can be executed. This is due to the risk that the exploit code being run can take over the system running the honeyclient itself. Thug does not just de-obfuscate javascript but tries to execute it in V8 javascript engine. This is required to extract the shellcode from the heavily packed and obfuscated javascript code. To achieve that, Thug uses a library called Libemu[12], which is an x86 emulator and is used to detect shellcode. Libemu monitors the system calls to the CPU and based on the behavior detects the shellcode. There are also limitations, for example the emulator does not support the x64 environment and fails to detect anything other than shellcode. Nevertheless, it supports the emulation of all popular plugins and their versions, which will help to trigger attacks specific to the corresponding version. Thug is able to manipulate with HTML DOM events. Most infected sites do not start to infect the victim immediately. The user for example has to do some clicks or move a mouse over some object to trigger the attack sequence. Fortunately, Thug is able to manipulate with DOM events, which is a useful feature. Thug uses YARA rules to classify URL addresses and dynamic code. Essentially, general known keywords are matched with YARA rules and also typical URL structures of EK's are inspected. The main feature which makes Thug the best low interaction honeyclient is the programmatically fully emulated javascript browser environment, with most of the main variables, its properties and the core functions are implemented. Lastly, Thug utilizes the abstract syntax tree and sets breaking points automatically to *eval()* calls longer than 64 characters. These breakpoints are identified through static analysis. Eventually, starting from the breakpoint the javascript code will be executed in a browser like emulated environment on top of V8 javascript engine. Thug is a perfect tool for crawling the malicious website, triggering any attacks vulnerable to the emulated browser profiles, fetching the malicious payloads and inspecting these with generic YARA rules. Furthermore, it literally automates every step of the security analyst procedures to help to detect any malicious actions. Hence, it is a semi-automatic tool and the real value can be achieved by manually setting the parameters for triggering the specific attacks [22, 25].

2.4.2 YALIH

The acronym for "Yet Another Low Interacion Honeyclient" is YALIH. As the name says it is a low-interaction honeyclient. Its headless browser has three input variants. By either directly inserting the URL to be visited and analysed or it can crawl websites taking a specific string as an input. The first step during the analysis process is to check the URL against online databases, to determine if the URL has already been classified as malicious. The headless browser is pre-configured to emulate the headers of the HTTP request and of course

[12] https://www.honeynet.org/project/libemu

itself is capable of performing the request to the target server. Furthermore, the most important features are session management, cookies handling, redirection capabilities and automatic referrer management. Websites with malicious goals have some checks in place to identify whether the visitor is not a bot or a honeyclient. This evasive activity by the attacker is referred as cloaking by the authors of YALIH. The cloaking procedure most commonly consists of several steps. Firstly, some redirections will take place. Then after that some cookies might be set for that current domain and some more redirections are performed, but in this case redirections are done usually within the same subdomain, since cookies can be set for a specific domain and all of its subdomains at the same time. During all of the previously explained redirects, the HTTP referrer field is also checked. HTTP referrer field is set every time during a redirect and the value of it will be the last webpage what the browser visited. These are the most common cloaking techniques used and YALIH is capable of emulating the browser correctly, automatically setting the expected values and sending a correctly crafted request to the server. After the honeyclient passes the checks set up an attacker, it will be redirected to the real attack. If it contains javascript or the attack is done via javascript, Rhino java engine is used to de-obfuscate the script. Obfuscation is used to bypass signature based defenses, therefore it is important to de-obfuscate everything. Finally, after all the previous procedures, the files are being saved and ran against several antivirus engines. In many cases the Antivirus engines yield no matches and the analyst needs to investigate the downloaded files manually. YALIH is a simple, fast and powerful tool [14].

2.4.3 Honey-C

HoneyC was one of the first low interaction honeyclients and it is written in Ruby programming language. The solution consists of three main components: visitor, queuer and analysis engine. The Queue module is responsible for maintaining the list of addresses to be visited and the input is gathered via search engine API's. Queue module also contains the configuration for the search patterns. Thus, the goal is to crawl websites based on the keywords and seek out potential malicious content. This task is done by the visitor module which essentially is a simple HTTP module of the Ruby library. The downside of this is the failure to follow complex redirects, such as javascript provides and there is no mimicking of a real web browser. Furthermore, the HTTP referrer field is not set. The goal of the visitor module is only to retrieve the data and send it through the analysis module. This module is responsible for matching the page content and HTTP header values against Snort[13] signatures. There is no IDS on the wire and the detection is achieved through a Ruby Snort wrapper, which has been developed by the authors themselves. Lastly, only HTTP signatures are used to avoid using excessive amount of signatures. The project is discontinued but the solution itself is solid and is able to quickly process and successfully detect known threats from websites [23].

2.5 High interaction honeyclients

Three high interaction honeyclients are described in detail which can be distinguished by different detection and analysis approaches.

[13] Snort IDS. https://www.snort.org/

2.5.1 Cuckoo

Cuckoo Sandbox[14] is the most used and popular open source driven malware analysis system currently available. The project is designed to automate the malware analysis process and it is written purely in python. Cuckoo utilizes virtual box as the hypervisor on the underlying OS. The general concept is to virtualize the entire OS with all the programs installed and simply execute the malicious payload. After execution the OS will be rerolled to the previous snapshot. Cuckoo supports three input types URL, file or a hash of a file and these inputs are accessible over the API and graphical UI. A file may be uploaded to several virtual machines concurrently, which are running different OS and software versions to maximize the detection ratio. Every virtual machine can have different Ethernet connection types: normal internet connection also called a “dirty-line”, no internet connectivity at all or an emulated network. Furthermore, the malware execution process is monitored thoroughly and the digital evidences, such as log files, can be retrieved later for deeper analysis. The key information that is retrievable after the execution of the sample are listed below:

- Memory dump of the machine
- PCAP of the connections
- CPU API calls from malicious processes
- List of files created, modified or deleted

Since the entire environment is designed for dynamic malware analysis an optional connection to a static analysis solution is configurable, such as IRMA[15] and Virustotal[16]. Most likely threat actors are using Virustotal also to see if their OPSEC has been compromised by querying hashes of their malicious payloads, therefore offline solutions like IRMA are favoured. Furthermore, Cuckoo supports a variety of file types and is capable of extracting nested achieve files. In the focus of this work the most important feature of it is the capability of analysing URL addresses. The URL is forwarded with a parameter and after the guest OS has booted a browser will be tasked to navigate to it. The default browser and its plugins must be pre-configured. To help to trigger the malicious events, Cuckoo will be emulating clicks in the browser window. The user has also the capability of manual control to interact with the guest OS environment. Lastly, Cuckoo is a very powerful tool with high detection rates due to using real computer software environments. Due to the virtualisation capabilities it is also very resource demanding to run multiple virtual machines concurrently and the automatic analysis process takes time [24].

2.5.2 CaptureHPC

CaptureHPC is a high interaction honeyclient framework which utilizes the VMware virtualization platform. The central server is orchestrating the virtual machines over the vmware API and it is also responsible for sending the inspection commands to the guest OS clients. Furhtermore, the configuration template is extendable and has a rich amount of properties which can be set before launching the virtual machines and executing the commands. For example, one can set the time limit how long the guest OS should operate and which URL to inspect. Hence, one or several URL addresses can be visited with the same client and with different web browsers. Also the same principles apply as for Cuckoo, the guest OS need to be pre-configured and after tasks have been completed the OS will be reverted to the previous snapshot. Furthermore, every guest OS needs to be preconfigured with the Captur client, which essentially is the client which operates, executes commands and monitors the guest

[14] https://github.com/cuckoosandbox/cuckoo
[15] https://irma.readthedocs.io/en/latest/
[16] https://www.virustotal.com

OS. The information is exchanged in XML between the orchestrator server and the software level client. The following VM states are monitored:

- Network activity
- Kernel and registry
- File creating and deletion
- Processes in memory

Unlike Cuckoo sandbox, there are no detection mechanisms and the framework gives only feedback of the OS changes. Then after the analyst must determine which event was malicious. Furthermore, there is no way for the analyst to interact manually with the guest OS. Lastly, any malware can avoid detection just by checking the process list for the CaptureClient [26].

2.5.3 Shelia

Shelia is a honeyclient focused on detecting attacks against Windows operating systems with the goal of mimicking a naive user. The input for the detecting process is retrieved via an email client connector, which extracts the suspicious URL addresses. The first component to receive the URL addresses is called the queuer. Next, the list will be sent to the client which will visit the URL addresses with the purpose of triggering client side attacks. Eventually, an analysis engine is monitoring the virtual machine to determine any malicious activity. The main component is the analysis engine. This engine will be monitoring file system changes, registry changes and network activity via WAPI hooking. It is a machine learning feature and it learns by establishing a baseline of normal activities in the guest OS. Furthermore, a specifically altered DLL is running in the memory with the goal of facilitating a process an attacker can hook or inject to. This DLL is then intercepting the API calls of the malicious process and these calls are presented to the auto learning feature. With the long term learning process this honeyclient produces almost no false positives [38].

2.6 Conclusion

This paragraph described the general background of website threats and presented the tools and their features, which help to detect these threats. The honeyclients reviewed present different angles and approached how to identify and detect malicious content from websites and these tools can be easily distinguished by their interaction levels. The paragraph answers the research question: How does proactive defense, particularly using client side honeypots, help to detect malicious characteristics from websites?

3 A methodology for revalidating cyber threat intelligence

This chapter is answering the second research question: How to validate cyber threat intelligence by adopting the honeyclient principles with the goal of keeping the web based indicators of compromise relevant? The author proposes an alternative methodology combining already known techniques and solutions with the aim of validating CTI entries to keep them relevant. The focus of the methodology is to enrich and re-validate web related threat indicators in a CTI DB without removing them.

3.1 Motivation - complement technical features of existing solutions

Every honeyclient is built to automate a specific task with a specific scope and certain technical restrictions. Therefore, every honeyclient is functioning well with the tasks it has been designed to solve. This thesis is oriented to develop a methodology aiming to enrich CTI databases by validating their entries. The solution does not have to detect or classify specific payloads nor does it need to retrieve any malicious component. On the contrary, the verdict has to validate and confirm whether the CTI entry is still valid. Every CTI entry has metadata which can be used to apply certain playbook of rules to confirm the entries validity. If one knows what to expect the certain set of signatures with scores can be used. If the CTI entry has few metadata a generic playbook can be used. To achieve the maximum detection using generic signatures javascript needs to be executed fully. One can just not relay on emulated javascript environments and this is the fundamental principle of this methodology. Secondly, in the scope of this methodology it is not needed to perform the full analysis cycle, like Thug does. Therefore, the current honeyclient solutions are not fully applicable for this task. The principles of browser profile emulation are adopted from current honeyclient solutions with slight improvements, discussed in paragraph *3.3.1*. The byproduct of this methodology is a **prototype** honeyclient tool, which consists of open source components. There are many open source components which achieve the same goal and the current pieces are chosen subjectively by the author.

Table 1. Comparison of honeyclient solutions.

	Feature	Thug	Yalih	Honey-C	This Thesis
1	Emulate Javascript browser environment	Yes	No	No	Yes
2	Execute Javascript in a sandbox	No	No	No	Yes
3	De-obfuscate Javascript	No	Yes	No	No
4	De-minify Javascript	No	Yes	No	No
5	Browser objects for Javascript	Partially	No	No	Yes
6	Shellcode emulation	Yes	No	No	No
7	HTML DOM Manipulation	Yes	No	No	No
8	Handle alternative protocols: data, file, ssh2	No	No	No	Yes
9	Allow proxy configuration	Yes	Yes	Yes	Yes
10	Costum User-Agents	Yes	Yes	Yes	Yes
11	Autofollow redirects from HTTP headers	Yes	Yes	Yes	Yes
12	Autofollow redirects from meta tags	Yes	No	No	Yes
13	Autofollow redirects from Javascript	Partially	No	No	Yes
14	Autofollow redirects from iframes	Yes	No	No	Yes
15	Autofollow pop-ups	Yes	No	No	Yes
16	Autofollow external scripts	Yes	No	No	Yes

17	Set HTTP Referrer field automatically	Yes	No	No	Yes
18	Set and use Cookies	Yes	Yes	No	Yes
19	Charecteristic based detection	No	No	Yes	Yes
20	Scoring algorithm	No	No	No	Yes
21	Signature based detection	Yes	Yes	Yes	Yes
22	Threat classification using signatures	Yes	Yes	No	Yes
23	Optional connection to Cuckoo and IRMA	Yes	No	No	Yes
24	Browser plugins	Yes	No	No	Yes
25	Open Source AV engines required	No	Yes	No	No
26	Ceritifcation verification	Yes	No	No	Yes
27	Fetch and analyze payload (Executable)	Yes	No	No	No

Explanation of rows from **table 1**:

1: Low interaction honeyclients mostly use a javascript runtime engine which are also used by browsers themselves. Additionally, some environmental variables are programmatically defined alongside with some of their properties, so that if a script calls a browser object and its property, it will not fail to execute. The real functions are not implemented and if invoked the execution process will fail.

2: Extract javascript from the response and evaluate it in a javascript sandbox. All the browser properties and variables must be pre-defined. Every script element will be evaluated.

3: De-obfuscate packed javascript.

4: Expand javascript source code to human readable format. The porpuse of this process is to generate an improved layout to maximize the detection of static signatures.

5: Emulated browser variables on top of a javascrip execution engine.

6: Emulate windows environment and intercept API calls to the CPU.

7: Perform interaction with the website by clicking mouse or keyboard buttons.

8: Detect and handle alternative data protocols.

9: Support proxy support for geolocation to detect watering hole attacks.

10: Browser profile with fully mimicked properties.

11 – 15: Follow and detect all possible redirect variations.

16 - 17: Set referrer and use cookies automatically according to RFC.

18: Generic detection rules at any stage.

19: Verdict is based on the weight of triggered signatures.

20: Static detection rules at any stage.

21: Group threats by any common basis using signatures. Use the classification to trigger certain rules according to the task or whitelist certain events during certain tasks.

22: Fetched files can be automatically pushed to third party solutions for additional analysis.

23: Emulating the presence of browser plugins.

24: Detection is performed merely by open source antivirus solutions.

25: Verify SSL certificates.

26: Emulate OS environment or intercept API calls to detect potential shellcode.

3.1.1 Choosing a known approach for the honeyclient

The scope of this paper is restricting the honeyclient technology to low interaction honeyclients, which perform faster and are more efficient with general signatures. The goal of the comparison is to choose the best of the three most commonly used techniques among low interaction honeyclients. Honey-C implements an IDS approach, YALIH relies on auto learning and de-obfuscation principles, Thug relies on classification of threats and emulates DOM and javascript browser environment. In the previously referenced studies **2.4.1, 2.4.2, 2.4.3** the honeyclients are measured against a clean data set which contains known good URL addresses to detect false positives and against a bad known data set. In several cases vulnerable and old versions of browser are used, such as Internet explorer 8 personality pretending to be ran from Windows XP SP3. The best false positive and false negative ratio is achieved by Thug, then after comes YALIH and finally Honey-C. Honey-C uses Snort IDS rules and they are mostly focused on server side attacks not client side attacks. Furthermore, most of the IDS signatures are generic which makes them false positive prone and all class type rules were used. Since YALIH uses CLAM antivirus, which also has many generic rules, the detection ratio is low but also the false positive rate is low. YALIH YARA signatures are generated based on malicious content with auto learning techniques. Therefore, after every iteration the false positive rate will increase, because it is learning based on the false positive results. Thug does not use generic signatures for detecting malicious content from websites and instead is aiming to identify breakpoints for the dynamic analysys by utilizing the javascript abstract syntax tree. Thug is the only honeyclient amongst the described three which utilizes a javascript engine to execute the code where breakpoints are set. After the analysis process, Thug does not give a verdict about the maliciousness of the website, but has a rich event log. The event log also accurate due to classification rules, which exclude known good patterns and include generic known bad patterns, therefore it is simple to parse the log and aggregate events to determine the maliciousness of the task. Conclusively, if YALIH is provided with more precise heuristic signatures and Thug is supplemented with improved classification rules, their false positive ration is fairly low. On the contrary, the IDS signature oriented Honey-C is no match against pure honeyclients.

Furthermore, based on multiple researches [7, 11, 13, 14, 15, 16, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37] the measurement of false negatives in malicious datasets implies that any kind of generic signatures in honeyclients are able to detect an acceptable amount of malicious activity and are also due to that are prone to false positives. The false positive and false negative amount can be decreased through auto learning processes and the detection ratio can be increased through classification of threats and applying the corresponding detection rules. The referenced studies show, that a proactive crawler is needed with javascript enumeration or classification, with emulated browser features and low interaction honeyclient principles for fast data processing.

Based on the studies in subjects **2.4.1, 2.4.2, 2.4.3** and in the previous immediate text section above referring to multiple researches, the following principles are adapted to this methodology:

- Execute Javascript in an enumerate environment
- Adopt browser profiles and their plugin data
- Follow all types of redirects
- Automatic cookies and HTTP referer management

- Web crawling logic
- Low interaction principles

The methodology aims to improve the listed principles. First, javascript engine is replaced with a sandboxed pure javascript environment, which improves security and does not relay on setting breaking points dependent on static analysis. Secondly, the methodology aims to extract static content such as request and response headers, response bodies, all nested frames and external content by intercepting every HTTP interaction with the purpose of scanning the extracted content against malicious indicators. Next, this content is evaluated with signatures with impact scores. Lastly, the most important principle to adapt is the low interaction feature, which allows a fast analysis process with low computer resources. The low interaction feature can be rephrased as a fully programmed headless browser described in subject **2.3.2**.

3.2 Fundamentals of enriching cyber threat intelligence

"*There is nothing more necessary than good Intelligence to frustrate a designing enemy.*" - George Washington[17].

In the past organizations were facing mass attacks, not trends have shifted to more complex attacks originating from APT groups. To face this new landscape collecting and exchanging CTI was introduced with the goal of detecting more sophisticated threat actors. CTI is essentially knowing the intentions, methods and motivations of the threat actors. APT groups are constantly improving and changing their TTP's to ensure high attack success rates. Therefore, to tackle this dynamic attributes of the threat actors, the CTI must be useful and in the correct format. **Figure 5** explains the minimal set of required information for an CTI entry to be applicable for this methadology. Collecting trivial data is a low priority and all IOC's need to have context. For example, the MISP[18] database contains events. Each entry event in the CTI database has attributes which are IOC's themselves but with context added to them. Eventually, context is metadata for the IOC itself. Metadata defines the type of the IOC, for example IP address, URL, file hash, email address and adding extra metadata or context is possible.

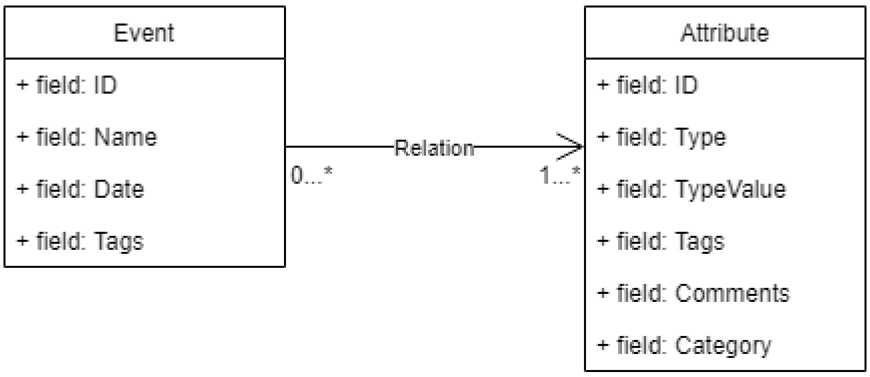


Figure 5. Simplified relation between CTI event and its attributes

[17] Quote copied and cited from: https://www.cia.gov/library/center-for-the-study-of-intelligence/csi-publications/books-and-monographs/the-founding-fathers-of-american-intelligence/art-1.html
[18] https://www.misp-project.org/

Collecting CTI from attack attempts against one's organizations is a must, especially investigating samples which have penetrated the first line of defense. For example, automatic sandboxes can extract the IOC's from malicious email attachments. Nevertheless, it is not possible to defend every asset equally good. Therefore, assets need to be prioritized and IOC's need context which defines their severity. Also, avoiding false positive entries to the CTI can prevent ghost hunting for the incident response team who use this CTI to investigate possible attack patterns. To sum it up, actionable intelligence must be timely precise and relevant. To achieve this, IOC's must be periodically updated with severity levels or impact scores. Furthermore, the higher influence IOC's must be prioritized for validation. Most often the high influence IOC's are determined through prioritizing critical business assets [21].

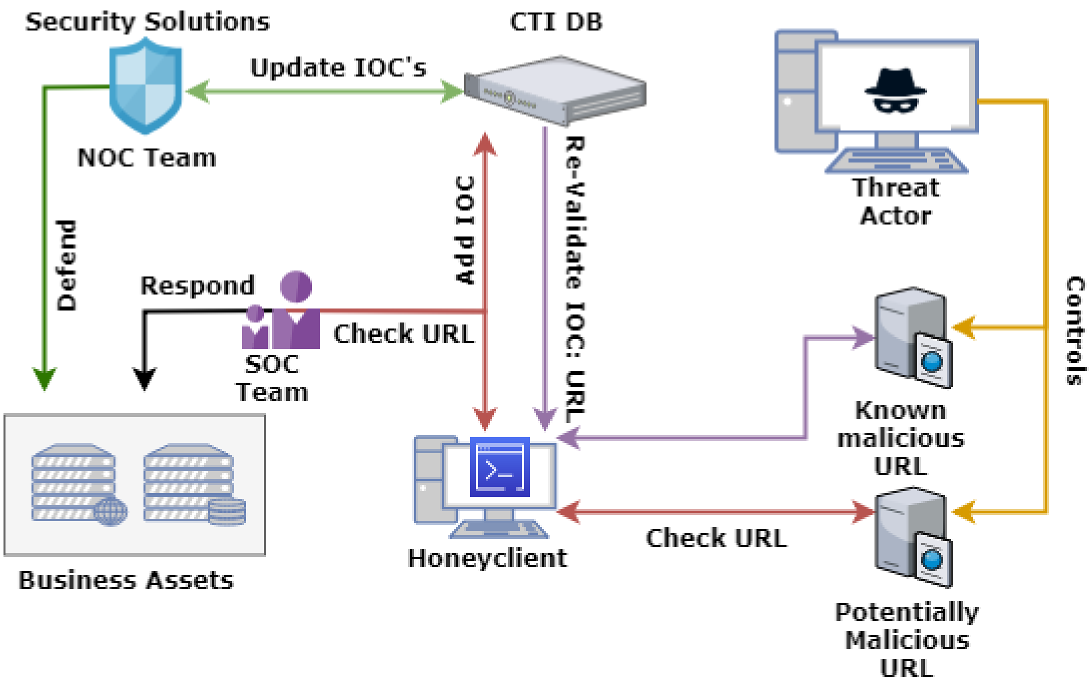


Figure 6. Enriching and validating CTI.

Validating URL addresses in the CTI DB (**Figure 6**), if ensured that this process **will not violate the operations security**, can be achieved by using honeyclients. The validation process in **Figure 6**:

- Honeyclient acquires Task list from CTI DB
  - Honeyclient crawls and validates known malicious URL with <u>known context</u> what to expect (*iframe, EK, malicious download*)
  - Honeyclient tags the CTI entry with validation tags: Last Updated, Likely malicious
- Honeyclient acquires Task list from SOC TEAM.
  - Honeyclient crawls and validates unknown potentially malicious URL only presuming the threat content what to expect (IOC's acquired from friendly companies with lesser context) and outputs information if any generic malicious content is detected.
  - SOC team investigates results and responds accordingly to supplement defenses for critical business assets.
- NOC team regularly updates IOC's from blacklists.

- If IOC has been checked ten times by the automation process the blacklist entry is removed
- High value IOC's will be sent to the SOC team to re-assessment (not trusting automation)

Also, the same honeyclients can be used to enrich the CTI DB by crawling suspicious URLs sent over emails and they can also be tasked to verify URLs extracted from automatic or manual malware analysis.

The validation process is a regular reoccurring process. Depending on the aim and the scope of the CTI DB, the timeframe is individual. The timeframe can be set by evaluating the false positive incident response occurrences. If there is such an event every week, then the CTI DB should be validated every week. Next, IOC's must be extracted over the CTI DB API in a common data format. The fully automated honeyclient solution validates the entries and appends the verdict to the same file, which can be re-imported to the CTI DB. Lastly, the purpose is not to delete the precious threat entry but to tag it according to the result of the validation process, for example: low risk, cleaned and the validation timestamp.

3.3 Proposed detection principles

This subject elaborates the improved or alternative principles described in the end of subject **3.1.1**. The following conceptual principles of the honeyclient are stated as following:

3.3.1 Overcome evasive challenges

There are several limitations to take into account when using honeyclients. It is a challenge for honeyclients to detect targeted and watering hole attacks. These challenges can be mitigated by using proxy servers in the targeted area, but it is still a guessing game whether there are specific IP address space limitations, OS limitations or which browser profile attributes are being checked and expected. Hence, leaving the edge cases by the side, it is still a challenge to build browser profiles with correct properties to maximize the trigger rate of malicious activities. Therefore, a headless browser has to act as close as possible to the real browser, mimicking all the properties and actions a normal browser would perform.

Firstly, starting with the HTTP headers, the main goal is to understand HTTP codes and act accordingly to the IETF[19] RFC's. Drive-by-downloads and EK's are fond of redirections, so all codes from 301-303 need to be correctly handled. Following the redirect is not enough, setting the referrer accordingly after each redirect is also a must. There are also HTML meta, frame and javascript redirects, which also need to be taken in account. Furthermore, cookies and session handling is required since most EK's set cookies on the landing pages and check their values in the following steps. Lastly, the HTTP header User-Agent has to be set correspondingly to emulate a request from a vulnerable browser [14].

Secondly, for the honeyclient to handle the javascript is a challenge by itself, which requires a pre-compiled javascript engine with all the browser properties and attributes present. The honeyclient must be able to emulate the global BOM[20] objects, such as: Window, Navigator, Document, Screen and most of their properties. The HTML DOM is a property of the BOM Window object. These HTML DOM objects are accessible over the browsers DOM API and manipulated with javascript. Most likely advanced adversaries probe for these objects and their properties in the memory and use their functionality to perform further tasks. Alongside the BOM properties, also the threat actors emulate browser characteristics such

[19] https://tools.ietf.org
[20] https://www.w3schools.com/js/js_window.asp

as plugins. This is due to the fact, that many of the vulnerabilities reside in the plugins or their API calls which helps the threat actor to choose the corresponding exploit code. Furthermore, a threat actor may probe for window and navigator sub-attributes such as "*webdriver*", "*htmlunit_webdriver*", "*phantom*" or "*selenium*", which are properties set by popular headless browsers, with the goal of preventing revealing the next steps of malicious content to the automated tool [19].

Table 2. Browser object properties usable for headless browser detection [17, 19].

Browser property	Reason for fingerprinting and emulating
Navigator.user-agent	Determine the browser and its version
Screen.width & Screen.height	Dynamic in real and static in headless browsers
Window.width & Window.height	Window dimension size – value is 0 in headless browsers
Navigator.plugins	Determine the plugins and their versions
Navigator.platform	If OS platform does not match with the User-Agent OS value
Navigator.online	Set if browser has internet connection, headless browsers usually have it unset
Window.webgl	Headless browsers have no graphics renderer
Navigator.mimetypes	Determine flash, java and pdf support
Navigator.*Language	Is not defined in headless browsers
Navigator.platform	Common mismatch with the User-Agent value

Table 3 explanation:

- Navigator.user-agent and Navigator.platform are most likely compared: Does the User Agent match with the real platform value?
- Navigator.online states that the browser has internet connection – this has to be enabled explicitly for web crawlers. A browser cannot browse the web if its offline.
- Screen.width & Screen.height are the full dimensions of the monitors resolution. Window.width & Window.height are the full dimensions of the currently open window. These cannot match and they all need to be populated. A browser without a port size is a bot.
- Window.webgl is often checked to identify Mozilla Firefox and Chrome browsers
- Navigator.plugins and Navigator.mimetypes need to be matched and populated if trying to trigger any traps with honeyclients. Every plugin has supported mimetypes. If a browsers plugin array is empty but the mimetypes are there, or if the browser plugin array is populated and the mimetype array is empty - this all refers to a bot.

Conclusively, one can say, that the BOM objects, their properties and attributes alongside with browser characteristics such as plugins, fonts and the javascript field "*user-agent*" itself, altogether make fingerprint of the browser.

3.3.2 Dynamic Content

Complex multilayered javascript obfuscation has been a challenge for low interaction honeyclients. Thus, this means they do not execute the javascript, instead it is deminified and beautified. In some cases, the code is executed in a pure javascript engine, without all of the browser properties. Furthermore, if the code is packed with a known javascript packer, that code may be unpacked. Next, the output is matched against static signatures or certain keywords, which is limiting the detection capabilities, especially of zero day attacks [13, 14].

Furthermore, automatic learning techniques are used as an alternative approach to enhance the chances of unknown threat detection. To achieve that every threat type needs to be categorized and certain characteristics need to be omitted to these categories. Then after, there is a deep learning curve and different learning data sets need to be used to generate a sturdy baseline. The downside of this learning technique is that if the threat actor changes the attack technique then the automatic learning baseline, which was generated based on the old techniques, fails to detect new approaches [18, 27, 28, 30, 31, 34]. Therefore, the proposed solution is to use a Javascript sandbox, which introduces the commonly used BOM objects and emulates OS folder paths.

3.3.3 Categorized and content oriented signatures

The content the honeyclient receives from the server is matched against signatures. The signatures are divided into groups by their detection nature. A signature can be applied for every group, but has slightly different matching patterns and impact scores, for example a detection match of an executable download attempt from post dynamic content has more influence than from static content. This is because if commonly abused normal functions in cyber-attacks emerge after the sandboxing process, it implies that these functions were hidden or obfuscated thus having a greater influence. The goal of grouping is to increase the effectiveness of a signature in the following three scenarios (**Table 4**).

Table 3. Signatures grouped by content

	Detect From / **Group**	**Http Responses**	**Http Requests**	**Http response body**	**Outcome of dynamic code after sandboxing**
1	Static Content	X	X	X	
2	URL’s	X		X	X
3	Dynamic Content				X

Group static content: Contains signatures for detecting static malicious elements from HTTP requests, Response headers and bodies such as possible obfuscation attempts with various encoding: ROT13, base64, hexadecimal, unicode. Furthermore, it contains signatures which aims to detect meta refresh from HTTP response bodies and HTTP 30x redirections from HTTP responses and also their combinations or sequences. Suspicious javascript calls such as *eval()”,* “*unescape().*

Group URL’s: Aims to detect suspicious URL’s created by DGA, known bad TLD’s, when URL is IP and known short URL services which are widely abused to evade static blacklists. Detection of alternative data protocols: file://, data://, zlib://, ssh2://, rar:// and their wrappers. Suspicious static encoding in URL addresses. Suspicious URL addresses with none default ports. The URL signatures are applied to both static content and post sandbox action content.

Group Dynamic Content: Aims to detect only post javascript execution elements (after sandbox has executed javascript): ActiveX events, javascript redirects, Wscript shells, executable downloads, de-obfuscated URL addresses.

3.3.4 Scoring based evaluation and playbooks

Playbook creating is a manual process and is the most time consuming factor of this proposed methodology. Thus, the richer the training set with different malicious examples the better the outcomes. The focus is to create a minimal set of general signatures, which aim to detect common malicious elements from websites. Playbooks consist of grouped signatures (**see 3.3.3**) which aim to detect a certain types of cyber threat indicators, which are using a certain attack vector. Essentially, they are a set of signatures aimed to detect specific countable event occurrences in a specific order. The evaluation result is an impact score. By creating a minimal amount of signatures, which detect generic events help to reduce the playbook creation time for a user. Thus, this keeps the training time in reasonable interval and reduces the systems parameterization time and complexity. It is determined that creating and training a specific threat event related playbook with five signatures takes three human hours for an intermediate signature creator and a generic playbook with thirty signatures can take up to several days. As defined in **chapter 1.3**, the main constraint of this methodology is to have a well-maintained CTI database where the database entries have context. Without this predefined context, it is futile to re-validate the database entries with generic signatures, because they yield many false positives and these playbooks take too much time to create. Since the scope of the methodology is to revalidate the CTI DB entries, most of the metadata information can be used to create these signatures and verify if the web based threat exists. Thus, the goal is to detect already known elements of the IOC and this means the prerequisite is a well maintained CTI DB, where every IOC has context. Lastly, effective training of playbooks can be achieved from several network traffic captures with similar type of attack scenarios. The two main sources to acquire these PCAP traffic is from the companies own packet capture server or any third party partner or service. One of the way to use these traffic captures: The network stream must be converted to files format on a filesystem and the contents must be set up identically on a local test server according to the PCAP traffic sequence.

The signature groups consist of different signatures as described in paragraph **3.3.3** and each signature has a metadata part. The metadata has to define the impact score.

Table 4. Scoring ratio.

1-50	Insufficient information
51-75	Suspicious
76-100	Likely malicious

The scoring process itself is a simple addition of every rule triggered and during the analysis process the same rule can fire multiple times, for example when multiple redirects occur or the same rule is defined in different groups with different impact scores. The proposed scoring measurement range is described **table 5**. The score can also exceed 100, which just further confirms its maliciousness.

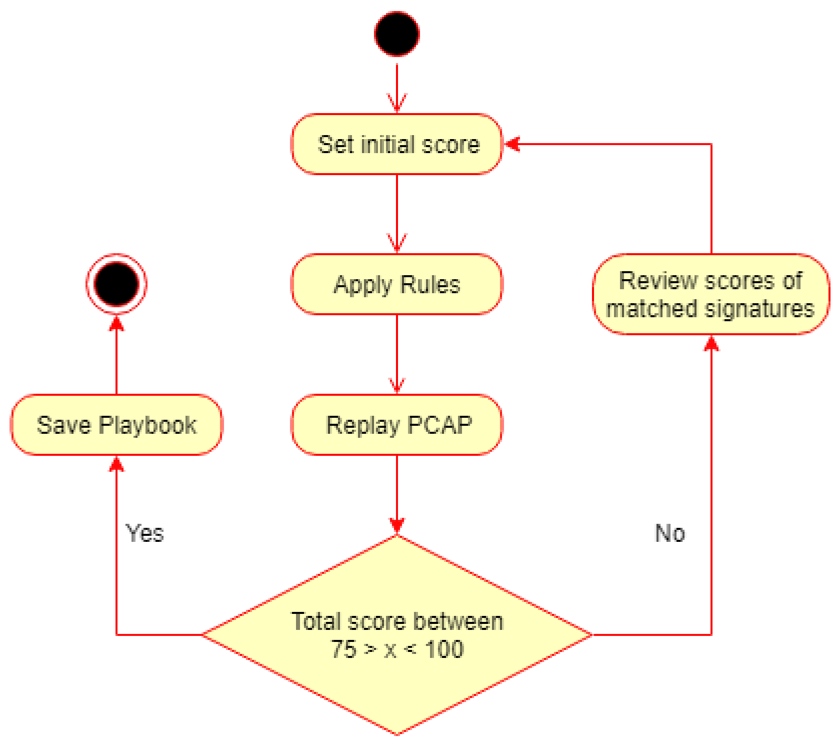

Set initial score
Apply Rules
Review scores of matched signatures
Save Playbook
Replay PCAP
Yes
No
Total score between 75 > x < 100

Figure 7. Score adjustments.

The diagram above (**Figure 7**) describes the process of setting the impact score of each signatures for each playbook. Firstly, the signatures are created by inspecting the malicious PCAP files alongside with studying the adversaries TTP's. The signatures need to stay as generic as possible and reflect the common techniques used by the potential adversary. The initial score is set higher if the element is unique or the specific sequence of elements combination is rare and the opposite applies to more common elements. Next, these signatures are matched against every HTTP stream extracted from the PCAP files and then the final score is calculated. It is to be noted, that one PCAP must contain a single malicious process, for example Angler EK process from the start to the end. This is required for creating detection oriented playbooks. As explained in subject **2.2.2** and **figure 2**, if a CTI database has entries that need to be re-validated and they have tagged context, such tags as: iframe, multiple redirect, http 301, short URL, executable download – a simple playbook can be created to check if these elements still exist on the known bad website related IOC. The playbook can be considered trained, if the average score stays between 51 and 99 on the training set.

Lastly, signatures can be tagged with several playbook ownerships but can belong only to one group. The group membership emphasizes the content it is intended to match and can be treated as a baseline. For example, a signature in the dynamic content group could lead to false positives against the same content found in static HTML content. The **main rule** of playbook ownership: if a signature is part of a playbook, it can be used by other playbooks in a read only manner. No change to the signature or score can be made. If there is a need to change, the signature must be extended and tagged appropriately. A playbook is a final entity after the learning process. Only new versions or variations can be made based on the already existing playbooks.

3.4 Conclusion

It is important to invest in risk management and determine the most vital and critical business assets with the goal of helping to secure the critical business assets by keeping the CTI validated and up to date. Thus, this subject answers the following RQ: How to validate cyber threat intelligence by adopting the honeyclient principles with the goal of keeping the web based indicators of compromise relevant? The answer to this question is a proposed methodology, with the aim to be implementable with a variety of open source components. The main components are anti-evasive techniques, fully executing javascript in a sandboxed environment excluding the need for de-obfuscation and deminification of the code and an alternative approach to manage detection signatures.

4 Implementing the methodology to a honeyclient

This subject answers the RQ3. How to implement the methodology using open source components? The result of this chapter is a designed architecture taking in account the detection principles described in chapter 3. The byproduct of the architecture is a fully automated prototype honeyclient.

4.1 Technical architecture

The proposed architecture is based on the methodology described in paragraph three, taking in account the main focus of the architecture, which is the fundamentals of enriching CTI, alongside with detection, scoring, anti-evasive principles and the low interaction principles for a fast analysis process with low computer resources. Next, the proposed architecture figure is used to explain the architecture in detail.

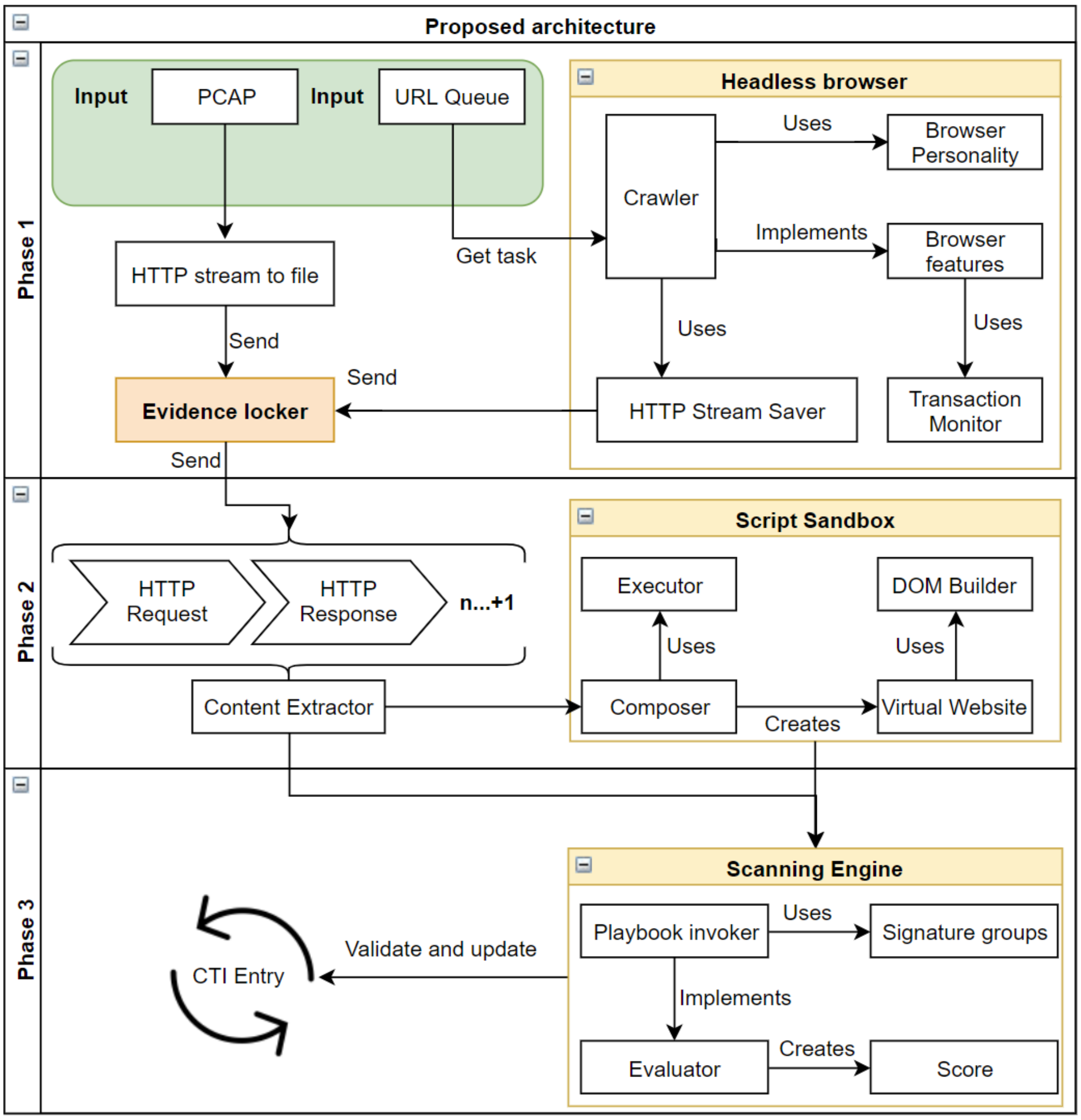


Figure 8. Prototype Honeyclient architecture.

The concept of the architecture in **Figure 8** is divided into three main phases.

Phase 1 is responsible for collecting and parsing the evidence data into a common format. The two input options are a PCAP file or crawling results of a headless browser. For the first input option, HTTP transactions are extracted from the PCAP network stream and saved in the correct sequence to the file system. The second input option gets a list of URL addresses to be crawled by the headless browser. Any personality can be used to mimic a certain browser with certain plugins. Furthermore, the Crawler component implements all the browser features such as handling cookies and auto-following redirects. The Crawler must be able to extract all the contents of any sub-frames presented in the main window frame. Next, an optional javascript engine with browser properties and variables is present. The script sandbox performs almost an identical task and this javascript engine can be used for layered coverage. If layered coverage is used, the entire prototype solution must be sandboxed. The most important part of the honeyclient is the transaction monitor, which is responsible for intercepting all the requests and responses between the client itself and the server with the goal of storing them on the disk as files. The end result of phase one for both inputs is the HTTP transaction history in the correct sequence, including headers and the content, which are stored in the evidence locker.

Phase 2 is responsible for preparing the evidence for the scanning engine based on two approaches. Firstly, the content extractor separates any static content and puts it to the scanning queue. Secondly, all the content is sent to the composer for dynamic analysis. The composer creates a virtual page based on all of the static DOM ignoring irrelevant elements. The virtual page DOM contents must be created inside the sandbox with javascript functions, since it is a pure script sandbox. **It is vital** to create the DOM contents correctly with the same corresponding tags, same ID strings and same Class ID strings, since the majority of the web related malware stores its components inside DOM elements and accesses the content over the HTML element ID string. Furthermore, this process must be repeated for every possible frame or nested frame found in the main window to get all the possible hidden DOM content required to initialize the malicious process. The executor is responsible for executing the dynamic content, keep an accurate analysis log file of the events and extract any URL addresses after the sandbox execution. The log file contains the main function calls and any dynamic change of the virtual page. Lastly, the log file itself is put to the scanning queue with the exported URL addresses.

Phase 3 is responsible for scanning the prepared data files and invoking the correct playbook for the most efficient detection and setting a summarized score based on the signature matches. A playbook contains certain signature sets with specially adapted scores corresponding to the task. The score adaption is performed manually through PCAP analysis described in paragraph **3.3.4**. Next, the evaluator applies the signature groups and matches them against the static and dynamic content correspondingly. Depending on the implementation, the evaluator can be tasked to automatically update the CTI entry over the CTI solutions API.

4.2 Implementing the architecture using open source components

The technical solution has to compose of a headless browser which will be tasked to mimic user behavior to detect malicious websites and it will enable to avoid anti-evasive headless browser techniques by behaving and functioning like a real web-browser. The second input option has to be already saved network traffic also known as PCAP. Furthermore, the study of the principles of exploit kits helps to distinguish the sequence of steps an exploit kit is using and makes it possible to design the main signatures for identifying the exploit kits key attributes. In addition, signatures have to be written to detect malicious indicators, such as hidden iframes, obfuscation, any kind of redirection in html, javascript and HTTP headers. The entire website analysis process will be monitored with minimal amount of generic signatures, which will have an impact score. The total sum of the score is used to determine the probability of the website containing the potentially malicious elements. Based on the proposed architecture the implementation is divided into three phases. The implementation platform will be Ubuntu OS 18.04, Java and Javascript programming languages are used to fully automate the tasks. The main contribution is a fully automated prototype honeyclient with multithreading features.

4.2.1 Phase one

This phase requires two input types, one for historical data and the other for live data. Firstly, PCAP files consist of network streams and need to be parsed for further analysis with static signatures. This tool needs to have certain features:

- Capability to extract http streams
- Deflate compressed GZIP streams
- Save results to file

TCPFlow[21] satisfies the previously stated requirements. After TCPFlow has extracted all the items from the PCAP file the unneeded files can be removed (**Snippet 1**).

```
tcpflow -a -r ../cap.pcap && rm !(*.html|*.[0-9][0-9][0-9][0-9][0-9])
```

Snippet 1. TcpFlow command for extracting HTTP stream from PCAP.

The html files are the responses bodies from the server and the numeric extension refers to the port number, which essentially are request headers and response headers. The PCAP file must not consist of any encrypted HTTP traffic – creating PCAP files with unencrypted content can be achieved through MITM proxy. The PCAP file contents must be manually and identically to the network stream re-crafted, so that the localhost webserver would respond with the same headers and content.

Secondly, the headless browser has to have a rich API for manipulating the interaction with websites. Furthermore, all the properties and functionalities of a real browser must be present. Creating a tool such as a headless browser is a project by itself and it is not needed to create it from scratch. In unit testing headful and headless browsers are often used and a tool called HTMLUnit[22] fits the requirements described in paragraph **4.1, Figure 8, Phase 1**. There are also alternatives headless browsers, which can be used, such as: *CasperJS, PhantomJS, Splash and zombieJS.*

Implementing the solution: First, the browser features and visitor profile need to be set up (**Snippet 2**).

```
webClient.getOptions().setJavaScriptEnabled(false);
webClient.getOptions().setActiveXNative(false);
webClient.getOptions().setAppletEnabled(true);
webClient.getOptions().setCssEnabled(true);
webClient.getOptions().setRedirectEnabled(true);
webClient.getOptions().setUseInsecureSSL(true);
//not needed, malware jail does dynamic part
//webClient.waitForBackgroundJavaScript(1000);
CookieManager cookieManager = new CookieManager();
```

[21] TCPflow. https://github.com/simsong/tcpflow
[22] HTMLunit. http://htmlunit.sourceforge.net/

```
cookieManager.setCookiesEnabled(true);
webClient.setCookieManager(cookieManager);
webClient.getOptions().setThrowExceptionOnFailingStatusCode(false);
webClient.getOptions().setThrowExceptionOnScriptError(false);
webClient.getOptions().setTimeout(5000);
```

Snippet 2. Set the webclient options and features.

The layered javascript coverage principle is not invoked since it was optional in this methodology, therefore dynamic script runtime is **disabled**, such as javascript or ActiveX. Insecure SSL is enabled, due to most of the malicious websites rely on generating fake certificates. Automatic redirect handling is enabled - this also automatically sets the HTTP referrer field correctly. Lastly, the session handling is enabled by adding a cookie handler and cookie databse.

The browser personalities can be adopted from Thug project[23] and configured to HTMLUnit. According to subject **3.3.1 table 3**: *Navigator.plugins* and *Navigator.mimetypes* need to match. Furthermore, table 3 states that *Navigator.online* must be set true. The window size and screen size is set automatically by htmlunit. These requirements are implemented in (**Snippet 3**).

```
private void initiate(){

    browserVersionBuilder = new BrowserVersion.BrowserVersionBuilder(BrowserVer-
sion.INTERNET_EXPLORER);
    browserVersionBuilder.setOnLine(true);
    webClient = new WebClient(browserVersionBuilder.build());
    Log.logger.info("Default IE Selected: " + webClient.getBrowserVer-
sion().getUserAgent());
}

private void pluginConfig(){

    webClient.getBrowserVersion().getPlugins().clear();
    PluginConfiguration pluginConfiguration = new PluginConfiguration(
        "Shockwave Flash", "Shockwave Flash 32.0 r0",
            "32.0.0.171", "Flash.ocx"
    );
    PluginConfiguration.MimeType mimeType_swf = new PluginConfiguration.Mime-
Type(
            "application/x-shockwave-flash", "Shockwave Flash",
            "swf"
    );
    PluginConfiguration.MimeType mimeType_spl = new PluginConfiguration.Mime-
Type(
            "application/futuresplash", "Shockwave Flash",
            "spl"
    );
    pluginConfiguration.getMimeTypes().add(mimeType_swf);
    pluginConfiguration.getMimeTypes().add(mimeType_spl);
    webClient.getBrowserVersion().getPlugins().add(pluginConfiguration);
}
```

Snippet 3. Adding plugin and mime type properties.

Next, the HTTP transaction is stored in the same format as TCPFlow does and save it to the evidence locker folder. For the honeyclient prototype the main function responsible for intercepting every HTTP data flow is the transaction monitor (**Snippet 4**). This satisfies the

[23] Browser personalities. https://github.com/buffer/thug/tree/master/thug/DOM/personalities

requirement defined in **4.1, Figure 8, Phase 1** which the end result is to save all the transaction data to the disk.

```
protected void setTransactionMonitor(String uid){

    Log.logger.info("[" + uid + "]" + "Injecting Transaction Monitor");

    webClient.setWebConnection(new WebConnectionWrapper(webClient) {
        @Override
        public WebResponse getResponse(final WebRequest request) throws IOExcep-
tion {
            WebResponse response = super.getResponse(request);

            staticContent.add(response.getResponseHeaders().toString() + "\n"
                    + response.getStatusCode() + " "+ response.getStatusMes-
sage() + "\n"
                    + "CurrentURL: " + request.getUrl() + "\n"
                    + "Referer: " + request.getAdditionalHeaders().toString() +
"\n"
                    + response.getContentAsString() + "\n"
            );
            return response;
        }
    });
}
```

Snippet 4. Setting up transaction monitor for intercepting and saving all data.

The transaction monitor also saves cookie data since all the response headers are saved and also acquires information about the referrer.

Before advancing further it is always recommended to enumerate the headless browser properties to detect any potential configuration flaws. To do the testing sequence for HTMLUnit, the javascript engine has to be enabled temporarily. The browser properties enumeration can be achieved by the code snippets in **appendix I**. The code snippet must be served on a webserver inside script tags and the URL address of the script location is the input for the crawler. Alternatively, HTMLUnit methods can be used to verify whether the values are expected, with the code snippet in **appendix II**. The principles of creating an evasive proof browser profile are discussed in paragraph **3.3.1** and **table 3** contains the browser properties that are most often checked and are therefore minimally required to be implemented.

4.2.2 Phase two

This phase is mainly responsible for extracting the dynamic content and executing it in a sandboxed environment. Since HTMLUnit parses the HTML content it is effectively possible to extract all the DOM elements, their ID values, their class values and most importantly their content. Furthermore, all the script data to be sent to the script sandbox can be successfully carved. It is important to save the data in ECMA script format (*escaping the strings according to ES6 standards*), which can be parsed by the javascript sandbox (**Snippet 5**).

```
for(String tag : htmlTagsList) {
    DomNodeList<DomElement> domElements = htmlPage.getElementsByTagName(tag);
    for (DomElement domElement : domElements) {
        if (domElement.getId().length() >0) {
            //simpleFrameRunner
            simpleResultsDOMbyId.put(posOfDomPointerById, Arrays.asList(domEle-
ment.getId(),StringEscapeUtils.escapeEcmaScript(domElement.getTextContent())));
            posOfDomPointerById++;
        }else if(domElement.getAttribute("class").length() > 0 &&
        domElement.getChildElementCount() == 0 && domElement.getTextCon-
tent().trim().length() >0){
            //domElement.getAttribute("class"); domElement.getTextContent();
            simpleResultsDOMbyClass.put(posOfDomPointerByClass, Ar-
rays.asList(domElement.getAttribute("class"),
                    StringEscapeUtils.escapeEcmaScript(domElement.getTextCon-
tent())));
            posOfDomPointerByClass++;
        }
    }
}
DomNodeList<DomElement> domElements = htmlPage.getElementsByTagName("script");
for (DomElement domElement : domElements) {
    if (domElement.getId().length() >0) {
        simpleResultsDOMbyId.put(posOfDomPointerById, Arrays.asList(domEle-
ment.getId(),
                StringEscapeUtils.escapeEcmaScript(domElement.getTextCon-
tent())));
        posOfDomPointerById++;
    }
}
for (DomElement domElement : domElements) {
    if(domElement.getTextContent().length() >0) {
        String stringToSeek = domElement.getTextContent();
        //Push js excluding whitelist
        if(exceptions.parallelStream().noneMatch(stringToSeek::contains)) {
            simpleResultsJS.put("//" + posOfJsPointer, domElement.getTextCon-
tent());
            posOfJsPointer++;
        }
    }
}
```

Snippet 5. Extracting HTML DOM components.

MalwareJail[24] is capable of executing dynamic scripts in a sandboxed environment with most of the browser properties present. It utilizes node.js[25] javascript environment and its sandboxing features. The purpose of extracting the elements is to recreate it inside the script sandbox. The script sandbox evaluates the script and the output is an event log and the last iteration step of the script. The same browser profile used by HTMLunit must be implemented here as well. This can be done by appending the browser header string to *./env/agents.js* file. Next, the extracted DOM elements need to be appended to the javascript file which will be the input for the sandbox (**Snippet 6**). Furthermore, all of the javascript content itself must be extracted including the removal of script tags and finally the data extracted must be appended to the virtual page input file after the extracted DOM data.

```
for (Map.Entry<Integer, List<String>> entry : simpleResultsDOMbyId.entrySet()) {
    //String key = entry.getKey(); //String value = entry.getValue();
    String key = entry.getValue().get(0);
    String value = entry.getValue().get(1);
    fr.write("document._addElementById(\"" + key + "\",\"" + value + "\");\n");
}

for (Map.Entry<Integer, List<String>> entry : simpleResultsDOMbyClass.en-
trySet()) {
    //String key = entry.getKey(); //String value = entry.getValue();
    String key = entry.getValue().get(0);
    String value = entry.getValue().get(1);
    fr.write("document._addElementByClass(\"" + key + "\",\"" + value +
"\");\n");
}
```

Snippet 6. Appending DOM elements to the virtual page.

[24] Malware-jail. https://github.com/HynekPetrak/malware-jail
[25] Node JS sandbox Environment. https://nodejs.org/api/vm.html

Using the above functions in **Snippet 6** it is possible to add the DOM elements by id or class name and their content to the virtual page to be executed by MalwareJail.

The entire process described in this (**4.2.2**) section must be invoked on all the frames found inside the webpage – this means all the frames found must be iterated and their content must be extracted to acquire any hidden content (**Snippet 7**).

```
List<FrameWindow> window = htmlPage.getFrames();
Log.logger.info("Found frames: " + window.size());
simpleParse(htmlPage);
if(window.size() > 0){
    for (FrameWindow frameWindow : window) {
        HtmlPage subpage = (HtmlPage) frameWindow.getFrameElement().getEn-
closedPage();
        simpleParse(subpage);
    }
}
```

Snippet 7. Framerunner code example.

Lastly, the sandbox can be executed with the virtual page created (**snippet 8**).

```
Runtime runtime = Runtime.getRuntime();

List<String> commandList = new ArrayList<>();
commandList.add("node");
commandList.add("./malware-jail/jailme.js");
commandList.add("--h404");
commandList.add("-t");
commandList.add("15000");
if(configFile.length() != 0) {
    commandList.add("-c");
    commandList.add(configFile);
}
commandList.add("./malware/" + folder + File.separator + fileName);

String[] commands = new String[commandList.size()];
commands = commandList.toArray(commands);

process = runtime.exec(commands);
Log.logger.info("Process at " + process.pid() + " | CMD: " + process.info().com-
mandLine());
```

Snippet 8. Initialize Malware jail.

In the current example (**Snippet 8**), the default IE on windows 10 profile is loaded, which has to match the browser profile used in HTMLunit. Also, the default BOM environment is used with the scanning delay of 15 seconds and all the requests to the internet get a response of HTTP code 404 to ensure operational security. If applicable, this option can be disabled to retrieve any stage one or stage two payloads. In the scope of CTI validation the HTTP 404 is strongly recommended to use for operational security purposes. The script sandbox timeout is intended to execute scripts which use "***sleep(time)***" functions also called timed execution to evade sandboxing. By increasing the sandbox timeout the process will get delayed and therefore requires mitigation, the solution is discussed in subject **4.2.4 Snippet 12**.

Eventually, in the end of this phase the prototype honeyclient has produced static analysis results, end state files of the javascript sandbox, log files containing all the steps taken, list of URL addresses extracted from static content and list of URL addresses extracted during script sandboxing or after the sandbox has finished.

4.2.3 Phase three

In the last phase the evaluation of the static and dynamic content is performed through signatures. A playbook must be chosen which will use signatures belonging to specific groups. The chosen playbook will invoke all the defined rules in its hierarchy. After the scanning phase, the evaluator checks the matched rules and stores their impact score. Based on the requirements and the scope of the implementation YARA[26] signatures fit perfectly. (**Snippet 9**) YARA is executed with parameters "*msLw*" to acquire all the information about the matches, such as match offset, length, count and line.

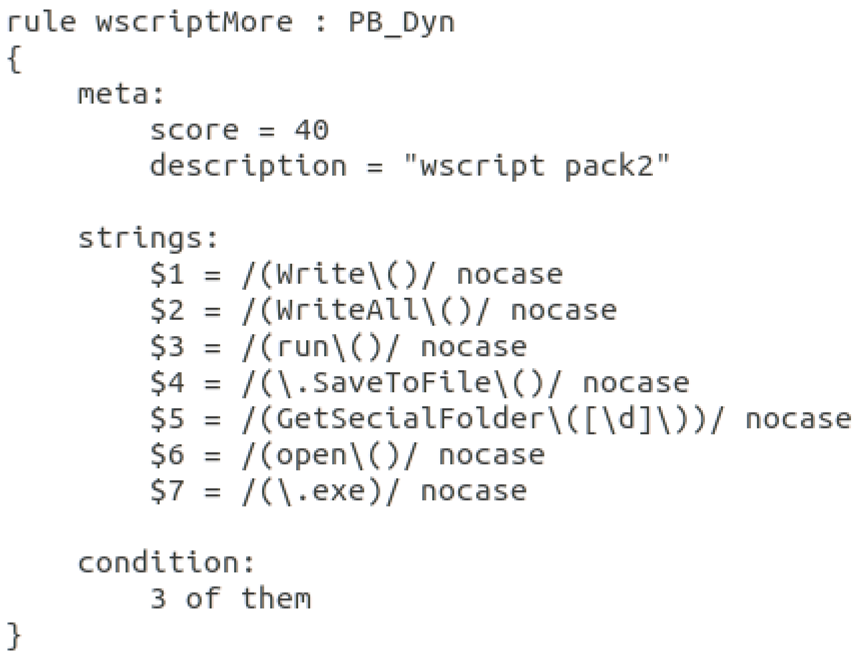
```
Runtime runtime = Runtime.getRuntime();

List<String> commandList = new ArrayList<>();
commandList.add("yara");
commandList.add("-m");
commandList.add("-s");
commandList.add("-L");
commandList.add("-w");
if(tags.size() != 0){
    for(String tag : tags){
        commandList.add("-t");
        commandList.add(tag);
    }
}
commandList.add("./yararules/"+playBookName);
commandList.add("./malware/" + malwareSamplesPath + File.separator + "results");

String[] commands = new String[commandList.size()];
commands = commandList.toArray(commands);
Log.logger.info("Initializing YARA " + malwareSamplesPath + File.separator +
"results/ <all files>");
Process process;

process = runtime.exec(commands);
Log.logger.info("Process at " + process.pid() + " | CMD: " + process.info().com-
mandLine());
```

Snippet 9. Initializing YARA scanner.

The layout of the rules must be compliant to YARA standards. The requirements for the current implementations are two meta fields: score and description. Atleast one playbook

[26] YARA. https://virustotal.github.io/yara/

tag must be also present and can not be empty. The YARA tag is equal to the playbook tag and seperates the rule name with a colon. The compliant format is presented in **Figure 9**.



```
rule wscriptMore : PB_Dyn
{
    meta:
        score = 40
        description = "wscript pack2"

    strings:
        $1 = /(Write\()/ nocase
        $2 = /(WriteAll\()/ nocase
        $3 = /(run\()/ nocase
        $4 = /(\.SaveToFile\()/ nocase
        $5 = /(GetSecialFolder\([\d]\))/ nocase
        $6 = /(open\()/ nocase
        $7 = /(\.exe)/ nocase

    condition:
        3 of them
}
```

Figure 9. YARA Signature layout.

The signature in the **figure 9** is member of the playbook: PB_Gen. It has an impact score of 40 and the group membership is defined by simply being in the correct folder. Currently, this signature is in folder number three, which means group 3 (**subject 3.3.3 table 4**). After the scanning process the evaluator outputs (**Snippet 10**) the final score and based on that the CTI entry is updated if needed.

```
 File file = new File("./results.csv");
FileWriter fr = new FileWriter(file, true);

for(Match match : parsedMatches){
    completeScore += match.getScore();
}
String verdict = "no_matches";
if(completeScore <= 50 && completeScore > 1){
    verdict = "Undetermined";
}else if(completeScore <= 74 && completeScore > 51){
    verdict = "Suspicious";
}else if(completeScore >= 75){
    verdict = "Malicious";
}

String writeable = parsedMatches.get(0).getJobid() + "," +
parsedMatches.get(0).getCurrentJobFull()+ "results/ <all files> "+
        "," + completeScore + "," + verdict + "\n";

String writeableLog = parsedMatches.get(0).getJobid() + "," +
parsedMatches.get(0).getCurrentJobFull()+ "results/ <all files> "+
        "," + completeScore + "," + verdict;

fr.write(writeable);
fr.close();
Log.logger.info(writeableLog);
```

Snippet 10. Evaluation process of all the static and dynamic matches.

The output format is a comma separated file containing the following information:

- JobUnique ID
- Samples path for YARA scanner
- Final score of all matched signatures scores
- Verdict

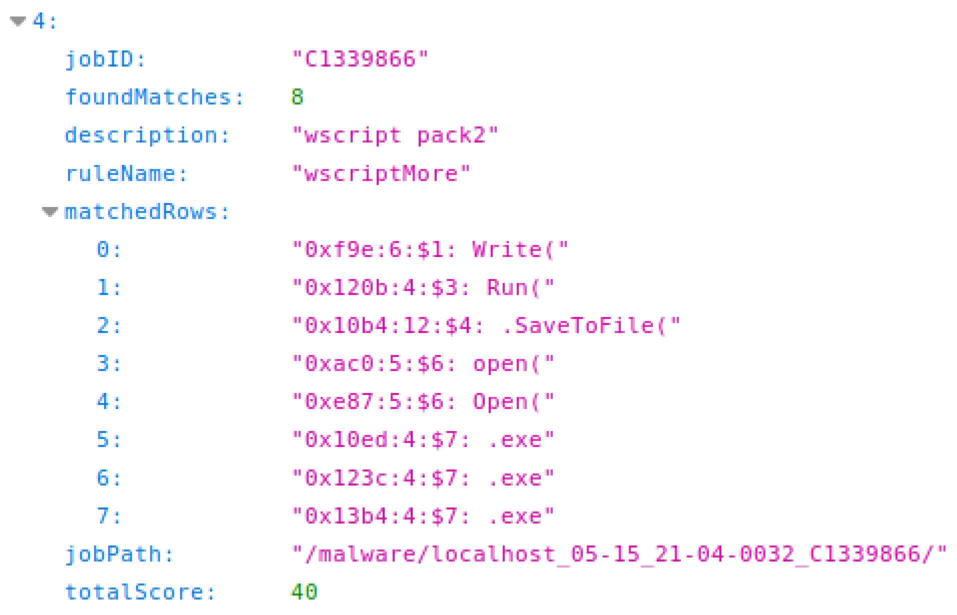
4.2.4 Execution and debugging

The final stage is to start the prototype honeyclient and the entire program flow (**Snippet 11**) by initializing a connection to the target web host. An example of the healthy log output of a single instance finishing successfully is in **appendix III**.

```
BaseInteretExplorer interetExplorer = new BaseInteretExplorer(thisJobUid);
Log.logger.info("(" + thisJobUid + ")" + "Starting job " + uri);
WebClient webClient = interetExplorer.getWebClient();
HtmlPage htmlPage = webClient.getPage(uri);
WebResponse webResponse = htmlPage.getWebResponse();
WebRequest webRequest = webResponse.getWebRequest();
webClient.close();
this.staticContent = interetExplorer.getStaticContent();
```

Snippet 11. Initializing a single instance of the program flow.

For the purpose of the playbook learning process a debugger option is implemented with the aim to help to adjust the matching strings of YARA rules or change scores accordingly. This option leaves all the evidence files on the disk and dumps the entire YARA match history in JSON format (**Figure 10**). With many samples, the disk can get full quickly. The JSON information is generated because the signature with the name "*wscriptMore*" has successfully identified the malicious content. The matching YARA signature itself for the corresponding JSON debug information (**Figure 10**) file is presented in **Figure 9** *YARA Signature layout*.



```
▾ 4:
    jobID:          "C1339866"
    foundMatches:   8
    description:    "wscript pack2"
    ruleName:       "wscriptMore"
  ▾ matchedRows:
      0:            "0xf9e:6:$1: Write("
      1:            "0x120b:4:$3: Run("
      2:            "0x10b4:12:$4: .SaveToFile("
      3:            "0xac0:5:$6: open("
      4:            "0xe87:5:$6: Open("
      5:            "0x10ed:4:$7: .exe"
      6:            "0x123c:4:$7: .exe"
      7:            "0x13b4:4:$7: .exe"
    jobPath:        "/malware/localhost_05-15_21-04-0032_C1339866/"
    totalScore:     40
```

Figure 10. Example of JSON debug information of a single rule hit.

In the **figure 10** currently the **forth** signature match info is collapsed of the total signature matches list. A match is an object which can be appended to a corresponding match "*ArrayList<match>*". Furthermore, information about how many times this signature has matched is 8 in **Figure 10**. The total score per match is acquired from the YARA signature metadata field *totalScore*. *MatchedRows* indicate the string offsets from where the signature match occurred. If a signature matches in different files, the match is appended to the total match array. The **final score** of the scanning job process is calculated based on the matches found in the match list. Every job has its own match list. Every match has a *totalScore* parameter and it is added to the **final score**, which ultimately is the verdict about the potentially malicious website. The *totalScore* parameter refers to all the matches of the single YARA rule according to the regex and the logical condition. Lastly, the jobID is the threads unique identifies, which solves the task omitted.

The final property of the prototype honeyclient is the ability to multithread (**Snippet 12**). Implementing multithreading requires unique filenames for file reads, file writes and for simultaneously acquiring the samples, extracting them, storing them and writing them to the corresponding virtual page for the specific job on the disk for the MalwareJail to be parsed. Every virtual page is unique to the jobID and the results of the static and dynamic analysis must be scanned separately. The only file that is common amongst all Threads is the *results.csv* which contains the verdict of every job completed.

```
jobs.addAll(Functions.parsedInputFile());
ExecutorService executor = Executors.newFixedThreadPool(threads);
while(!jobs.isEmpty()) {
    UUID idOne = UUID.randomUUID();
    String[] shortUID = idOne.toString().split("-");
    String uid = shortUID[0].toUpperCase();
    String job = jobs.remove();
    executor.submit(() -> {
        try {
            //For Training mode use 1 thread! "training=false" by default
            WebsiteValidator.setTrainingModeOn = training;
            WebsiteValidator ws = new WebsiteValidator(uid);
            ws.rootFunctionsCaller(new URL(job));
        } catch (Exception e) {
            Log.logger.log(Level.SEVERE, e.getMessage(), e);
        }
    });
}
```

Snippet 12. Multithreading feature.

The input for the honeyclient thread farm is a simple URL list separated by line brakes. Achieving a unique program flow for every thread is to append the corresponding unique ID to every operation which could lead to thread collision. A valid program flow log output is in **appendix III.** The purpose of the multithreading feature is to improve the speed of the validation process and allow concurrent jobs to complete with the goal of maintain a low interaction honeyclient main feature, speed. Due to the fact that many scripts invoke timeouts to the obfuscated scripts the script sandbox takes time to complete and therefore multithreading is required.

4.3 Overview of phases

Already existing open source components can be successfully adapted to the proposed architecture (**Figure 11**). The headless browser and sandbox environment browser profiles must match. Furthermore, validating the browser profiles can be achieved through javascript, the code snippet is available in appendix I. The two input options can be used either way, but currently the PCAP input is intended to create playbooks, which are final entities. These playbooks are invoked after the headless browser and dynamic script environment have finished execution. All the components are tied together with using java programming language and the prototype source code is available on **Github**[27]. The following figure represents the workflow of the honeyclient.

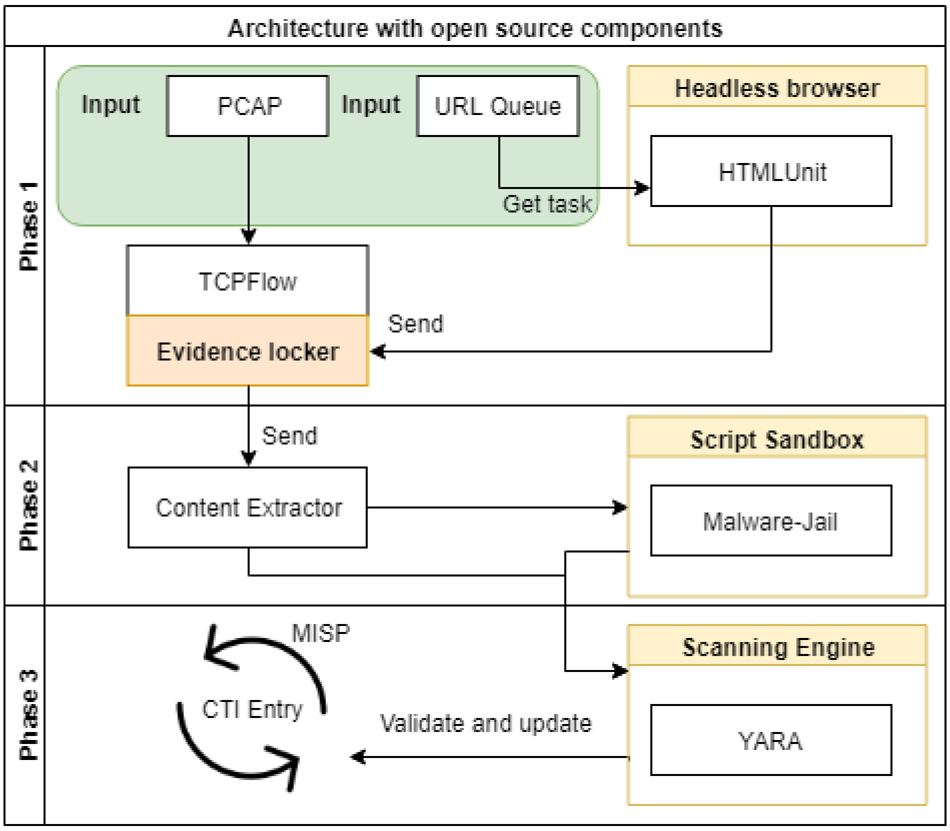


Figure 11. Open source components in three phases.

The ultimate goal of the architecture is to validate and update CTI DB entries, specifically HTTP IOC's severity levels and their scores. The requirement to do so is to have context about what to detect with YARA signatures and commonly well maintained CTI DB entries have that context for every IOC.

4.4 Conclusion

This paragraph answers the RQ: How to implement the methodology using open source components? **The unique difference** amongst other low interaction honeyclient is that this prototype utilizes a pure script sandbox which is fully automatically populated by the headless browser and besides that maintaining the working speed of a classical honeyclient. An alternative approach proposed is to use a minimal set of signatures with the aim to detect specific events from a malicious web based IOC. Detection is depending on the level on context the IOC's have in the CTI database. Similar IOC's can be validated in bulk. The main contribution is **2000 (+-100)** lines of open source code which fully automates the entire validation process of the prototype honeyclient. It also provides a rich debugging feature, which exports the signature matches, matched lines and the matched content itself to the disk in JSON format and this helps to debug and improve playbooks with less amount of time.

[27] **GitHub Project**: https://github.com/icoscx/WebsiteRaker

5 Analysis of the results and validation

The current subject answers the following RQ: How effective is a context oriented fully automated CTI validating honeyclient? Context orientation refers to seek for certain unique elements or events which are already known from the CTI database and can be applied for validation. The analysis and validation of the results takes in account the main constraint of the methodology, which requires a well maintained and context rich CTI DB.

5.1 Validating browser profiles and transaction monitor

Testing the browser profile with code snippets from **appendix I** and **appendix II** are used to verify that the browser configuration for this analysis topic is ready. The configuration code can be observed from the authors Github [39] and will not be discussed in detail.

Next, a test case is set up for redirect handling and cookies management. The goal is to verify if the transaction manager can extract all the information required.

```
1.php
<?php header('Location: http://localhost/2.php', true, 301); ?>
<!DOCTYPE html>
<head><title>Test 301</title></head>
<body>
<script>//must be ignored because of header location
window.location="http://localhost/3.php?id=2"; </script>
</body>

2.php
<!DOCTYPE html>
<head><title>TestMetaRefresh</title></head>
<?php echo "referer\n";
echo $_SERVER['HTTP_REFERER'];
echo "<meta http-equiv='refresh' content='2;url=http://localhost/3.php?id=2'>";
?>
<script>//must be ignored because of meta refresh
window.location="http://localhost/3.php?id=2"; </script>

3.php
<!DOCTYPE html>
<head><title>test Iframe and Cookies</title>
<?php setcookie("Victim", $value, time()+6000, "/");
setcookie("pathThestS", $value, time()+6000, $_SERVER['REQUEST_URI']); ?>
</head>
<body>
<script type="text/javascript"> var a = 2; var b = 3; var c = eval("a*b");
document.write(c);document.write("http://localhost/index.php"); </script>
<iframe id="secret" src="http://localhost/4.php?1"></iframe>
<div id="first">entry</div>
</body>

4.php
<!DOCTYPE html>
<head><title>Inside Frame</title></head>
<body>
<?php echo "referer\n";
echo $_SERVER['HTTP_REFERER'];?>
<script>document.write("<p>para</p>");</script>
<div id="4withid"> 4 - withid</div>
</body>
```

Snippet 13. Transaction monitor tests.

In **Snippet 13** the honeyclient has to go to 1.php without touching the content. This is defined by the RFC for the HTTP code 301 and the same applies for 2.php but the redirection is achieved by a Meta Refresh. For both the meta refresh and HTTP 301 redirect the referrer field must not be set. 3.php Sets cookies and presents an Iframe. According to the HTTP RFC, traversing a frame a referrer must be set to the request to the frame. The content of the iframe on page 4.php must be parsed within 3.php. When opening the frame the referrer is the main frame. Lastly, the cookies must be set for 3.php only even if the cookie path is "/". Cookies are not allowed to traverse iframes. The log file and content extracted by the transaction monitor is in **appendix VI**, which confirms all the requirements stated.

The transaction monitor is implemented correctly: it saves every transaction required and the prototype honeyclient operates according RFC's like real browsers do.

5.2 Scenario

The validation of the automated tool is performed against a large dataset 39 184 malware samples acquired from Github[28]. Two playbooks were generated based on 500 randomly selected samples for each, one generic and one specific threat oriented. The samples used to train the playbook are excluded from the validation process. The tests were carried out on Ubuntu 18.04 LTS with 4 cores at 3,4GHz and 8 GB of RAM. The OS itself was running inside a Vmware workstation to avoid any possible infection. In the following scenario Company X has a plan to validate its CTI DB since the amount of entries has grown over 39 184 and is hindering the incident response process by generating many false positive events. The Company X uses a well maintained CTI database and the CTI DB administrator is implementing the proposed methodology to revalidate its entries automatically, systematically and constantly. The system administrator is skeptical and every validation result that yields over 50 as a score, the administrator would mark as potentially malicious.

5.2.1 Playbook: Generic

The generic playbook contains 30 YARA signatures which aim to detect generic static content and suspicious URL addresses. The signature set for the playbook can be found on github[29] and the file where to initiate is rakerConfig.json[30]. Next, the playbook is applied on the known bad 38 184 extracted samples from PCAPs for revalidation purposes. Generic signatures are less likely to detect malicious content. In the current test, none of the post sandbox oriented signatures are used. The signatures used are in **appendix IV.**

Table 5. Generic playbook

Result	Count
Undetermined Score < 50	26 530
Suspicious 50 < Score < 75	6212
Malicious Score > 75	6440

[28] **Malware collection.** https://github.com/icoscx/WebsiteRaker/tree/master/malware_samples
[29] **Playbook_start** is the main YAR file. The playbook is selected by a TAG in configuration file https://github.com/icoscx/WebsiteRaker/tree/master/yararules
[30] Initiate Playbook configuration. https://github.com/icoscx/WebsiteRaker/blob/master/rakerConfig.json

The results are expected in Table 6 - Only 12 652 samples were deemed likely malicious and the true positive percentage is **31,52 %**.

Next, 950 clean and still working websites[31] are acquired for the precision and recall test. The datasets contains 43 200 000 websites and 950 still working websites are selected. 50 random samples are added to the test totalling the test set to 1000 URLs/samples.

Table 6. FP, TP of PB_Gen

	50 malicious samples	950 clean websites
Classification: "Malicious"	22 (TP)	165 (FP)
Classification: "Normal"	28 (FN)	785 (TN)

50 samples are known bad samples, 950 are not. From 950 the generic playbook detected 165 clean websites as malicious.

Precision: TP/(TP+FP) = 22/(22+165) = 12%

Recall: TP/(TP+FN) = 22/(22+28) = 44%

Generic playbook is resulting in average number of matches of which 12% were correctly classified and from all of the positive results 44% were detected correctly.

5.2.2 Playbook: Wscript

Since in the testing scenario the Company X is using windows OS and only Internet Explorer according to the company policy, the malware is all Internet Explorer oriented. Next, the playbook Wscript contains only five signatures and the generic playbook signatures are not invoked at all. The signatures used are in **appendix V**.

Table 7. WScript oriented playbook

Result	Count
Undetermined Score < 50	6270
Suspicious 50 < Score < 75	5015
Malicious Score > 75	24744

By using WScript detection oriented general signatures the results are impressive (Table 7) by detecting **75,36 %** of the malicious content as true positive matches and the false positive ratio is less than 25 %. It is also to be noted that without executing javascript (disabling

[31] DATASET of WWW. https://www.bigdatanews.datasciencecentral.com/profiles/blogs/big-data-set-3-5-billion-web-pages-made-available-for-all-of-us

malware jail and HTMLunit Rhino engine) the WScript playbook which is purely intended for post sandbox matches, detects 2,6% of 38 184 samples.

Next, 950 clean and still working websites[32] are acquired for the precision and recall test. The datasets contains 43 200 000 websites and 950 still working websites are selected. 50 random samples are added to the test totalling the test set to 1000 URLs/samples.

Table 8. FP, TP of PB_WScript

	50 malicious samples	950 clean websites
Classification: “Malicious”	41 (TP)	18 (FP)
Classification: “Normal”	9 (FN)	932 (TN)

50 samples are known bad samples, 950 are not. From 950 the WScript playbook detected 18 clean websites as malicious.

Precision: TP/(TP+FP) = 41/(41+18) = 69%

Recall: TP/(TP+FN) = 41/(41+9) = 82%

WScript oriented playbook is resulting in high number of matches of which 69% were correctly classified and from all of the positive results 82% were detected correctly.

5.3 Measuring the multithreading feature

The measurement of the multithreading features has two purposes:

- Time factor dependence of the timeout
- Detection impact of various timeout values

Network delay: 0 ms

Table 9. Time consumption for validating 39 184 samples in cost of detection accuracy.

SandBox timeout	Time with 15 threads	Time with 5 threads	Time with 1 thread
15 seconds	03H:12MM:22SS	05H:51MM:12SS	09H:21MM:51SS
45 seconds	03H:37MM51SS	06H:18MM:01SS	10H:01MM:09SS
90 seconds	03H:42MM43SS	06H:25MM:54SS	10H:09MM:59SS
SandBox timeout	Detection ratio with WScript playbook	Detection ratio with WScript playbook	Detection ratio with WScript playbook

[32] DATASET of WWW. https://www.bigdatanews.datasciencecentral.com/profiles/blogs/big-data-set-3-5-billion-web-pages-made-available-for-all-of-us

15 seconds	**75,36 %**	same	same
45 seconds	**78,39%**	same	same
90 seconds	**79,1%**	same	same

Firstly, the multithreading feature (**Table 8**) results in three times faster scanning time than with one thread and helps to automatically and systematically validate the CTI DB-s on daily bases with shorter validation times. The low interaction honeyclient property has been confirmed, it works fast besides cuckoo sandbox. For example, the analysis time for every sample in Cuckoo would take about three times longer.

Secondly, some of the javascript based malware relays on timeouts or sleep functions to avoid detection by low interaction honeyclients. Furthermore, current tests were all executed with 15000 ms which is 15 seconds. The effective time and detection ratio is between 45 seconds and 15 seconds and that would be 30 seconds. The thread count will not affect the detection ration. Also, some of the malware samples use random functions to create the timeout or sleep function time spans and there is a slight error margin present here. Lastly, setting the sandbox timeout at a specific value, does not mean it waits 45 seconds always. It will finish when the code execution has finished or exceeds the predefined timeout.

5.4 Conclusion of analysis

WScript oriented playbook performed **3 times** better than the generic playbook. The post sandbox evaluation is effective due to the fact that most often the same principles and vectors are abused by malware creators and this will all emerge after sandboxing. The script sandbox is the key to overcome any obfuscation challenges. All these 39 184 samples have a different obfuscation tactic and javascript packing method and that does not matter if the sandbox is able to execute the script and output the end state. This methodology proposed a sandboxed environment fully ignoring any attempt to detect and de-obfuscate javascript by conserving the properties of a low interaction honeyclient. The prerequisite of using this method is to have a well maintained and context rich CTI DB. This subject has answered the following RQ: How effective is a context oriented fully automated CTI validating honeyclient? It works fast and can be tasked to perform simultaneous tasks. The results are dependent on the CTI context and the accuracy of signature design. Focusing on post malware sandbox detection yields more accurate results and less false positives. The parametrization time is only needed to spend on creating and learning playbooks. Playbooks can also be easily extended and changed for slight changes if a new task is similar to the previous one. Playbook debugger logs the rule matches in detail, saves all the match able content and ultimately enhances the playbook design process.

6 Conclusion and future work

Over half of the Thesis focuses on already studied researches. This is due to the fact that the field of detecting or analyzing malicious websites is very vast and has had a lot of researches done before. Therefore, it was crucial not to re-invent the wheel and not to overlap with any already completed experiments with the same aspects as this Thesis.

Future work: Having high end servers and unlimited resources the methodology can be implemented with Cuckoo Sandbox, to achieve even more improved results. It can also be a two dimensional effort, if the low interaction honeyclient deems the web related IOC suspicious cuckoo sandbox can be used for a layered coverage. The low interaction honeyclient works fast and is less resource demanding, therefore it can be the first layer of validation to filter away the content which does not require much effort to detect. Using just Cuckoo is resource demanding and with many CTI entries to be validated the effort does not pay off.

Beside different playbooks it is also possible to automate the selection of multiple scanning profiles. For example, some of the malware samples are meant to execute in specific years, so multiple profiles could be iterated through presenting different years, in the future or in the past. Secondly, it has been identified, that some malware does not run in a BOM environment, so another iteration could be a BOM free profile. Currently, this methodology is used to revalidate already known malicious web based threat indicators. The automated solution can also have programmed to download all the samples (payloads) from the server and send them automatically to automatic malware analysis tools, such as IRMA and Cuckoo. This furthermore enriched the CTI DB with indicators extracted from the payload. After that, automatic IDS signatures could be created from the CTI DB to monitor for any possible matches. Thirdly, in the current prototype no DOM interaction has been implemented, such as mouse movements and clicks or pressing keyboard keys. This is another layer of protection and overcoming that can trigger more web based malware.

Conclusion

This thesis aimed to design a methodology to validate known cyber threat intelligence database entries with a fast working honeyclient. The methodology was successfully implemented and tested. The prototype code is open source and can be adopted to anyone's needs. Furthermore, the prototype does not reinvent the wheel and uses already developed open source components. The current automated prototype was written in JAVA programming language, the contribution is the automation code by the thesis author, which is roughly **2000** lines. The validation of this methodology was performed on offline malware samples, since malicious websites are taken down mostly within 24 hours and malicious threat actors change their payload and exploitation servers quickly by using domain generation algorithms. Therefore, for development purposes avoiding randomness and uncertainness of the website being still malicious decreases.

Lastly, the honeyclient is used in a production environment by the thesis author himself and is a valuable addition to the tools used by a SOC analyst on daily basis. The results of the tool in the production environment are depending on the depth level of the playbook signatures and the complexity of the targeted web hosts.

References

[1] Netcraft Ltd, "January 2019 Web Server Survey," [Online]. Available: https://news.netcraft.com/archives/2019/01/24/january-2019-web-server-survey.html. [Accessed Feb 2019].

[2] w3techs, "Usage of content management systems for websites," [Online]. Available: https://w3techs.com/technologies/overview/content_management/all. [Accessed Feb 2019].

[3] Symantec Corporation, "Internet Security Threat Report Volume 23," [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf. [Accessed Feb 2019].

[4] ENISA, "ENISA Threat Landscape Report 2018," [Online]. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018/at_download/fullReport. [Accessed Feb 2019].

[5] SANS, "The Sliding Scale of Cyber Security," [Online]. Available: https://www.sans.org/reading-room/whitepapers/ActiveDefense/sliding-scale-cyber-security-36240. [Accessed 16 Feb 2019].

[6] C. Sauerwein, C. Sillaber, A. Mussmann and R. Breu, "Threat Intelligence Sharing Platforms: An Exploratory," [Online]. Available: https://wi2017.ch/images/wi2017-0188.pdf. [Accessed 27 Feb 2019].

[7] Birhanu Eshete and V. N. Venkatakrishnan. 2014. WebWinnow: leveraging exploit kit workflows to detect malicious urls. In Proceedings of the 4th ACM conference on Data and application security and privacy (CODASPY '14). ACM, New York, NY, USA, 305-312. DOI: https://doi.org/10.1145/2557547.2557575

[8] P. Pols, "The Unified Kill Chain - Designing a Unified Kill Chain for analyzing, comparing and defending against cyber attacks," [Online]. Available: https://www.csacademy.nl/images/scripties/2018/Paul_Pols_-_The_Unified_Kill_Chain_1.pdf. [Accessed 28 Feb 2019].

[9] D. Stuttard and M. Pinto, The Web Application Hacker’s Handbook, vol. II, Indianapolis: John Wiley & Sons, Inc, 2011, pp. 431-498.

[10] B. Sullivan and V. Liu, Web Application Security, New York: McGraw-Hill, 2011, pp. 11-13.

[11] J. CHANG, KRISHNA K. VENKATASUBRAMANIAN, A. G. WEST, I. LEE and U. o. Pennsylvania, "Analyzing and Defending Against Web-based Malware," [Online]. Available: https://kven.me/papers/acmsurvey13.pdf. [Accessed 2 Mar 2019].

[12] Trendmicro, "Ransomware as a Service," [Online]. Available: https://documents.trendmicro.com/assets/resources/ransomware-as-a-service.pdf. [Accessed 4 Mar 2019].

[13] T. Luo and X. Jin, "Next Generation Of Exploit Kit Detection By Building Simulated Obfuscators," Palo Alto Networks Inc, [Online]. Available: https://pdfs.semanticscholar.org/30e2/675ec065e4c9285f1e0ea9a8560323c6121b.pdf. [Accessed 5 Mar 2019].

[14] Mansoori, M., Welch, I. and Fu, Q. (2014). YALIH, Yet Another Low Interaction Honeyclient. In Proc. Twelfth Australasian Information Security Conference (AISC 2014) Auckland, New Zealand. CRPIT, 149. Parampalli, U. and Welch, I. Eds., ACS. 7-15

[15] V. BARTL, "A CLIENT HONEYPOT," MASARYKOVA UNIVERSITY, [Online]. Available: https://is.muni.cz/th/dtmhv/thesis.pdf. [Accessed 6 Mar 2019].

[16] A. Jhilam Biswas, "Analysis of Client Honeypots," Manipal Institute of Technology, [Online]. Available: https://pdfs.semanticscholar.org/c25f/97b9a5313bc29194949e720cda96ac300d19.pdf. [Accessed 10 Mar 2019].

[17] P. Karsai and C. Benyi, "Case study: headless browsers in web forum spam," Vamasoft e-Security, [Online]. Available: http://vamsoft.com/downloads/articles/vamsoft-headless-browsers-in-forum-spam.pdf. [Accessed 7 Mar 2019].

[18] F. Gadaleta, Y. Younan and W. Joosen, "BuBBle: A Javascript Engine Level Countermeasure against Heap-Spraying Attacks," [Online]. Available: http://cd80.ca/files/bubble.pdf. [Accessed 10 Mar 2019].

[19] G. Vlot, "Automated data extraction: Detectin web-bot detection," Open University, 2018. [Online]. Available: http://www.open.ou.nl/hjo/supervision/2018-g.vlot-msc-thesis.pdf. [Accessed 14 Mar 2019].

[20] O. Ruwase and M. S. Lam, "A Practical Dynamic Buffer Overflow Detector," Standford University, [Online]. Available: https://suif.stanford.edu/papers/tunji04.pdf. [Accessed 14 Mar 2019].

[21] J. Friedman and M. Bouchard, "Defnitive Guide Cyber Threat Intelligence," CyberEdge, 2015. [Online]. Available: https://cryptome.org/2015/09/cti-guide.pdf. [Accessed 11 Mar 2019].

[22] A. Dell'Aera, "Thug: Python low-interaction honeyclient," [Online]. Available: https://buffer.github.io/thug/doc/. [Accessed 15 Mar 2019].

[23] C. Seifert, "HoneyC," Victoria University of Wellington, [Online]. Available: https://github.com/honeynet/honeyc. [Accessed 17 Mar 2019].

[24] Cuckoo Foundation, "Cuckoo Sandbox Book," [Online]. Available: https://media.readthedocs.org/pdf/cuckoo/latest/cuckoo.pdf. [Accessed 20 Mar 2019].

[25] Q. LI and G. Clark, Security Intelligence, Indianopolis: John Wiley & Sons, INC, 2015, pp. 154-159.

[26] R. Hes, R. Steenson and C. Seifer, "The Capture-HPC client architecture," 2010. [Online]. Available: https://ecs.victoria.ac.nz/foswiki/pub/Main/TechnicalReportSeries/ECSTR09-11.pdf. [Accessed 30 Mar 2019].

[27] M. Cova and C. K. a. G. Vigna, "Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code," 2010. [Online]. Available: https://sites.cs.ucsb.edu/~vigna/publications/2010_cova_kruegel_vigna_Wepawet.pdf. [Accessed 22 Mar 2019].

[28] L. Invernizzi, S. Benvenuti and P. M. Comparetti, "EVILSEED: A Guided Approach to Finding Malicious Web Pages," 2012. [Online]. Available: https://sites.cs.ucsb.edu/~vigna/publications/2012_SP_Evilseed.pdf. [Accessed 23 Mar 2019].

[29] M. T. Qassrawi and H. Zhang, "Detecting Malicious Web Servers with Honeyclients," 2011. [Online]. Available: https://pdfs.semanticscholar.org/6679/638fe42f1229b8e81047447a5f82b0e71ba7.pdf. [Accessed 21 Apr 2019].

[30] Y. Xue, J. Wang, Y. Liu, H. Xiao and J. Sun, "Detection and Classification of Malicious JavaScript via Attack Behavior Modelling," July 2015. [Online]. Available: http://people.sutd.edu.sg/~sunjun/Publications/ISSTA2015a.pdf. [Accessed 22 Mar 2019].

[31] T. Taylor, K. Z. Snow, N. Otterness and F. Monrose, "Cache, Trigger, Impersonate: Enabling Context-Sensitive Honeyclient Analysis On-the-Wire," 2016. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2017/09/cache-trigger-impersonate-enabling-context-sensitive-honeyclient-analysis-wire.pdf. [Accessed 10 Mar 2019].

[32] S. Kaur and H. Kaur, "Client Honeypot Based Malware Program Detection Embedded Into Web Pages," [Online]. Available: https://pdfs.semanticscholar.org/02ab/aced01c7ef81ad6b598fe10b9c18b67c6848.pdf. [Accessed 6 Mar 2019].

[33] Y. TAKATA, "Thesis: Analyzing Hidden Features of Web-based Attacks," February 2018. [Online]. Available: https://core.ac.uk/download/pdf/159504368.pdf. [Accessed 25 Apr 2019].

[34] C. Curtsinger, B. L. a. B. Zorn and C. Seifert, "ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection," 2015. [Online]. Available: https://yacin.nadji.us/classes/f16-adv-comp-sec/papers/15-zozzle.pdf. [Accessed 12 Mar 2019].

[35] B. Stock, B. Livshits and B. Zorn, "Kizzle: A Signature Compiler for Detecting Exploit Kits," July 2016. [Online]. Available: https://www.doc.ic.ac.uk/~livshits/papers/pdf/dsn16.pdf. [Accessed 05 Mar 2019].

[36] A. B. Sayamber and A. M. Dixit, "Malicious URL Detection and Identification," August 2014. [Online]. Available: https://pdfs.semanticscholar.org/0dd7/477bb1cb58b286d70b83af1e6e6495e7eb32.pdf. [Accessed 02 Feb 2019].

[37] H.-Y. Lin, T.-M. Koo, H.-C. Chang and Y.-T. Hsu, "Malicious Website Detection Based on Honeypot Systems," 2013. [Online]. Available: https://download.atlantis-press.com/article/6875.pdf. [Accessed 18 Feb 2019].

[38] J. R. Rocaspana, "SHELIA A Client HoneyPot for Client-Side Attack detection," 2007. [Online]. Available: https://www.cs.vu.nl/~herbertb/misc/shelia/shelia07.pdf. [Accessed 20 Jan 2019].

[39] I. Pure, "Github - WebsiteRakes," 15 05 2019. [Online]. Available: https://github.com/icoscx/WebsiteRaker. [Accessed 15 May 2019].

Appendix

I. Enumerating browser properties with JS

The following script was used to enumerate browser properties (*navigator*, *screen*, *window*) to check if the browser profile configuration is correct for HTMLUnit.

```
//any browser variable can be omitted to "var obj" to be enumerated
//Works for IE, Chrome, firefox, htmlunit (except for navigator)
var obj = navigator;
for (var key in obj) {
   let value = obj[key];
   document.writeln("<p>" + obj + "." + key + " = " + value);
   if(typeof value === 'object'){
      for (var key2 in obj[key]) {
         let value2 = obj[key][key2];
         document.writeln("<p> + >" + obj[key] + "." + key2 + " = " + value2);
         if(typeof value2 === 'object'){
            for (var key3 in obj[key][key2]) {
               let value3 = obj[key][key2][key3];
               document.writeln("<p> ++ >" + obj[key][key2] + "." + key3 + " = "
+ value3);
               if(typeof value3 === 'object'){
                  for (var key4 in obj[key][key2][key3]) {
                  let value4 = obj[key][key2][key3][key4];
                  document.writeln("<p> +++ >" + obj[key][key2][key3] + "." +
key4 + " = " + value4);
                  }
               }
            }
         }
      }
   }
}
document.write('<hr>htmlunit properties<hr>');
//for HTMLUnit navigator (its navigator does not follow the plugin
//and mimetype object convention, properties etc)
var obj = navigator;
for (var key in obj) {
   let value = obj[key];
   document.writeln("<p>" + obj + "." + key + " = " + value);
   if(typeof value === 'object'){
      for (var key2 in obj[key]) {
         let value2 = obj[key][key2];
         document.writeln("<p> + >" + obj[key] + "." + key2 + " = " + value2);
      }
   }
}
document.write('<hr>Plugins and mimetypes<hr>');
//Print plugins
document.writeln("<p>Plugins found: " + navigator.plugins.length +  "<p>");
for(var i=0; i<navigator.plugins.length; i++){
   document.writeln("<p>Name: " + navigator.plugins[i].name + '|' +
   "filename: " + navigator.plugins[i].filename + '|' +
   "description: " + navigator.plugins[i].description + '|' +
   "version: " + navigator.plugins[i].version);
   //print any supported mimetype for plugin
   for(var j=0; j<navigator.plugins[i].length; j++){
      document.writeln("<p>Mimetype for plugin - type: " + naviga-
tor.plugins[i][j].type + '|' +
      "suffixes: " + navigator.plugins[i][j].suffixes + '|' +
      "description: " + navigator.plugins[i][j].description + '|' +
      "enabledPlugin: " + navigator.plugins[i][j].enabledPlugin);
   }
}
```

II. Enumerating browser properties with HTMLUnit

Alternative approach to appendix I code snippet – does not require enabling JSengine

```
static void enumPlugins(WebClient webClient){

    for(PluginConfiguration plugin : webClient.getBrowserVersion().get-
Plugins()){
        Log.logger.info("desc: " + plugin.getDescription()
                + " fnma: " + plugin.getFilename() + " name:" + plugin.getName()
+ " ver:" + plugin.getVersion()
        );
        for(PluginConfiguration.MimeType mimeType : plugin.getMimeTypes()){
            Log.logger.info(mimeType.getDescription() + "||" + mimeType.getSuf-
fixes() + "||" + mimeType.getType());
        }
    }

}

static void enumFeatures(WebClient webClient){

    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append(webClient.getBrowserVersion().getUserAgent() + "|\n"
            + webClient.getBrowserVersion().getApplicationCodeName() + "|\n"
            + webClient.getBrowserVersion().getApplicationCodeName() + "|\n"
            + webClient.getBrowserVersion().getApplicationName() + "|\n"
            + webClient.getBrowserVersion().getApplicationMinorVersion() + "|\n"
            + webClient.getBrowserVersion().getBrowserLanguage() + "|\n"
            + webClient.getBrowserVersion().getBuildId()  + "|\n"
            + webClient.getBrowserVersion().getApplicationVersion() + "|\n"
            + webClient.getBrowserVersion().getCpuClass() + "|\n"
            + webClient.getBrowserVersion().getCssAcceptHeader() + "|\n"
            + webClient.getBrowserVersion().getHtmlAcceptHeader() + "|\n"
            + webClient.getBrowserVersion().getImgAcceptHeader() + "|\n"
            + webClient.getBrowserVersion().getNickname() + "|\n"
            + webClient.getBrowserVersion().getPlatform() + "|\n"
            + webClient.getBrowserVersion().getProductSub() + "|\n"
            + webClient.getBrowserVersion().getScriptAcceptHeader() + "|\n"
            + webClient.getBrowserVersion().getSystemLanguage() + "|\n"
            + webClient.getBrowserVersion().getUserLanguage() + "|\n"
            + webClient.getBrowserVersion().getXmlHttpRequestAcceptHeader() +
"|\n"
            + webClient.getBrowserVersion().getVendor() + "|\n"
            + webClient.getBrowserVersion().getPlugins().size() + "|\n"
            + webClient.getBrowserVersion().getBrowserVersionNumeric() + "|\n"
            + webClient.getBrowserVersion().getBuildId() + "|\n"
            + webClient.getBrowserVersion().isOnLine() + "|\n"
            + webClient.getBrowserVersion().getPixesPerChar()
    );

    Log.logger.info(stringBuilder.toString());

}
```

III. Normal flow of the honeyclient

The following lines represent the normal program flow of the honeyclient, currently set in training mode (excessive logging). Single thread output.

```
[13/05 17:15:06][info]Default IE Selected: Mozilla/5.0 (Windows NT 6.1; Tri-
dent/7.0; rv:11.0) like Gecko
[13/05 17:15:06][info][187BBEC8]Injecting Transaction Monitor
[13/05 17:15:06][info](187BBEC8)Starting job http://localhost/1.php
[13/05 17:15:06][info](187BBEC8)Last page Navigated to http://lo-
calhost/3.php?id=2 status: 200 OK date: Mon, 13 May 2019 14:15:06 GMT
[13/05 17:15:06][info]Found frames: 1
[13/05 17:15:06][info]Creating virtual page file: /home/ico/Desktop/Work/Web-
siteRaker-20190424T100537Z-001/WebsiteRaker/malware/localhost_05-13_17-15-
0006_187BBEC8/localhost_17-15-0006.js
[13/05 17:15:06][info]Processing command for NodeJS in folder localhost_05-
13_17-15-0006_187BBEC8 and file localhost_17-15-0006.js
[13/05 17:15:07][info]Process at 48546 | CMD: Optional[/usr/bin/node ./malware-
jail/jailme.js --h404 -c ./config_localhost_05-13_17-15-0006_187BBEC8.json
./malware/localhost_05-13_17-15-0006_187BBEC8/localhost_17-15-0006.js]
[13/05 17:15:07][info]Exporting results to: /results/localhost_17-15-0006.dy-
namic.log
[13/05 17:15:07][info]Exporting results to: /results/localhost_17-15-
0006.static.log
[13/05 17:15:07][info]No Dynamic URLs extracted from: localhost_05-13_17-15-
0006_187BBEC8/localhost_17-15-0006.js
[13/05 17:15:07][info]Initializing YARA localhost_05-13_17-15-0006_187BBEC8/re-
sults/ <all files>
[13/05 17:15:07][info]Process at 48563 | CMD: Optional[/usr/local/bin/yara -m -s
-L -w -t PB_Dyn ./yararules/playbook_dynamic.yar ./malware/localhost_05-13_17-
15-0006_187BBEC8/results]
[13/05 17:15:07][info]187BBEC8,/malware/localhost_05-13_17-15-0006_187BBEC8/re-
sults/ <all files> ,150,Malicious
[13/05 17:15:07][warning]Yara Debugger is enabled, massive log flood. USE for
training a playbook!
```

IV. Playbook generic

Group 1[33]:

Name	Match against	Condition	Score
evalfun : PB_Gen	$ = /(eval\(function)/ nocase $ = /(eval \(function)/ nocase	any of them	12
eval : PB_Gen	$1 = /(eval\()/ nocase	#1 > 10	8
evalflood : PB_Gen	$1 = /(eval\()/ nocase	#1 > 30	16
ccOninMozilla : PB_Gen	$ = /(@cc_on)/ nocase	any of them	7
iframetag : PB_Gen	$1 = /(iframe)/ nocase	#1 > 6	5
hiddeniframetag : PB_Gen	$1 = /(width:0|height:0|border:0)/ nocase $2 = /(border:none|border: none)/ nocase $3 = /(iframe)/ nocase $4 = /(position:absolute|position: absolute)/ nocase	$1 and $2 and $3 and $4	15
abusedtags : PB_Gen	$ = /(<svg onload=)/ nocase $ = /(<object onload=)/ nocase $ = /(<script onload=)/ nocase	any of them	10
charcode : PB_Gen	$1 = /(fromCharCode)/ nocase $2 = /(unescape\()/ nocase $3 = /(charCodeAt\()/ nocase	#1 > 30 or #2 > 30 or #3 > 45	20
manydocumentWrite : PB_Gen	$1 = /(document.write)/ nocase	#1 > 150	20
hex : PB_Gen	$a = /(0[xX][0-9a-fA-F]{2})/ nocase $b = /([xX][0-9a-fA-F]{2})/ nocase	#a > 50 or #b > 50	10
base64FunCalls : PB_Gen	$f = /(atob|btoa|;base64|base64,|encode64)/ nocase	$f	10
genericJsObfuscation : PB_Gen	$string0 = /eval\(([\s]+)?(unescape|atob)\(/ nocase $string1 = /var([\s]+)?([a-zA-Z_$])+([a-zA-Z0-9_$]+)?([\s]+)?=([\s]+)?\[([\s]+)?\"\\x[0-9a-fA-F]+/ nocase $string2 = /var([\s]+)?([a-zA-Z_$])+([a-zA-Z0-9_$]+)?([\s]+)?=([\s]+)?eval;/	any of them	10

[33] https://github.com/icoscx/WebsiteRaker/tree/master/yararules/Group_1

metarefresh : PB_Gen	$ = /(http-equiv='refresh')/ nocase $ = /(http-equiv="refresh")/ nocase	any of them	5
httpredirection : PB_Gen	$ = /(302 Found)/ nocase $ = /(304 Not Modified)/ nocase $ = /(307 Temporary Redirect)/ nocase $ = /(308 Permanent Redirect)/ nocase $ = /(301 Moved Permanently)/ nocase	any of them	3
httpredirec-tion_moved : PB_Gen	$1 = /(301 Moved Permanently)/ nocase	#1 > 3	15
mimeIsExe : PB_Gen	$ = /(application\/x-msdownload)/ nocase $ = /(application\/x-msdownload)/ nocase $ = /(application\/octet-stream)/ nocase	any of them	50

Group 2[34]:

dataUri : PB_Gen	$ = /(<object data="data:)/ nocase $ = /(<Anchor data="data:)/ nocase $ = /(<IFRAME data="data:)/ nocase $ = /(<Image data="data:)/ nocase $ = /(<EMBED SRC="data:)/ nocase	any of them	15
urlisip : PB_Gen	$1 = /((https|http):\/\/(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5]))/ nocase	$1	17
urlhassexe : PB_Gen	$1 = /(\.exe)/ nocase $2 = /(\.bin)/ nocase	$1 or $2	33
shortURL : PB_Gen	$ = /(bit\.ly\/)/ nocase $ = /(goo\.gl\/)/ nocase $ = /(tinyurl\.com\/)/ nocase (*Rest of 230 short urls are in github*)	any of them	25
bad_tld : PB_Gen	$1 = /(\.su\/|\.top\/|\.link\/|\.pw\/|\.xyz\/| \.ga\/|\.gq\/|\.tk\/|\.ru\/|\.bid\/|\.win\/| \.club\/|\.trade\/|\.info\/|\.biz\/|\.cc\/|\.qa\/| \.ws\/)/ nocase	$1	15

[34] https://github.com/icoscx/WebsiteRaker/tree/master/yararules/Group_2

Group 3[35]: Elements found after script execution (sandbox does not always manage to finish the script iteration)

abusedfunctions : PB_Gen	$1 = /(unescape\()/ nocase $2 = /(decompress\()/ nocase $3 = /(Uint8Array\()/ nocase $4 = /(inflate\()/ nocase $5 = /(setTimeout\()/ nocase $6 = /(Math\.pow)/ nocase $7 = /(Math\.round)/ nocase $8 = /(deflate\()/ nocase	#1 > 30 or #2 > 2 or #3 > 2 or #4 > 1 or #5 > 15 or #6 > 30 or #7 > 30 or #8 > 1	10
createhtmltags : PB_Gen	$1 = /(createElement\('script'\))/ nocase $2 = /(createElement\('iframe'\))/ nocase	$1 or $2	3
createhtmltags_many : PB_Gen	$1 = /(createElement\('script'\))/ nocase $2 = /(createElement\('iframe'\))/ nocase	#1 > 30 or #2 > 10	12
navigation : PB_Gen	$1 = /(window\.location)/ nocase $2 = /(window\.navigate)/ nocase	#1 > 2 or #2 > 2	10
InnerHTML : PB_Gen	$inner = /(innerhtml)/ nocase	#inner > 150	15
documentWrite : PB_Gen	$1 = /(document[\d].write)/ nocase $2 = /(document[\d\d].write)/ nocase	#1 > 15 or #2 > 30	15

[35] https://github.com/icoscx/WebsiteRaker/blob/master/yararules/Group_3/DynamicDOM.yar

V. Playbook WScript

Goup 3: PB_Dyn == WScript

Name	Match against	Condition	Score
wscript : PB_Dyn	$1 = /(ExpandEnvironmentStrings)/ nocase $2 = /(WScript\.Shell)/ nocase $3 = /(TEMP)/ nocase $4 = /(MSXML2\.XMLHTTP)/ nocase $6 = /(MSXML2.XMLHTTP.[\d\d].\.open)/ nocase $7 = /(responsebody\.get\()/ nocase $8 = /(ADODB\.Stream)/ nocase $9 = /(HTTP\/404)/ nocase $10 = /(LoadFromFile\()/ nocase $11 = /(MathRandom\()/ nocase	5 of them	40
wscriptYear : PB_Dyn	$1 = /(getyear)/ nocase $2 = /(geatyear\()/ nocase $3 = /(date\()/ nocase $4 = /(@cc_on)/ nocase	($1 or $2 or $3) and $4	15
wscriptMore : PB_Dyn	$1 = /(Write\()/ nocase $2 = /(WriteAll\()/ nocase $3 = /(run\()/ nocase $4 = /(\.SaveToFile\()/ nocase $5 = /(GetSecialFolder\([\d]\))/ nocase $6 = /(open\()/ nocase $7 = /(\.exe)/ nocase	3 of them	40
wscriptEvenMore : PB_Dyn	$1 = /(WScript\.Sleep\()/ nocase $2 = /(FileExists\()/ nocase $3 = /(send\()/ nocase $4 = /(WScript\.Shell.Environment\()/ nocase $5 = /(\.dll|\.exe)/ nocase	2 of them	40
wscriptMoreThree : PB_Dyn	$1 = /(WScript\.CreateObject\()/ nocase $2 = /(WScript\.Shell)/ nocase $3 = /(cmd\.exe)/ nocase	$1 and $2 and $3	50

VI. Navigation test script results

Successful cookies management, referrer setting and RFC compliance

```
[Date=Thu, 16 May 2019 13:12:32 GMT, Server=Apache/2.4.29 (Ubuntu), Loca-
tion=http://localhost/2.php, Content-Length=179, Keep-Alive=timeout=5, max=100,
Connection=Keep-Alive, Content-Type=text/html; charset=UTF-8]
301 Moved Permanently
CurrentURL: http://localhost/1.php
Referer: {Accept=text/html, application/xhtml+xml, */*, Accept-Encoding=gzip,
deflate, Accept-Language=en-US}
<!DOCTYPE html>
<head><title>Test 301</title></head>
<body>
<script>//must be ignored because of header location
window.location="http://localhost/3.php?id=2"; </script>
</body>

[Date=Thu, 16 May 2019 13:12:32 GMT, Server=Apache/2.4.29 (Ubuntu), Vary=Accept-
Encoding, Content-Encoding=gzip, Content-Length=190, Keep-Alive=timeout=5,
max=99, Connection=Keep-Alive, Content-Type=text/html; charset=UTF-8]
200 OK
CurrentURL: http://localhost/2.php
Referer: {Accept=text/html, application/xhtml+xml, */*, Accept-Encoding=gzip,
deflate, Accept-Language=en-US}
<!DOCTYPE html>
<head><title>TestMetaRefresh</title></head>
referer
<meta http-equiv='refresh' content='2;url=http://lo-
calhost/3.php?id=2'><script>//must be ignored because of meta refresh
window.location="http://localhost/3.php?id=2"; </script>

[Date=Thu, 16 May 2019 13:12:32 GMT, Server=Apache/2.4.29 (Ubuntu), Set-
Cookie=Victim=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0; path=/,
Set-Cookie=pathThestS=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0;
path=/3.php?id=2, Vary=Accept-Encoding, Content-Encoding=gzip, Content-
Length=221, Keep-Alive=timeout=5, max=98, Connection=Keep-Alive, Content-
Type=text/html; charset=UTF-8]
200 OK
CurrentURL: http://localhost/3.php?id=2
Referer: {Accept=*/*, Accept-Encoding=gzip, deflate, Accept-Language=en-US}
<!DOCTYPE html>
<head><title>test Iframe and Cookies</title>
</head>
<body>
<script type="text/javascript"> var a = 2; var b = 3; var c = eval("a*b");
document.write(c);document.write("http://localhost/index.php"); </script>
<iframe id="secret" src="http://localhost/4.php?l"></iframe>
<div id="first

[Date=Thu, 16 May 2019 13:12:32 GMT, Server=Apache/2.4.29 (Ubuntu), Vary=Accept-
Encoding, Content-Encoding=gzip, Content-Length=178, Keep-Alive=timeout=5,
max=97, Connection=Keep-Alive, Content-Type=text/html; charset=UTF-8]
200 OK
CurrentURL: http://localhost/4.php?l
Referer: {Accept=*/*, Referer=http://localhost/3.php?id=2, Accept-Encoding=gzip,
deflate, Accept-Language=en-US}
<!DOCTYPE html>
<head><title>Inside Frame</title></head>
<body>
referer
http://localhost/3.php?id=2<script>document.write("<p>para</p>");</script>
<div id="4withid"> 4 - withid</div>
</body>
```

VII. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Ivo Pure**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

An automated methodology for validating web related cyber threat intelligence by im-plementing a honeyclient,

supervised by **Risto Vaarandi**, **Raimundas Matulevicius**.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

IVO PURE

16/05/2019