Applying Machine Learning for Detecting Exploit Kit Traffic

Jonas Sandnes
Master’s Thesis Autumn 2017
Applying Machine Learning for Detecting Exploit Kit Traffic

Jonas Sandnes
Abstract

Detection of malicious traffic can be challenging. Network intrusion detection systems are one of the tools that can be used in order to detect traffic, but many IDSs rely on static signatures that can be avoided relatively easy by attackers.

In this thesis we explore the potential of using machine learning in order to classify exploit kit traffic. We have created a machine learning system that aims to detect exploit kit traffic based on analyzing HTTP logs only. By using existing machine learning algorithms we have investigated the performance of such a system and developed it in a lightweight manner to be potentially used as a complement to a traditional IDS. The system shows relatively good performance in detecting exploit kit traffic considering that it only uses features extracted from HTTP traffic logs. However, the system in its current state does not perform well enough to be directly applied in a realistic setting. In order to improve the performance it seems necessary to include features from additional sources that do not require too much processing power such as passive DNS. The thesis concludes that the system have potential to become a good complement to an IDS, but it requires additional improvements to be used in a network deployment.
Acknowledgements

I would like to thank my supervisors Dr. Martin Eian at mnemonic and Professor Audun Jøsang at the University of Oslo for giving me valuable feedback and guidance through my thesis work. The help has been invaluable and has made me think of ideas to solve the thesis problems that I would not have managed alone.

I would also like to thank my peer students and friends who have supported me during my thesis work and studies.

Thanks to my colleagues at mnemonic. mnemonic has provided me with an exciting work environment with high competence where information security is in focus and I am grateful I can work with a field of study that I have a great passion for.

I would also like to thank my mother for always supporting and encouraging me.
# Table of Contents

1  Introduction .................................................................................................................. 1  
   1.1 Research method ........................................................................................................ 2  
   1.2 Research questions .................................................................................................... 4  
   1.3 Related work .............................................................................................................. 5  
   1.4 Chapter overview ...................................................................................................... 7  
2  Exploit kits .................................................................................................................... 9  
   2.1 Background ............................................................................................................... 9  
   2.2 Installing the exploit kit ............................................................................................ 10  
   2.3 Redirecting victim ...................................................................................................... 10  
   2.4 Landing page ............................................................................................................ 12  
   2.5 Exploit page ............................................................................................................. 13  
   2.6 Payload ................................................................................................................... 13  
   2.7 Vulnerabilities .......................................................................................................... 15  
      2.7.1 CVE and CVSS .................................................................................................... 15  
      2.7.2 Targeted software .............................................................................................. 16  
   2.8 Types of malware ..................................................................................................... 17  
      2.8.1 Ransomware ....................................................................................................... 18  
      2.8.2 Crimeware ......................................................................................................... 18  
      2.8.3 Clickfraud .......................................................................................................... 18  
   2.9 Countermeasures ...................................................................................................... 18  
      2.9.1 Keeping software at a minimum .......................................................................... 19  
      2.9.2 Patching ............................................................................................................ 19  
      2.9.3 Security protection ............................................................................................ 19  
3  Concepts of machine learning ...................................................................................... 21  
   3.1 Background ............................................................................................................... 21  
   3.2 Supervised learning .................................................................................................. 21  
   3.3 Classification ............................................................................................................ 22  
      3.3.1 Picking the right features .................................................................................... 23  
      3.3.2 Training the classifier ......................................................................................... 23  
   3.4 Unsupervised learning ............................................................................................... 26  
   3.5 Reinforcement learning ............................................................................................. 26
List of Figures

Figure 1: Research method (Adapted from [1]) ................................................................. 2
Figure 2: Example of iframe usage redirecting to an exploit kit (Adapted from [12]) .......... 11
Figure 3: Exploit kit chain .................................................................................................. 14
Figure 4: Distribution of vulnerabilities by product (Data adapted from CVSS [22]) .......... 16
Figure 5: Victim redirected towards an exploit kit server ................................................... 20
Figure 6: Machine learning phases ..................................................................................... 26
Figure 7: Flowchart of the system ....................................................................................... 28
Figure 8: Illustration of fitting a machine learning model .................................................... 34
Figure 9: Example of underfitting and overfitting ................................................................. 35
Figure 10: Feature importance ranking ................................................................................ 59
List of Tables

Table 1: CVSS severity scale .................................................................................................................. 15
Table 2: Top 10 exploited vulnerabilities in 2016 (Adapted from [23]) .............................................. 17
Table 3: K-fold cross-validation ............................................................................................................. 38
Table 4: Converting categorical to nominal .......................................................................................... 39
Table 5: One hot encoded values ......................................................................................................... 40
Table 6: Pros and Cons of automatic extraction ..................................................................................... 42
Table 7: Pros and Cons of manual extraction .......................................................................................... 43
Table 8: Data sets overview .................................................................................................................... 44
Table 9: Exploit kit data set overview .................................................................................................... 44
Table 10: Selected features .................................................................................................................... 48
Table 11: Binary classification results: Support Vector Machine .......................................................... 49
Table 12: Binary classification results: Random Forest Classifier ....................................................... 50
Table 13: Binary classification results: Multinomial Naïve Bayes ....................................................... 50
Table 14: Binary classification results: model performance comparison ........................................... 51
Table 15: Multi-class classification results: Support Vector Machine ................................................ 52
Table 16: Multi-class classification results: Random Forest Classifier .............................................. 52
Table 17: Multi-class classification results: Multinomial Naïve Bayes ................................................. 53
Table 18: Multi-class classification: model performance comparison .................................................. 53
Table 19: Testing results ........................................................................................................................ 55
Table 20: Feature importance ranking .................................................................................................. 58


1 Introduction

Communication technology has over the last decades changed the daily life of many people across the globe. Internet has contributed in changing the ways we can communicate with each other and our daily habits involving interaction with digital technology. Many modern businesses companies rely and base their business model on the Internet to provide services. Individuals and entities use these services and may provide sensitive data while assuming that the communication is secure. The increasing usage of the Internet is in many ways positive. However, with the growing dependability comes the concern for security. If a business experiences downtime in essential online services or an individual becomes victim of identity theft then the consequences could be very serious. The development in the last decades have resulted in a new type of crime known as cybercrime. Cybercriminals continuously develop new methods to breach computers and servers for their own financial gain.

A major method of spreading malicious software (malware) are exploit kits. Exploit kits are automated toolkits which are executed in a chain of events when a user visits a malicious or hacked website. They are developed to target vulnerabilities in the web browser or the browser’s plugins in order to install malware. If a computer is vulnerable then the malware can be installed unknowingly by the user. As a result of the malware, many users become victims of identity theft or have their personal files compromised.

Identifying compromised websites and exploit kits can be challenging. Modern exploit kits are developed so the HTTP traffic can be mistaken as normal traffic. In addition, patterns used by an exploit kit can be changed regularly to avoid detection by signatures or regular expressions.

The motivation for our thesis is to apply machine learning in order to detect exploit kit traffic. We will explore the possibility of developing a lightweight system to be compatible with traditional intrusion detection systems. Based on existing machine learning algorithms we will investigate the performance of classification when it is limited to only HTTP log data.
1.1 Research method

For this thesis we have followed a specific design research method [1]. The method is structured into the following stages:

- Awareness stage
- Suggestion stage
- Development stage
- Evaluation stage
- Conclusion stage

The research method is illustrated below:

![Figure 1: Research method (Adapted from [1])](image-url)
In the awareness stage we address the research questions. We then proceed to a suggestion stage where we design possible solutions to the research questions in the previous stage. The suggestion stage is followed by a development stage where a possible solution is implemented. Next, we proceed to the evaluation stage where we evaluate the implementation on whether the results are satisfactory. If the results are not satisfactory enough then we can return to previous stages.

The strength of the research method we have chosen in this thesis is that it is designed to be flexible and adaptable to changes. If a stage happens not to be as expected or if a new idea to the research problem emerges then we should be able to go back to a previous stage to reevaluate our design.

Our awareness approach to this thesis is that we will cover two different fields of study. We will first start by introducing exploit kits and proceed with a separate chapter to introduce relevant concepts of machine learning. The goal is to combine the two fields of study and use what we have learned to create a machine learning system that focus to detect exploit kit activity.
1.2 Research questions

Exploit kits are a threat to all users browsing Internet websites. In case of a compromised website, the attacker can redirect traffic from the website towards a website hosting a malicious exploit kit. This traffic is not easily identified as malicious and without monitoring the traffic then it can go unnoticed.

In this thesis we want to investigate an approach to apply machine learning on Hypertext Transfer Protocol (HTTP) in order to detect exploit kit activity. The system will be from an intrusion detection system’s (IDS) point of view where it stores logs continuously as it is monitoring network traffic. We want the system to be lightweight in order to provide high performance. The data set used to train the machine learning models in the thesis will be limited to what is available to a traditional IDS and we will not use any content-based data extracted by crawling the websites. We will use real network traffic data gathered from IDSs in order to train our machine learning system.

Based on our goals we described above we formulated the following research questions:

Q1) What are the most relevant traits of recent modern exploit kits?

Q2) How can we design a lightweight machine learning system to process, analyze and classify malicious exploit kit activity?

Q3) What accuracy of classification can we expect of the machine learning system with the data we have available?

Q4) How effective would the machine learning system perform if it were to be integrated with an IDS?
1.3 Related work

Detecting exploit kit traffic or malicious traffic in general has previously been researched and discussed in multiple research publications. In recent years open source machine learning tools and frameworks have made this research field more available to everyone.

In this section we want to highlight some of the work of other security researchers which we have studied through our own research for this thesis.

WebWinnow

WebWinnow is a system which aims to detect whether a URL hosts an exploit kit [2]. The features used in the system are based on an extensive analysis of 40 different exploit kits. In addition to applying machine learning techniques, the system is also built to identify attack-centric and self-defense behaviorism that was found in the analysis of the exploit kits in lab-environments and in live-analysis. Attack centric behaviors includes obfuscation, redirection and fingerprinting. The self-defense behaviors with the purpose to evade detection techniques includes IP blocking, cloaking and signature evasion. The system showed great results applying different algorithms as for example Bayesian networks, logistic regression and random forest. The false positive rates in the testing phase were as low as 0.001%, while the training phase showed up to 100% true positive rates.

While WebWinnow has similarities with what we want to achieve in our thesis, there are some strict differences. WebWinnow is content based and involves content-specific features which we do not include in our thesis. The interest and research problem we have in this thesis is based on an IDS point of view. We will not focus on, for example, having a content-crawler, which will inspect sites in order to extract features and determine exploit kit content and behavior, but we will try to see what can be achieved by only using logs which are made available on the IDS.
JSAND

Cova et al. [3] propose a system which aims to detect malicious JavaScript code. The JSAND (JavaScript Anomaly-based aNalysis and Detection) system uses features and machine learning techniques in order to establish characteristics of normal JavaScript code. By establishing the characteristics of normal code then the system may find anomalous code that may be malicious. Web pages are visited using an instrumented browser, so that events which occur can be recorded. The JSAND system was tested by visiting over 140,000 web pages and showed low false positive rates. The presented system is also able to analyze obfuscated code and generate signatures which can be used for network threat detection.

The work presented focuses on the exploitation phase in the exploit kit chain. In our thesis we focus on the full chain of the exploit kit and a more lexical based approach. By visiting web pages and recording events that occurs it can bring a lot of valuable features for detecting malicious behavior. However, in our thesis we want to restrict the analysis from an IDS point of view. Depending on the amount of activity on a network, an IDS may receive thousands or millions of lines of HTTP requests every minute. It would take too much resources to visit all the unique web pages which are observed and we will therefore limit our work to try use a machine learning system which can be used with relatively few resources.

Anatomy of drive-by download attack

Le et al. [4] propose a framework for analysis which focuses on potential state changes observed when a web browser render a HTML document. The research presents a detailed description of the different stages in the exploit kit chain and suggests features which can be used for analysis for each of the different stages of the exploit kit.

The research focuses on the different stages similar to those we present it in our thesis. In addition, it includes certain features we wish to include in our machine learning system. However, in our thesis we will not focus on content-based features. In order to retrieve content-based features we need a lot of resources to retrieve the content by, for example, using a HTML crawler. The goal in our thesis is to create a lightweight analysis system that can process data at a high speed.
1.4 Chapter overview

Our thesis is divided into 7 different chapters.

- **Chapter 2: Exploit kits**
  
The chapter investigates different topics of exploit kits and how it is structured. Also, it will cover how attackers operate with the kits and suggest countermeasures to minimize the risk of becoming a victim.

- **Chapter 3: Machine learning**
  
The chapter describes several machine learning approaches, and the different stages of building and validating a machine learning model.

- **Chapter 4: Technical**
  
The chapter describes the technical details of designing the machine learning system. It will also present the chosen machine learning algorithms and the evaluation techniques that has been applied.

- **Chapter 5: Results**
  
The chapter describes experienced challenges during development and performance results of the machine learning system.

- **Chapter 6: Discussion**
  
The chapter discusses the performance of the machine learning system and the possible impact it would have in a continuous real-traffic environment.

- **Chapter 7: Conclusion**
  
The chapter summarize the thesis and will present a conclusion to the research problem. Suggestions for future work is presented.
2 Exploit kits

2.1 Background

Malware distribution is increasing and cybercriminals try to find new ways to spread their malware. One of the more popular methods of serving malware in recent years has been the use of drive-by downloads. Drive-by downloads happen in the event of a victim visiting a compromised website or clicking a link which redirects to a malicious server. The malicious server will attempt to scan the victim’s client software for vulnerabilities. If a vulnerability is found then it will try to execute an exploit in order to serve a malicious payload. All the events might be performed unnoticed and can result in an infected client without the victim’s knowledge. The owner of the malicious server manages to do this with the help of a toolkit that we refer to as an exploit kit.

Exploit kits are not a new method of serving malware. The first known exploit kit was observed in 2006 by the name WebAttacker [11]. The kit was developed to find vulnerabilities to exploit and was sold in a Russian underground market for $20. Later the same year, a second exploit kit named Mpack was released. It had the same functionality as WebAttacker, but offered statistical analysis and was sold for $1000 [11]. A few years down the road, the market for exploit kits has increased significantly. Modern exploit kits are dangerous tools in the cybercriminals’ arsenal because of the flexible nature of the kits. Many of the exploit kits are being distributed as Software as a Service (SaaS) or as a more recent cybersecurity-definition: Malware as a Service (MaaS). The authors may rent or sell their malicious toolkits to buyers. The leading exploit kits are available in cybercriminal markets as a subscription-based service for thousands of dollars per month [9][10].

Exploit kits are constructed as a chain of actions in order to successfully deliver malware to a victim. The exploit kit chain consists of four main steps:

- Redirect
- Landing page
- Exploit page
- Payload
In order to understand how the exploit kits works in practice we will introduce an overview of how it operates and the conditions needed for a successful chain infection.

### 2.2 Installing the exploit kit

Exploit kits are, as previously mentioned, sold in cybercrime markets and many of the kits are developed in a way so they are easy to install and use for any cybercriminal. However, some exploit kits have previously been leaked and uploaded so they can be attainable for free, such as the RIG exploit kit which was leaked in February 2015 [10]. Many exploit kits are easily configured and maintained with a graphical user interface (GUI). Through the GUI the attacker is presented with statistics on how many victims that have accessed the malicious server and how many exploit attempts were successful. This gives the attacker an analysis of how well the exploit kit is performing. In addition, the GUI is built to make it easy to manage the payload to be served by successful exploits [9][10]. The user-friendliness adds to the fact that the authors have created the exploit kits by distributing it as a SaaS. The available software exploits which come with the kit are often stored on an external server which the exploit kit author manage and updates regularly as part of the subscription-based service [9][10].

### 2.3 Redirecting victim

**Pre-condition:** Victim accessing compromised website.

**Post-condition:** Redirect victim.

In order to get traffic on the malicious server the attacker needs to find ways to redirect victims. This can be done in multiple ways. The most common way is by compromising legitimate websites and then proceed to redirect the user traffic towards the malicious server. In order to compromise websites the attacker will take advantage of potential vulnerabilities and inject malicious code into the code of the website. A website can either be specifically targeted or found with the use of probe-scanning which will automatically scan many websites on the Internet for vulnerabilities. Many websites use Content Management Systems (CMS) such as WordPress, Drupal and Joomla that are regularly updated for security vulnerabilities [17]. However, in many cases these CMS can be used in combination with third-party plugins that might be outdated and contain vulnerabilities. These plugins can be targeted by attackers as they are often not maintained by the website owner.
The injected code on a compromised website can be in the form of a simple redirect or heavily obfuscated code which makes it hard to analyze [12]. Another method is the use of an inline frame (iframe) in Hypertext Markup Language (HTML). Iframe is a HTML element that can be used in order to reuse code and load another HTML document into another frame. This can be abused by attackers using the iframe to load the exploit kit when accessing the compromised website [12][20]. Attackers may also try to obfuscate the injected code in different ways. Security measures and network detection technologies uses signatures in order to try detect possible malicious iframes and other redirects. Malicious redirects and security vendor signatures follow a pattern which makes them static so it becomes possible to avoid them in various ways. As an example, some observed versions of the exploit kit Magnitude disguised their iframe as a PNG-image by encoding values into the color of the image’s pixels [10]. Other methods which have been reported is checking the presence of security software on the victim machine before executing the exploit kit [11][13]. In addition, some exploit kits have been observed using gates as a step between the compromised website and the exploit kit server [12]. By using a gate as a proxy it can exclude some incoming redirects by checking the victim for their geolocation based on their IP or checking their user agent for what operating system they are running. If the victim does not meet the desired criteria then the attacker might redirect from the gate to a fake website.

Figure 2: Example of iframe usage redirecting to an exploit kit (Adapted from [12])
Another method of redirecting traffic is the use of malvertising. The word malvertising is a combination of ‘malware’ and ‘advertising’. Many websites use third-party advertisement companies in order to generate income through ads. Even though the use of ads on websites is very common today the issue is that the domain owner does usually not have control over who will place ads on their website. However, attackers cannot simply just buy an ad for a high profile website to generate traffic. This can be done by gaining access to a trusted advertiser which has been compromised by an attacker. These compromised advertisers have previously been sold in underground markets [10]. Another method is by acting as a legitimate advertiser over time to gain reputation. After gaining enough reputation in the advertisement network then the attacker can place their ads on high profile pages to generate large volume of traffic.

## 2.4 Landing page

**Pre-condition:** Victim redirected to exploit kit

**Post-condition:** Identified software client which might be vulnerable. Redirect to exploit page.

The first step in the exploit kit is the landing page. The landing page will scan the victim’s machine for what browser the victim is using and for additional plugins which might be installed. By scanning the victim then the attacker may choose to use the exploit that will more likely result in a successful payload download. The scanning activity will usually go unnoticed of the victim because the attacker can hide the connection to the malicious exploit kit server, for example, with the use of hidden iframes. As an example, the Angler exploit kit has previously been observed mixing obfuscated code in-between random verses of literature [10]. This will make static analysis of network traffic more challenging since security products often relies on signatures to detect malicious code. At the same time, the exploit kit author will have an algorithm at hand which makes it easy to decode the obfuscated code.
2.5 Exploit page

**Pre-condition:** One or more possible vulnerable software entities have been identified

**Post-condition:** A software vulnerability was successfully exploited.

The exploit page is the third step in the exploit kit chain. After profiling the victim for software in the landing page then the exploit kit will receive a relevant exploit from a remote server managed by the author of the exploit kit. It is to be believed that some authors operate with different rates of cost for different exploits. For example, zero-day vulnerabilities have a higher rate of successful exploitation to deliver a payload than a vulnerability which has been patched in a security update by their respective software developer. The older and more known the vulnerability, the more likely it is to be patched so it can not be exploited. The reason why zero-day vulnerabilities are very dangerous is because the software which is vulnerable does not have a security update available yet. Zero-day vulnerabilities might unnoticed without proper network and forensic analysis. On the other hand, it is also known that exploit kit authors are up to date with released open source vulnerabilities. Recently a researcher published a vulnerability for Flash which only a few days later was observed being used in the Neutrino exploit kit [16]. There are also other incidents which are similar. In July 2015, the compromise of the Hacking Team group resulted in the leak of a zero-day vulnerability for Flash [15]. This was shortly after used in the wild by the Angler exploit kit. As a result of this the amount of requests associated with Angler exploit kit was greatly increased [14].

2.6 Payload

**Pre-condition:** Successfully exploited a software vulnerability to deliver the payload.

**Post-condition:** Successfully delivered the payload to victim.

The payload is the fourth and last step in the exploit kit chain. This step will follow a successful exploit of a vulnerability in the victim’s client and grants the attacker the opportunity to make the victim download a file of the attacker’s choice. If the user has an anti-virus software installed on their client then the payload might be blocked or quarantined. Many exploit kits provide a GUI which makes it easy for the attacker to decide what file to be delivered as a payload. The attacker will more likely want to have a payload that will go undetected past potential security
protection software. Some exploit kits provide a user-friendly anti-virus service check before uploading a payload [10]. The check is done by sending the binary of the file to a subscription based malware scanning service which tests the file against multiple anti-virus engines. If the chosen payload is flagged as malicious by multiple anti-virus vendors then the attacker might want to choose another file that goes undetected. The payload step finishes the exploit kit chain and is considered successful if the payload was installed on the victim’s client.

The following figure illustrates the exploit kit chain as described in the previous sections:

![Figure 3: Exploit kit chain](image-url)
2.7 Vulnerabilities

2.7.1 CVE and CVSS

Common Vulnerabilities and Exposure (CVE) is a system which provides a dictionary to publicly known security vulnerabilities [21]. Each vulnerability listed is assigned a common name which is also known as a CVE identifier. CVE was launched in 1999 as a result of a demand for a common reference system to known vulnerabilities. All vendors and information security tools used their own names for vulnerabilities and it was not always clear whether information sources referred to the same vulnerability or two different ones. Today, CVE is adopted as the standard for vulnerability and exposure referencing. The CVE format is given by current year and arbitrary numbers.

Common Vulnerability Scoring System (CVSS) is an open framework for measuring the severity and impact of a vulnerability [22]. As with CVE, the CVSS framework was developed due to a missing universal framework to determine the severity of known vulnerabilities. For each vulnerability a CVSS severity score is assigned to it and the score can be used to determine the potential impact if it was to be exploited. The severity makes it easier for an individual or an organization to determine whether they should take countermeasures.

CVSS are ranked in the following severity rating scale:

<table>
<thead>
<tr>
<th>Severity</th>
<th>CVSS Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>0.0 – 3.9</td>
</tr>
<tr>
<td>Medium</td>
<td>4.0 – 6.9</td>
</tr>
<tr>
<td>High</td>
<td>7.0 – 8.9</td>
</tr>
<tr>
<td>Critical</td>
<td>9.0 – 10.0</td>
</tr>
</tbody>
</table>

*Table 1: CVSS severity scale*

Common for both CVE and CVSS is that they were developed in order to have a universal framework to address known vulnerabilities and severities. By using them in combination they can be a great reference tool for anyone involved with software maintenance or security. On the other hand, some vulnerabilities can be undisclosed and used by attackers. This is known as zero-day vulnerabilities. The name describes a vulnerability for which the software vendor
has not yet issued a patch it or released an advisory to the public on how to take countermeasures on the exploit.

### 2.7.2 Targeted software

Exploit kits target popular software plugins and web browsers in order to find vulnerabilities and deliver their malicious payload. The following web browsers and plugins have been heavily targeted by exploit kits in recent years:

- Adobe Flash Player
- Java
- Microsoft Internet Explorer
- Microsoft Silverlight

The reason why the listed software products have been targeted is not coincidental. They are popular software used heavily worldwide for web browsing, media, games and bank applications. If we take a closer look at how many vulnerabilities in total that have been discovered and given a CVE for each of the software in the last 5 years we get the following graph:

![Figure 4: Distribution of vulnerabilities by product (Data adapted from CVSS [22])](image-url)
The following vulnerabilities are identified as the most used by exploit kits in 2016 [23].

<table>
<thead>
<tr>
<th>CVE ID</th>
<th>CVSS Score</th>
<th>Product</th>
<th>Exploit kit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2016-0189</td>
<td>7.6</td>
<td>Microsoft Internet Explorer</td>
<td>Magnitude, Neutrino, RIG, Sundown</td>
</tr>
<tr>
<td>CVE-2016-1019</td>
<td>10.0</td>
<td>Adobe Flash Player</td>
<td>Magnitude, Neutrino, Nuclear Pack</td>
</tr>
<tr>
<td>CVE-2016-4117</td>
<td>10.0</td>
<td>Adobe Flash Player</td>
<td>Angler, Magnitude, Neutrino, RIG</td>
</tr>
<tr>
<td>CVE-2015-8651</td>
<td>9.3</td>
<td>Adobe Flash Player</td>
<td>Angler, Hunter, RIG, Sundown</td>
</tr>
<tr>
<td>CVE-2016-0034</td>
<td>9.3</td>
<td>Microsoft Silverlight</td>
<td>Angler, RIG, Sundown, Hunter</td>
</tr>
<tr>
<td>CVE-2016-1010</td>
<td>10.0</td>
<td>Adobe Flash Player</td>
<td>Angler</td>
</tr>
<tr>
<td>CVE-2014-4113</td>
<td>7.2</td>
<td>Microsoft Windows</td>
<td>Nuclear Pack</td>
</tr>
<tr>
<td>CVE-2015-8446</td>
<td>9.3</td>
<td>Adobe Flash Player</td>
<td>Angler</td>
</tr>
<tr>
<td>CVE-2016-3298</td>
<td>2.6</td>
<td>Microsoft Internet Explorer</td>
<td>Neutrino</td>
</tr>
<tr>
<td>CVE-2015-7645</td>
<td>9.3</td>
<td>Adobe Flash Player</td>
<td>Angler, Hunter, Magnitude, Neutrino, Nuclear Pack, RIG, Spartan</td>
</tr>
</tbody>
</table>

Table 2: Top 10 exploited vulnerabilities in 2016 (Adapted from [23])

Adobe Flash Player vulnerabilities are greatly represented in the top 10 list and it has, together with Internet Explorer, been the most targeted software products in recent years. Originally, attackers targeted Java, but after the owner company Oracle made changes to the software to block unsigned or self-signed applets then the attackers changed their primary target [28][29].

### 2.8 Types of malware

All types of malware can be delivered through exploit kits and it is up to the attacker behind the exploit kit server to decide which malware that is to be delivered in the payload. In this section we will present three common malware types that are associated with the exploit kit payload.
2.8.1 Ransomware

Ransomware is a type of malware that will block the victim from gaining access to some or all of their data. The victim will find their screen frozen with a message or having their files locked with encryption. The attacker will ask for a ransom in order to decrypt the files to their normal condition. It is most common for ransomware authors to request ransom payment through cryptocurrencies such as Bitcoin that offers anonymity in the transaction, but alternative options of payment such as gift cards are also commonly observed. Ransomware has increased in recent years and Cisco Talos reported in 2015 that over 60% of Angler exploit kit payloads were a variant of ransomware [14].

2.8.2 Crimeware

Crimeware, also known as banking Trojans, is a type of malware that aims to gather sensitive data for financial gains. There are different approaches to how they behave. For example, they might log all keystrokes the victim inputs on their infected client or capture screenshots at certain time intervals or when accessing specific websites. The stolen data is then sent back to the attacker and can be abused or sold in underground markets.

2.8.3 Clickfraud

Clickfraud refers to the action of abusing pay-per-click (PPC) online advertisements. Advertisers using advertising-services with PPC will pay every time an advertisement of theirs is clicked on a website. The attacker will use the infected clients to its disposal to generate clicks on ads which will generate revenue. Clickfraud may be performed by running automated scripts on websites created solely to abuse online advertising companies’ revenue-model.

2.9 Countermeasures

This section describes some of the countermeasures an entity can take in order to minimize the risk of becoming a victim of an exploit kit.
2.9.1 Keeping software at a minimum

The more software and plugins which are installed on a client, the more it is at risk of having vulnerabilities which can be exploited in the present or in the future as vulnerabilities are discovered. To reduce the risk it can be clever to have a policy to uninstall all unnecessary software and only keeping the software that is actually used on the client.

2.9.2 Patching

Exploit kits use vulnerabilities in browsers and plugins that are popular to use. There might be multiple reasons for this.

- Software contains vulnerabilities due to poor focus on security during development.
- Finding vulnerabilities is of high interest due to many people using the software.

One or a combination of both can possibly be the reason, but when vulnerabilities are discovered then it is common for the software developer to prioritize a security patch update. As an example, we previously mentioned the Hacking Team breach in 2015 which released two Flash vulnerabilities in the wild. After the vulnerabilities were released then exploits for the vulnerabilities were integrated in the Angler exploit kit within 24 hours after release [10]. The security update that patched the vulnerability was released a day after it was integrated in Angler and as a consequence it gave attackers from one to several days’ time to exploit any client with Flash installed.

All in all, exploit kits can target multiple vulnerabilities. By keeping software up to date it will reduce the risk of being exploited through vulnerabilities that have not been patched already. Keeping software up to date can be done either by checking for new updates routinely, having an update management software or, if available, using automatic updates for the software.

2.9.3 Security protection

We can divide security based protection into two categories:

**Network-based-protection** focuses on analyzing incoming and outgoing network traffic.

**Host-based-protection** focuses on analyzing behavior and files in the host machine.
Network based protection can be in form of an intrusion detection system (IDS) or intrusion preventive system (IPS). Whereas an IDS will look for security exploits and report it to the network owner, an IPS will include the same features as the IDS, but also be inline of the network connection and actively block suspicious or malicious threats. If the IDS or IPS is configured with exploit kit signatures then they might be able to detect or block if a client attempts to access an exploit kit. These exploit kit signatures can be based on previous observed exploit kit incidents and written in a regex pattern. However, the weakness in using this method is if the exploit kit author decides to change the pattern of, for example, the Uniform Resource Locator (URL) then the traffic might go unnoticed by the detection system.

Host based protection is installed on the client and can be in form a host based IDS or an anti-virus software. Anti-virus software will have access to the client’s directories in order to scan new or existing files. It will actively try to identify malicious files or behavior that will be blocked or quarantined.

All in all, the advices we have presented in this section will reduce the risk of becoming a victim of an exploit kit attack. However, even the most cautious user can become a victim due to zero-day vulnerabilities or simply by accessing assumed safe high profile websites that have been compromised.

*Figure 5: Victim redirected towards an exploit kit server*
3 Concepts of machine learning

3.1 Background

Machine learning represents a technique for computers to enhance their ability to adapt to change by learning from previous experiences. It is one of the subfields of artificial intelligence (AI) which have had a high increase of progress in the last decades. Machine learning applies learning algorithms in order to create a machine learning model. The model is trained by providing an experience in form of an input data set that it will learn from. By providing training data, the model will be able to make predictions on new data without being explicitly programmed for it. Today we can find machine learning technology in many consumer products and professional tools. Some examples are online language translators, face recognition in digital cameras, search engines, e-mail spam filtering, automated medical diagnosis tools and robot navigation.

There are 3 major approaches in order to train a system for machine learning:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

3.2 Supervised learning

Supervised learning is done by giving the learning model input of labeled data. The labeled data consists of data samples which have been labeled in order to help guiding the model in what the desired output of a correlated input is. The labeled data is the main step in training a model on how to treat relevant data, but it is also a crucial step in order to have the model make the correct predictions later on. Since the labeled data is often partially or fully manually labeled then it is important for the labels to be correct as it will affect the performance of the system.
There are two different approaches which are usually associated with supervised learning.

- **Regression:** the goal of predicting a continuous value which correlates to the input. As an example there are many online movie streaming websites with functionality implemented where they can predict and present what kind of movies the user might like expressed as a continuous score. These recommendations are often based on what kind of movies the user has previously been watching or based on a rating system which gives the user a choice to input ratings for the movies based on their liking. If a user rates multiple movies in the genre «comedy» good, but has rated movies in the genre «horror» poorly, then the machine learning model will be more likely to recommend a comedy movie for the user.

- **Classification:** the goal of assigning a discrete class to new input data. One example is e-mail spam filtering where the two classes are specified as «spam» and «ham». The system will categorize the different data in to the two categories as a result of the labeled training set. As an example, if an e-mail contains the word «inheritance» and «millionaire» then these are feature values which might be associated with fraud e-mails and have been regularly present in the training set labeled for spam. If these words are observed in addition with other words which are known in spam e-mails, then the machine learning model might classify the e-mail to be spam. However, there might also be strong indicators which will weight an e-mail to be legitimate even though it contains typical spam words. For example, if the e-mail also contains a name of a legitimate person which is associated with the recipient of the e-mail then it may be considered as non-spam (ham).

The fundamentals of the training are equal for the two, but classification is more relevant for the desired outcome and work in this thesis. By using classification we want our machine learning system to be able to classify exploit kit data and non-exploit kit data.

### 3.3 Classification

Classification can be performed in different ways, but the most basic way is by binary classification. Binary classification is done by creating two classes which the data is to be categorized in. Classification is not a specific algorithm by itself, but a task to be solved with assistance of one or multiple algorithms.
3.3.1 Picking the right features

As we mentioned earlier we will use HTTP logs as the primary source for training the machine learning model. The HTTP logs will be preprocessed by extracting certain attributes which are representative for the classification and our research. These attributes are what we define as features. We will preprocess the HTTP logs and create feature sets which contains multiple nominal, categorical and binary values to be used to train and test the machine learning system. Choosing relevant features is crucial to have the machine learning system classify correctly in terms of the classes we have specified. If we were to choose many irrelevant features then we can risk having noise in the classification model and the performance can become poor.

Selecting good features can be done by picking a subset of the features and test them accordingly. One approach to test the features are wrappers [6]. The wrappers are used with the classifier to rate the feature subsets with their classification performance. After testing each of the subsets then it will be able to find which features scores best in correlation with the classifier. However, this approach can require a high amount of computational resources as it will try all combinations of feature subsets. Another alternative is by trial-and-error to see how well the model perform with different feature sets. This alternative is quicker and do not require as much performance and time. It can be better to use this alternative if one would like to be flexible in adding new features to test their impact. We will later in this thesis describe which features we have chosen and the importance of each feature in the performance results.

3.3.2 Training the classifier

We can divide the process of implementing classification into 3 different phases:

- Training
- Validation
- Testing

**Training**

Training is the first phase in developing the classifier of a machine learning model. As previously mentioned, the supervised learning approach embraces labeled data in order to train a model. Labelling a data set can be done in different ways, but it needs to be precise since the
labels will have an impact on the classification of the machine learning system. The data set should contain features that are relevant to the classification and in addition have a separate label set where each entry corresponds to the equal index of the data set. By providing relevant training data to a model, then the goal will be for the model to learn how to use the training data to classify unlabeled data it will encounter in the future.

The required size of the training data set depends on the quality of the data set as a whole. Each sample in a small data set needs to be of high quality and representative for the classification that is to be done. However, a risk of having few samples is that there can be a certain bias. To avoid this we can simply add more samples, but this is sometimes easier said than done. Data for one or more classes in the machine learning classifier can be easily obtainable, but others can be restricted in terms of availability or existence. There are solutions to this by weighting one class’ samples higher than another class, or by having duplicates of each sample of an underrepresented class to balance the data set. However, the risk of doing this is that it requires the data of the underrepresented class to be of a very high quality. The reason is that any representation of noise will be weighted more highly and this can potentially have a negative impact on the classifier. If we have a large training set then we minimize this risk as any noise will be nullified by the other samples. On a general basis, the size of the training data should be large and of high quality. However, labelling data requires resources in form of expertise and time, therefor it is not always easy to acquire a large set of training data. Some classification approaches does not require a high amount of data to perform well and can be relevant where there is little or no training data [5][7].

The quality of the labeled data that is provided is also important for the training process. Data which is gathered from real-world data sets might not always have the best features or quality in order to be used in training. For example, attributes in the data might turn out to be incorrect, absent or unnecessary. That is why a quality assurance of the training data is crucial in order to have a best possible chance of having a good output result. We will later in our thesis go into detail for how we acquired our data set, how we validated the quality and on what issue we encountered.

**Validation**

The validation phase is done in order to validate the learning of the training phase. The system will be provided with an unlabeled data set which it will be required to classify by itself. If the
percentages of true positives and true negatives are high after the validation is performed then it can be concluded that the validation was successful. On the other hand, if the validation was unsuccessful then the features should be reconsidered. If the features represent the classes poorly then the outcome of the validation can be unsatisfactory. If there are too many irrelevant features then it might result in the model becoming more complex than necessary. Another reason of bad training can be the quality of the data which were used as we discussed in the previous section.

The validation can be done in multiple ways. One approach is by splitting the data set into a training set and a validation set. Another alternative is by cross-validating the full data set to ensure and validate all samples which have been collected. We have used cross-validation in order to train and validate our data set, and we will describe this more in detail in the next chapter.

**Testing**

The testing phase is done as a final quality assurance of the previous phases. In this phase we wish to have a diverse data set in order to challenge the trained machine learning model. The main goal here is to see how the machine learning model can adapt to slight changes in a new sample or to see how it handles data which is not equal to the samples represented in the training data set. What we want here is to have a data set that contains various samples of all classes. When we do the testing it is also of interest to count accuracy and numbers in terms of incorrect classifications for each class.

**Summary of classification**

As a summary, the previously mentioned phases of training and validating a system can be compared to a student taking a school subject. The process of picking features can be compared to a teacher picking the relevant curriculum for the subject. The student will ideally follow the chosen curriculum and try to learn the concepts of what is presented. The student will likely be given a mandatory assignment or test during the school semester in order to verify what they have learned (validation phase). If the student does not pass this test then they have to go back to revising what they have not learned (training phase). The final exam (testing phase) will be given as a way of testing the student’s knowledge based on their previous learning. The ideal way of checking this is testing based on the curriculum of what they have learned, but give
them challenges which they should have learned based on abstract concepts of the curriculum. If the final exam (testing) is too similar to the mandatory assignment (validation) then it might be easy for the learner to recall the previous answers. However, if the final examination is challenging then the teacher is hopeful that the student will be able to perform well to show that they have learned how to handle new information based on their previous learning.

![Training Validation Testing](image)

*Figure 6: Machine learning phases*

### 3.4 Unsupervised learning

The approach of unsupervised learning is to let the learning system receive unlabeled data that it will attempt to find hidden structures and patterns in order to put the samples in classes. Since there are no labeled data then it does not have any way of identifying errors based on evaluating predictions. One common approach to unsupervised learning is by clustering. Clustering will group objects so the more equal samples used in the training phases are grouped together.

### 3.5 Reinforcement learning

Reinforcement learning is different from both supervised learning and unsupervised learning. The system will have guidance from an external evaluator where it will receive a numerical reward based on their prediction [8]. It will not have a learning phase, but it will be learn by trial and error.
4 Implementation and technical details

4.1 Introduction

The machine learning system we have developed for this thesis is constructed to parse, preprocess and analyze HTTP logs. The desired outcome of the system is to classify whether a HTTP request is associated with exploit kit activity or not. Our system will apply techniques to increase the quality of the classification, and to lower the false positive and false negative rates.

In this chapter we present how our machine learning system is constructed, what features it contains, how the system is performing the classification and discuss challenges which are associated with machine learning.

4.2 Overview

The system we have developed in this thesis applies machine learning techniques in order to distinguish and classify exploit kit traffic in a HTTP log format. It contains different code components in order to process input data.

The system consists of the following modules:

- Log parser
- Log handler
- Labeller
- Feature extractor
- Session handler
- Training phase
- Testing phase

All the modules together is the product of the machine learning system we have developed for this thesis.
4.3 System details

4.3.1 Design

![Flowchart of the system](image)

*Figure 7: Flowchart of the system*

4.3.2 Programming language and libraries

**Python**

The programming language used for developing the machine learning system in our thesis is Python. Python is a popular open source programming language that is often used for scientific computing. The Python language has support for a wide variety of scientific based libraries that can help us develop our machine learning system.
**Scikit-learn**

Scikit-learn is an open source based machine learning library [18]. The development was started in 2007 and is today a popular programming library for machine learning. It is developed to support SciPy, NumPy and matplotlib libraries. Scikit-learn is under active development and includes state-of-the-art implementations of different machine learning algorithms.

**4.3.3 Logs and training set**

The training set used is from user generated HTTP traffic gathered with the use of IDS-technology. Each HTTP request contains multiple data fields.

The following data fields are included in the logs we have available:

- Timestamp
- Domain
- URL
- User-agent
- Referer
- HTTP method
- Protocol
- HTTP code
- Amount of bytes
- Source IP address and port
- Destination IP address and port

We will use multiple fields for our features in order to train our machine learning system. One of the challenges in training the machine learning system is the process of choosing good and relevant features out of our log data. In order to have the best classification ability in our system then we need to have features that will not be too generic and not generate noise for the classification process.
4.3.4 Preprocessing

The preprocessing module’s main task is to preprocess the files and data. Our system will require an input of multiple data sets that will be used for analysis. The different sets we need in order to perform our machine learning analysis is a training set for the training phase and a testing set for the testing phase. All samples in both of the sets will need to have a label associated with it. During the training phase we will input two sets which are separated into exploit kit traffic and non-exploit kit traffic. In order to label the data set properly we will create a separate label array following the same index structure as the data sample array. As the files are read it will add each new sample into an array of data samples and add a corresponding label to the label array depending on which file the data sample originated from. When we have read both of the files we will merge the exploit kit array with the non-exploit kit array. Each data sample with an index will then have a corresponding label with the same index number in the label array that we can use for training the classifier. In order to validate the training set we will cross-validate it by splitting the set into folds. The cross-validation technique will be described more in detail later in this chapter. As for the testing phase, we will give the machine learning system an unlabeled data set that contains a mix of exploit kit traffic and non-exploit kit traffic. The testing set will be handpicked samples of both exploit kit traffic and non-traffic traffic that we consider to be possibly challenging to distinguish from each other. We will also have a corresponding label array for the testing set, but this will not be disclosed to the machine learning system as we want to manually check the results after the classification is finished.

4.3.5 Feature extractor

The feature extractor is responsible for handling data and extract the features we want to use in the machine learning system. The feature extractor will parse the data sample array we have stored in the preprocessing module and turn it into a format that we will use for each feature. It is important to create this module so it is easy to change features if we want to add or remove features in the feature set.

4.3.6 Machine learning

In this thesis we will apply multiple machine learning algorithms in order to try find a suggested solution to our research problem. There are many available algorithms we can use, but each algorithm has its strengths and weaknesses. We will attempt to find a machine learning
algorithm which suits our research problem and the data we have available in order to get the best possible outcome. The algorithms used are imported from the open source library Scikit-learn that we described in section 4.3.2.

We want to use the following algorithms to create the machine learning models for our thesis:

• Random Forest Classifier
• Support Vector Machine
• Multinomial Naive Bayes

The principle in many machine learning algorithms is that it requires a good quality data set in order to have high performance. However, acquiring enough data can be challenging. In addition, with the use of supervised learning we need labeled data sets that takes a lot of effort to be made manually. For our thesis we have collected real data sets where the traffic comes from IDSs. We have a large amount of benign client traffic available through the IDS-sensors we used for our research problem, but the challenge was that we do not have as much exploit kit traffic. We used both automatic and manual strategies in order to build a large exploit kit set so we could have a representable amount of volume and balance for our machine learning models. In order to validate the quality of the gathered exploit kit data we used reputation-based data to verify that the domain has been flagged for exploit kit traffic. Reputation based lists are distributed in a large-scale basis and based on the reputation we have available we were able to verify that the labelling is correct. As for the non-exploit kit data we verified that it did not contain any exploit kit data by using the same reputation sources. However, it is worth noting that the non-exploit kit data set may contain what is considered malicious requests. We do not consider malicious requests to be within the scope of our thesis as the research problem focuses on exploit kits. What we have defined as non-exploit kit traffic is therefore not what is considered benign traffic, but it is traffic that is not associated with exploit kit activity. By validating the data sets we could proceed to use them for training and testing.

4.3.7 Session handler

As we described in chapter 2 then the exploit kit follows a typical standardized chain of events. The session handler is constructed in order to take a single HTTP request from a log file and try to see it in context based on the logs we have available. By implementing a session handler,
we hope to create a module that can give us more features than from what we can extract with a single HTTP request. The module will look for distinct patterns out of the HTTP logs by counting data, occurrences and searching for patterns that are typically associated with exploit kit activity.

A log file may contain thousands of single requests and there are certain parameters we can use in order to find a session. Ideally, we could trace a session based on a source IP, but it is not always ideal since an IP address can be shared between many clients. This is possible with the use of Network Address Translation (NAT) or proxies where multiple clients can share a single IP and we can therefore not treat an IP address as a unique key factor. We will for our thesis define a session to be based on timestamps, source IP-address and user agents. If we combine these parameters, we can search for HTTP requests containing the same user agent and IP within a timeframe. In addition, we can try to link HTTP requests with the referrer field. If we are able to find the HTTP request chain then it can be helpful in order to determine if not only one request is malicious exploit kit related, but a partially or fully exploit kit activity chain where we observe redirects towards a payload download.

With former experience of working in a security operation center (SOC) we have analyzed a good amount of exploit kit events which have given us an idea of what to look for in a HTTP log. Based on our research in this thesis and from our previous SOC experience we have come up with the following session-based features:

1. Amount of requests towards a domain
2. Redirect from a third-party website

One distinct feature is the amount of HTTP requests towards a specific domain. Exploit kits are constructed to be swift and discreet. Meanwhile a user visits a compromised website then the malicious requests towards the domain with the hosted exploit kit happens in the background and the user will not notice the traffic unless they monitor their traffic. The exploit kits keep the amount of HTTP requests needed at a minimum to try exploit a victim as fast as possible. The amount of HTTP requests is a feature we can use to identify activity towards a specific domain with few requests. However, the amount of HTTP requests alone can be a weak feature because HTTP requests from a website can come from various sources. As an example, a user
might visit a website with embedded advertisements. If the user makes requests towards the domain of the advertisement server then the requests might be restricted to a few requests. In addition, requests towards advertisement servers can also look suspicious by eye due to the URL structure and can sometimes be mistaken as exploit kits. The amount of requests is therefore not a great feature by itself, but it can greatly help the machine learning model to get a better context of what activity happened in a single session towards a domain.

Another feature we can extract from a session is by identifying the redirect chain. The redirects of an exploit kit are inevitable and can be identified through the referer field in HTTP logs. From the start of the exploit kit chain through the successful payload delivery there are at least three redirects. The first redirect will typically have a referer of a compromised legitimate website and will be made towards the landing page hosted on the exploit kit server. The following redirects from the landing page and the exploit page will be made from the same domain. We have based this feature on the previous mentioned session feature and it will flag the session if the requests towards a domain started with a redirect from an external domain.

4.4 Challenges

4.4.1 Overfitting

It is important to have in mind the concept of overfitting when we select features and assemble enough data for the machine learning model. Overfitting happens when the model is overly complex and has too many inputs in the features which creates noise. This is challenging because the machine learning model may perform well for the training data you input, but it does not perform well on new data we feed it after the training is finished. A model that is overfit does generally have a poor predictive ability as a result that it will easily handle small, unexpected changes from what it observed in the training phase.
4.4.2 Underfitting

Underfitting is when our machine learning model is not able to capture a pattern of the training data. The result is likely to be that our model will not do well with new data, but in addition it will also do poorly with the training data we provide. An example of why models can become underfit is because the user is not able to find a solution that fits the data for the problem and algorithm. The overall assumption of the data becomes too generic for the model, as for example, where the data is assumed to be of a linear model, but it is not able to fit to the expectation.

There are some measures we may take in order to try solve high variance or high bias. Some of the countermeasures we can apply are listed below.
High bias:
- Increasing the complexity of the model

High variance:
- Adding more data for more diverse features
- Reducing the complexity of the model

The model complexity of a machine learning model may not always be what one had expected before attempting to fit it. A trial-and-error strategy can be performed in order to find out how well the machine learning model is able to fit the data we have available. The restriction of a machine learning model is that it is limited by a complexity which is based on its algorithm. If we then choose a model that is not properly suited for our data or problem then it can perform poorly. If we have a complex model to a complex data set then we get a model that fits the data in a desired way. On the other hand, we have underfitting where a model simply do not fit the data we have and the model can be considered too simple for what we want to achieve.

![Underfitting, Good fit, Overfitting](image)

*Figure 9: Example of underfitting and overfitting*

Figure 9 illustrates the difference between underfitting and overfitting. While the concept of underfitting can be addressed by attempting to use another machine learning model, then the problem of overfitting is an often more challenging task to solve. We want ideally to have a model which is somewhere in-between overfit and underfit. However, achieving this can be challenging because it may be both time consuming and require a structural approach.

As long as we feed more data into a machine learning model then the algorithm will learn. If the training continues for too long then the result of the training set may not be as effective anymore because the model has started overfitting. When it starts overfitting then the model has started phasing into the right part of Figure 9 and began learning unnecessary details from
the data it is trained with. The key is that we want the machine learning model to be accurate, but adaptable.

Appropriate data is also a key factor in having a good and balanced machine learning model. When we try to fit our provided data then it is challenging to filter out noise from quality data. If we have a complex model with too much noise then the result will likely be poor. This is because the model takes the noise into consideration when it predicts data it has not seen before. The result of the noise can result in a false positive that is, performance wise, something we want to avoid.

As a summary of the two previous sections we can conclude that a concept of having a clear generalization is important to focus on since the machine learning model will often have a high bias or high variance.

### 4.5 Machine learning algorithms

In the system we are developing for this thesis we will train multiple machine learning models. This section will shortly describe the different models we want to use. In addition, we will present some techniques we will use in combination with the models.

#### 4.5.1 Random Forest Classifier

Random Forest Classifier is a classifier that combines a collection of different non-correlated decision trees. The decision trees are generated randomly by selected subsets of the provided training set. Each of the decision trees can be considered weak on its own and will be prone to potential noise in the training data. However, by combining all the single decision trees together we hope to cancel out noise and the total product of combining all the created decision trees together is what we predict to create a stronger learner. Random forest will generally use the decision trees in order to aggregate votes to decide on the classification of the test data, but the classifier we will use has an alternative take on the voting aggregation and will instead use averaging for the probabilistic prediction [24].
4.5.2 Support Vector Machine

Support Vector Machine (SVM) is a supervised learning model that features algorithms to be used with classification or regression. In SVM, each sample has a plot in an n-dimensional space whereas each feature value creates a particular coordinate. The learning model will then proceed to classify the samples by differentiating them into groups. The separation is done by a hyper-plane and if the classification is binary then the hyper-plane will function as a simple line separating the two groups. Any data plot which is present on a specific side of the hyper-plane belongs to the specific class. In order to find the best hyper-plane then the SVM will look at the distance to the nearest data plot of each class. The hyper-plane is considered better based on the higher margin of the distance between hyper-plane and the nearest class data plot.

The SVM supports different kernel functions for the trained model. For our thesis we have decided to use the a linear kernel based on LIBLINEAR [32][33]. LIBLINEAR is a linear classifier based on SVM that is efficient and scales well with high amounts of data.

4.5.3 Multinomial Naïve Bayes

Naïve Bayes is a simple, but strong machine learning algorithm. The reason why the algorithm is called naïve is because it considers the value of a feature to be independent of any other features. It will therefore ignore any possible correlation between the given features and this can be a strength in some cases. For our thesis we will include lexical features from the URL and domain which can be used in favor in the Naïve Bayes model.

The Bayes’ Theorem is traditionally given as the following:

\[
p(x|y) = \frac{p(y|x)p(x)}{p(y)}
\] (4.1)

There are different versions of the Naïve Bayes classifiers, but for this thesis we will use Multinomial Naïve Bayes. Multinomial Naïve Bayes works similar to Naïve Bayes, but instead of considering if a feature is present or absent, it will instead count how many times the outcome occurs. Bayes’ Theorem will be used in section 6.3 to compute the likelihood of true positive alerts when using machine learning to detect exploit kits.
4.6 Evaluation techniques

4.6.1 K-fold cross validation

For validating the results of the trained machine learning models we will use a validation technique called *k-fold cross-validation*. In k-fold cross-validation the data set is partitioned into *k* subsets of the same size which we refer to as *folds*. For each iteration, one of the *k* subsets is held back to be used as a validation test while the other *k* - 1 subsets are to be used to train the machine learning model. The process is then repeated *k* times and each of the *k* subsets are to be held back as a validation set. K-fold cross-validation is applied in order to use the full dataset as validation instead of choosing a random subset which can have an unbalanced bias.

We can decide on our own what we want the *k* parameter to be in the k-fold cross-validation technique. However, if we decide to let the *k* be a small value, as for example 2 to 5, then the variance of the validation can be higher due to instability of the training sets. We will choose 10 k-folds as it has previously been tested to reduce variance [19].

Our k-fold cross-validation will then be as following:

<table>
<thead>
<tr>
<th>#</th>
<th>Train folds</th>
<th>Test fold</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 10</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>1, 2, 3, 4, 5, 6, 7, 9, 10</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>1, 2, 3, 4, 5, 6, 8, 9, 10</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1, 2, 3, 4, 5, 7, 8, 9, 10</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>1, 2, 3, 4, 6, 7, 8, 9, 10</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>1, 2, 3, 5, 6, 7, 8, 9, 10</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>1, 2, 4, 5, 6, 7, 8, 9, 10</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>1, 3, 4, 5, 6, 7, 8, 9, 10</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>2, 3, 4, 5, 6, 7, 8, 9, 10</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 3: K-fold cross-validation*

We can add the *k* measurements together and create an average score for the performance of our trained machine learning model.
A normal k-fold cross-validation technique will separate the folds at random. The risk of doing this is that the folds may be unbalanced based on the classification groups. Rather than having the process be decided by random, we will for the validation phase use \textit{stratified k-fold cross-validation}. The technique is equal to a normal k-fold cross-validation, but each of the folds are created so they have a balanced representation of the full dataset. Previous research has shown that stratified k-fold cross-validation is consistently better than regular cross-validation both with bias and variance in mind [19].

For our thesis the stratified k-fold cross-validation technique was a natural choice because it covers both training and validation phases of the machine learning phases we mentioned in chapter 3. Alternatively we could have split the data set into a training set and validation set, but this could have created a coincidental bias in the performed split. By using the k-fold technique we will be able to test our full data set and verify the validation results of all the folds.

### 4.6.2 One-hot encoding

For our machine learning system we have chosen multiple features which are presented as categorical attributes. Strings and categorical features does not work well with some machine learning models such as linear models and SVM [25][26]. One way to deal with this issue is to transform the categorical features into nominal values. For each unique categorical feature we could had transformed it into a unique nominal key ID. However, a machine learning model would be likely to interpret these as numerical values and not as an ID. As an example, we have a set of categorical features. We want to encode them into nominal values so the machine learning model will process them properly.

<table>
<thead>
<tr>
<th>Sample #</th>
<th>Country</th>
<th>Sample #</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Norway</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Sweden</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Denmark</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Norway</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Finland</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>Iceland</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

\textit{Table 4: Converting categorical to nominal}
The machine learning model would interpret the ID here as actual nominal values. This would result in some unwanted predictions as the following:

- Sweden is the average of Norway and Denmark
- Finland is greater than Denmark
- Iceland is much closer to Finland than Norway

This is not how we originally intended the values to be interpreted. One way to deal with this problem is by transforming categorical features to nominal values using a technique called *one-hot encoding*. One-hot encoding creates a new feature for each unique categorical value introduced in the sample set. The new feature is then used for the machine learning model to interpret whether each categorical value is present in the sample data.

<table>
<thead>
<tr>
<th>Sample #</th>
<th>Country is Norway?</th>
<th>Country is Sweden?</th>
<th>(…)</th>
<th>Country is Iceland?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(…)</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>(…)</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>(…)</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>(…)</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>(…)</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>(…)</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 5: One hot encoded values*

One-hot encoding works well with classification and regression models, and since we will primarily use classification for our thesis then it will be a good technique to use in order to get more value of categorical data.
5 Results

In this chapter we discuss the experimental results we have obtained by implementing the system for our thesis. In addition, we discuss some of the challenges we encountered during the implementation and analysis. We also present the performance and the results of our system. Lastly, we take a closer look at the data set that we used and present the selected features.

5.1 Challenges and experiences

5.1.1 Gathering data sets

One of the challenges we encountered during the work of this thesis was the task of gathering relevant sample data. Our research problem was based on using data logs from an IDS and we wanted HTTP logs to be our primary source of data for training the machine learning system. In our thesis we had access to HTTP logs through mnemonic’s IDS technology which monitors networks in different continents around the world. This gave us the opportunity to search in stored logs to find relevant data which we could use in order to train our machine learning system.

5.1.2 Finding the exploit kit data

Finding the relevant exploit kit data in thousands of HTTP logs can be an activity similar to finding the needle in the haystack. We had thousands of log files available, but only a few of them contained actual exploit kit activity. In order to extract the activity we needed to come up with a strategy in order to gather the necessary exploit kit data we were to use in our thesis. We used primarily two strategies in order to gather the logs which we needed to create an exploit kit data training set.

Parse HTTP logs

We wanted to automate the extraction process of gathering exploit kit data. We created a script which would parse all stored HTTP logs on an IDS and store the search result which would be relevant to exploit kit activity in a separate file. In order to distinguish the exploit kit and non-exploit kit traffic we used reputation based data to identify domains which were associated with
exploit kit activity. The risk of using this method is that if the reputation sources are weak then it will create a lot of false positives. We manually examined the extracted data set after it was assembled and it showed that the reputation sources we had available were good enough to avoid too many false positives. Another downside of using this method is that by parsing through all HTTP logs on the IDS it may cause performance issues. Depending on the specifications of the hardware the IDS is running on then it may cause load issues and in a worst-case scenario have incoming network packets become dropped. This is not something we want for an IDS and during the automatic parsing we monitored the IDS performance closely. Before parsing the logs we verified that the IDS could handle the parsing and no packets were dropped as a consequence of our thesis work.

We consider the following strengths and weaknesses using the described method:

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to use</td>
<td>Requires solid validation sources</td>
</tr>
<tr>
<td>Saves time</td>
<td>Risk of false positives</td>
</tr>
<tr>
<td>Little manual work</td>
<td>May affect IDS performance</td>
</tr>
</tbody>
</table>

*Table 6: Pros and Cons of automatic extraction*

**Manually extract from logs**

The second method we used to find data for the exploit kit set was by looking for triggered IDS alerts related to exploit kit activity. The alerts includes a timestamp which we could use as a reference to which IDS the traffic was seen and in which log file that contained the HTTP log entry. The IDS alerts are mainly generated as a result of signature- and reputation-based detection mechanisms. Since it is crucial to produce a significantly large training set for the exploit kit data in our machine learning system then we found this method was way too time-consuming. However, by extracting the data manually we could verify the quality of each request we added to our data set.
We consider the following strengths and weaknesses using the described method:

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality assurance</td>
<td>Time consuming</td>
</tr>
<tr>
<td>Less false positives</td>
<td>Risk of human-error</td>
</tr>
<tr>
<td>Get a better overview of the data</td>
<td>Requires some experience to validate</td>
</tr>
</tbody>
</table>

Table 7: Pros and Cons of manual extraction

5.1.3 Gathering enough relevant data

Another challenge we experienced was finding enough relevant data to represent the exploit kit set. We had more than enough non-exploit kit data, but finding actual exploit kit data was harder because the ratio between each non-exploit kit request and exploit kit request is very high. We will explain this more in detail later in the thesis.

5.1.4 Disappearance of a high profile exploit kit

During our work on the thesis and gathering the data set we experienced an example on how rapid the threat landscape of information security can change. In the start phase of our thesis we identified the most used exploit kits to be the Angler exploit kit which was the most active kit at the time. However, in June 2016 a very noticeable change happened: all activity related to the Angler exploit kit almost vanished instantly. There has been no official confirmation to why it happened, but Russian authorities had arrested a group of individuals related to the Lurk banking Trojan around the same time as the disappearance [27]. Research done by Cisco Talos in [27] associated registrar information of the domains used by both Angler and Lurk, and it showed that there were clear links between the exploit kit and the crimeware.

For our thesis this meant that it became more challenging to gather relevant exploit kit data from the HTTP data we had available. Angler was the major high profile exploit kit when we started the planning of the thesis and the disappearance resulted in a stagnation of exploit kit activity. We had fortunately started the gathering data sets at an earlier stage, but the void of the Angler kit was noticeable for our thesis work. In the aftermath of this event, we noticed that activity related to Neutrino and RIG exploit kit were increasing.
5.2 Classification data

For the rest of this chapter we will present the result of the classification. We want to present the performance for each of the machine learning models we have used and compare the results. The classification was done in two main steps:

1. Cross-validating the training data set
2. Testing the machine learning model with an additional set

The cross-validating step is done in order to validate the data set we use for training. The testing is done afterwards to challenge the machine learning model with data samples we believe can be difficult to classify.

5.2.1 Data set

The training set consists of samples that we have used for training our machine learning models. Each data sample is converted to a feature set format that we have defined. We have the following sets and contained samples:

<table>
<thead>
<tr>
<th></th>
<th>Exploit kit samples</th>
<th>Non-exploit kit samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training set</td>
<td>8642</td>
<td>16868</td>
</tr>
<tr>
<td>Testing set</td>
<td>500</td>
<td>1000</td>
</tr>
</tbody>
</table>

*Table 8: Data sets overview*

The training set is used for cross-validating and the testing set is used for a deeper analysis of the machine learning performance. There are various exploit kits made by different authors. We have mentioned some exploit kits in the previous chapters and we have decided to include three different exploit kits that have been active in recent years. In total we have sampled data from the following exploit kits:

<table>
<thead>
<tr>
<th></th>
<th>Angler</th>
<th>Neutrino</th>
<th>RIG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training set</td>
<td>6092</td>
<td>1408</td>
<td>1642</td>
</tr>
</tbody>
</table>

*Table 9: Exploit kit data set overview*
5.2.2 Features

We have defined a set of features that are created from the data sets we have available. The features can be combined into two categories:

- Lexical-based features
- Session-based features

Both of the categories are solely based on HTTP logs. We could have used additional feature categories based on external data or content-based data, but we wanted to limit the features to what we had available on an IDS.

For the features we have chosen there are three different value types:

- Categorical
- Nominal
- Binary

The categorical values consists of strings that we will convert as described in section 4.6.2. The nominal values are integer values. Lastly, we have binary values which are based on thresholds. The binary values are primarily used in the session-based features where the logs are analyzed by the session handler and the value is set depending on whether the analysis results are over or under the given threshold.

The features are based on certain data in the data samples. The type of data we have primarily used is the following:

- Domain
- URL
- Session
Domain features

A domain is a string that correlates the name of an Internet host. Malicious domains can sometimes be recognized in terms of being all made up of arbitrary characters which makes them look suspicious. However, many exploit kits use domains that are not necessarily something that stands out from the benign traffic domains. For example, some exploit kits use randomly generated domains by appending different words from a dictionary to form a domain name. Without former analyzing experience it can be challenging to identify these domains, but the domains usually consist of high character length and is built from words that have no relation to each other. Another interesting trait of the domain is the inclusion of the top-level domain (TLD). TLD is the last identifier in a domain name which correlates to the Domain Name System (DNS) of the Internet. As an example, the TLD in “www.example.com” is the “.com”. Many exploit kits use TLDs that are not as commonly used by non-exploit kit traffic and the TLD will then work as a keyword for the classifier.

We also decided to include entropy as a domain feature. The entropy is defined in terms of probability. If a text string contains only “AAAAA” then the entropy will be calculated as 0 because the next character is always the same. On the contrary, the entropy value will be higher if a string contains multiple characters. The value will then reflect how difficult it is to predict each character in the given string. We believe the entropy will help us give an estimate of the complexity of a domain and can be useful for identifying domains that has been automatically generated with arbitrary characters or by appending dictionary words.

URL features

The URL appends the domain or IP in order to reference a specific resource of the host. The URL can give us valuable information such as keywords, directories and characters that can be useful for our machine learning system. As an example, let us say a request is made for the resource “www.example.com/evil/evilscript.js”. The requests indicates that a requests has been made for the domain “example.com” and the client wants to access a JavaScript(JS) file named “evilscript.js” in the directory “evil”. Important keywords and information can be useful for our machine learning system from the extracted URL’s in the HTTP logs.
In addition, we have created features that will count the number of occurrences of a specific character. We have decided to include “&”, “/” and “-” since all of these are frequently used in many exploit kit URL’s.

Lastly, we have included a feature that calculates the length of a URL. The length of the URL is calculated by the amount of characters in the string. Many exploit kits have what we consider long URLs and it is of interest to see whether it can help the machine learning system to classify samples correctly. However, we consider this feature alone to be very weak as it is also normal for legitimate requests to have long URLs.

**Session features**

A HTTP log contains several single HTTP requests. We can perform an analysis by examining these requests one by one, but by doing so we are unable to get a better context on why the HTTP requests were made and information about the current session made by the client of the HTTP requests. As we already have described, the exploit kit chain is restricted to specific steps in order to successfully infect a victim. We want to use this strict and static behavior to make context-based behavior analysis.

The first feature we have decided to include is to create a binary feature that evaluate the amount of requests that has been made towards a domain from a specific client. In order to identify this we have used source IP, user agent and domain to map the requests and search through the HTTP logs. An exploit kit chain is restricted in behavior and it is developed to have a minimal amount of HTTP requests in order to quickly infect a victim. From the redirect to the payload there are at least four requests, but requests towards the exploit page can sometimes be repeated in order to attempt to exploit multiple vulnerabilities. Based on this observation we have decided to set a threshold on five or less requests towards a domain.

The second feature we have decided to include is the observation of a redirect in a session. The attacker is dependent on having traffic redirected towards the exploit kit server and will attempt to compromise third-party websites. The binary feature value is then set on whether the session includes a redirect from a third-party website. However, it is important to note that redirects are also used in legitimate web browsing, but the redirect is an essential first step of the exploit kit chain and we have therefor included this as a part of the feature set.
The session features allows us to see the HTTP requests in context. The features are binary values set by thresholds and are based on conditions we have considered to be traits of exploit kits. We believe the features can help the machine learning models to differentiate the exploit kit and the non-exploit kit traffic.

**Selected features**

The features we have selected for our machine learning models are the following:

<table>
<thead>
<tr>
<th>Category</th>
<th>Data</th>
<th>Feature</th>
<th>Value type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lexical</td>
<td>Domain</td>
<td>Domain name</td>
<td>Categorical</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Entropy</td>
<td>Nominal</td>
</tr>
<tr>
<td></td>
<td>Domain</td>
<td>URL</td>
<td>Categorical</td>
</tr>
<tr>
<td></td>
<td></td>
<td>URL length</td>
<td>Nominal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Count ‘&amp;’</td>
<td>Nominal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Count ‘/’</td>
<td>Nominal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Count ‘-’</td>
<td>Nominal</td>
</tr>
<tr>
<td>Session</td>
<td>Session</td>
<td>Session requests</td>
<td>Binary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Redirect</td>
<td>Binary</td>
</tr>
</tbody>
</table>

*Table 10: Selected features*

In total we have 9 features divided into two categories that is based on 3 different data types. The features are chosen based on observations we have made from analyzing the exploit kit traffic compared to the non-exploit kit traffic. We also tested multiple features in terms of performance and we concluded that the listed feature set represented the best performance. We will use these features for our machine learning system when we will train and test the models.
5.3 Classifier performance – cross-validation

5.3.1 Data set

The training set consists of data samples that we have assembled by using manual and automatic extracting methods in order to sample events from the IDSs.

5.3.2 Binary classification

Binary classification is done by defining two groups to distinguish from each other. This suits our research problem well since we want to detect exploit kit traffic from non-exploit kit traffic. In order to test the performance of each machine learning model then we have used stratified cross-validation. By using cross-validation we can validate the full data set and minimize the risk of having issues in terms of bias and variance. The accuracy presented for each fold in the following sections indicates the correct classification percentage score for the contained data.

Support Vector Machine

The following table presents the binary classification results by cross-validating the data set with Support Vector Machine:

<table>
<thead>
<tr>
<th>Fold</th>
<th>Accuracy(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>93.06</td>
</tr>
<tr>
<td>2</td>
<td>96.55</td>
</tr>
<tr>
<td>3</td>
<td>97.45</td>
</tr>
<tr>
<td>4</td>
<td>97.76</td>
</tr>
<tr>
<td>5</td>
<td>96.43</td>
</tr>
<tr>
<td>6</td>
<td>97.49</td>
</tr>
<tr>
<td>7</td>
<td>97.65</td>
</tr>
<tr>
<td>8</td>
<td>97.96</td>
</tr>
<tr>
<td>9</td>
<td>98.04</td>
</tr>
<tr>
<td>10</td>
<td>91.49</td>
</tr>
<tr>
<td>Sum(avg.)</td>
<td>96.39</td>
</tr>
</tbody>
</table>

*Table 11: Binary classification results: Support Vector Machine*

The average accuracy of the 10 folds that was cross-validated is 96.39%. The difference between the minimum and maximum score is 6.55%.
Random Forest Classifier

The following table presents the binary classification results by cross-validating the data set with Random Forest Classifier.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Accuracy(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>99.29</td>
</tr>
<tr>
<td>2</td>
<td>99.10</td>
</tr>
<tr>
<td>3</td>
<td>99.10</td>
</tr>
<tr>
<td>4</td>
<td>99.14</td>
</tr>
<tr>
<td>5</td>
<td>99.14</td>
</tr>
<tr>
<td>6</td>
<td>99.06</td>
</tr>
<tr>
<td>7</td>
<td>98.86</td>
</tr>
<tr>
<td>8</td>
<td>99.33</td>
</tr>
<tr>
<td>9</td>
<td>99.02</td>
</tr>
<tr>
<td>10</td>
<td>99.10</td>
</tr>
<tr>
<td><strong>Sum(avg.)</strong></td>
<td><strong>99.11</strong></td>
</tr>
</tbody>
</table>

*Table 12: Binary classification results: Random Forest Classifier*

The average accuracy of the 10 folds that was cross-validated with our data set is 99.11%. The difference between the minimum and maximum score is 0.47%.

Multinomial Naïve Bayes

The following table presents the binary classification results by cross-validating the data set with Multinomial Naïve Bayes.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Accuracy(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>87.70</td>
</tr>
<tr>
<td>2</td>
<td>87.19</td>
</tr>
<tr>
<td>3</td>
<td>87.77</td>
</tr>
<tr>
<td>4</td>
<td>86.95</td>
</tr>
<tr>
<td>5</td>
<td>89.18</td>
</tr>
<tr>
<td>6</td>
<td>87.81</td>
</tr>
<tr>
<td>7</td>
<td>89.38</td>
</tr>
<tr>
<td>8</td>
<td>88.28</td>
</tr>
<tr>
<td>9</td>
<td>88.00</td>
</tr>
<tr>
<td>10</td>
<td>88.82</td>
</tr>
<tr>
<td><strong>Sum(avg.)</strong></td>
<td><strong>88.11</strong></td>
</tr>
</tbody>
</table>

*Table 13: Binary classification results: Multinomial Naïve Bayes*
The average accuracy of the 10 folds that was cross-validated with our data set is 88.11%. The difference between the minimum and maximum score is 2.43%.

**Model performance comparison**

A comparison of the different machine learning models is presented in the following table:

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Support Vector Machine</th>
<th>Random Forest Classifier</th>
<th>Multinomial Naïve Bayes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>96.39%</td>
<td>99.11%</td>
<td>88.11%</td>
</tr>
<tr>
<td>Minimum</td>
<td>91.49%</td>
<td>98.86%</td>
<td>86.95%</td>
</tr>
<tr>
<td>Maximum</td>
<td>98.04%</td>
<td>99.33%</td>
<td>89.38%</td>
</tr>
<tr>
<td>Inconsistency</td>
<td>6.55%</td>
<td>0.47%</td>
<td>2.43%</td>
</tr>
</tbody>
</table>

*Table 14: Binary classification results: model performance comparison*

If we compare the performance of the models then the Random Forest Classifier has the better overall results. Not only does it have the best mean score, but it did also have the most consistency of performance when comparing the minimum and maximum scores. The Support Vector Machine shows overall decent performance, but includes a trade-off with inconsistency. The difference between the best and worst fold in Support Vector Machine is 6.55%, which we consider high. The Multinomial Naïve Bayes model came last in terms of performance of our cross-validation. However, the consistency of the Multinomial Naïve Bayes was better than the Support Vector Machine. All in all, the Random Forest Classifier showed the most satisfactory performance and consistency by applying binary classification on our data set.

### 5.3.3 Multi-class classification

Multi-class classification extends the binary classification by supporting more than two groups. We want to extend our machine learning system by splitting the exploit kit data set into a class for each of the different exploit kits we have sampled. By doing this we will have four classes in total: Angler, Neutrino, RIG and non-exploit kit. We want to see how well the machine learning models can classify the different exploit kits and at the same time be able to separate them from the non-exploit kit data.
Support Vector Machine

The following table presents the multi-class classification results by cross-validating the data set with Support Vector Machine.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Accuracy(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>94.12</td>
</tr>
<tr>
<td>2</td>
<td>92.59</td>
</tr>
<tr>
<td>3</td>
<td>90.83</td>
</tr>
<tr>
<td>4</td>
<td>90.47</td>
</tr>
<tr>
<td>5</td>
<td>94.55</td>
</tr>
<tr>
<td>6</td>
<td>95.73</td>
</tr>
<tr>
<td>7</td>
<td>94.75</td>
</tr>
<tr>
<td>8</td>
<td>95.69</td>
</tr>
<tr>
<td>9</td>
<td>94.12</td>
</tr>
<tr>
<td>10</td>
<td>90.58</td>
</tr>
<tr>
<td>Sum(avg.)</td>
<td>93.34</td>
</tr>
</tbody>
</table>

*Table 15: Multi-class classification results: Support Vector Machine*

The average accuracy of the 10 folds that was cross-validated with our data set is 93.34%. The difference between the minimum and maximum score is 5.26%.

Random Forest Classifier

The following table presents the multi-class classification results by cross-validating the data set with Random Forest Classifier.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Accuracy(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>98.47</td>
</tr>
<tr>
<td>2</td>
<td>98.63</td>
</tr>
<tr>
<td>3</td>
<td>98.90</td>
</tr>
<tr>
<td>4</td>
<td>98.71</td>
</tr>
<tr>
<td>5</td>
<td>98.75</td>
</tr>
<tr>
<td>6</td>
<td>98.63</td>
</tr>
<tr>
<td>7</td>
<td>98.67</td>
</tr>
<tr>
<td>8</td>
<td>99.14</td>
</tr>
<tr>
<td>9</td>
<td>98.55</td>
</tr>
<tr>
<td>10</td>
<td>98.74</td>
</tr>
<tr>
<td>Sum(avg.)</td>
<td>98.72</td>
</tr>
</tbody>
</table>

*Table 16: Multi-class classification results: Random Forest Classifier*
The average accuracy of the 10 folds that was cross-validated with our data set is 98.72%. The difference between the minimum and maximum score is 0.67%.

**Multinomial Naïve Bayes**

The following table presents the multi-class classification results by cross-validating the data set with Multinomial Naïve Bayes.

<table>
<thead>
<tr>
<th>Fold</th>
<th>Accuracy(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80.92</td>
</tr>
<tr>
<td>2</td>
<td>80.25</td>
</tr>
<tr>
<td>3</td>
<td>80.96</td>
</tr>
<tr>
<td>4</td>
<td>80.60</td>
</tr>
<tr>
<td>5</td>
<td>81.81</td>
</tr>
<tr>
<td>6</td>
<td>80.44</td>
</tr>
<tr>
<td>7</td>
<td>81.58</td>
</tr>
<tr>
<td>8</td>
<td>82.12</td>
</tr>
<tr>
<td>9</td>
<td>81.29</td>
</tr>
<tr>
<td>10</td>
<td>82.10</td>
</tr>
<tr>
<td><strong>Sum(avg.)</strong></td>
<td><strong>81.21</strong></td>
</tr>
</tbody>
</table>

*Table 17: Multi-class classification results: Multinomial Naïve Bayes*

The average accuracy of the 10 folds that was cross-validated with our data set is 81.21%. The difference between the minimum and maximum score is 1.94%.

**Multi-class classification results**

A comparison of the different machine learning models is presented in the following table:

<table>
<thead>
<tr>
<th>Support Vector Machine</th>
<th>Random Forest Classifier</th>
<th>Multinomial Naïve Bayes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>93.34%</td>
<td>98.72%</td>
</tr>
<tr>
<td>Minimum</td>
<td>90.47%</td>
<td>98.47%</td>
</tr>
<tr>
<td>Maximum</td>
<td>95.73%</td>
<td>99.14%</td>
</tr>
<tr>
<td>Inconsistency</td>
<td>5.26%</td>
<td>0.67%</td>
</tr>
</tbody>
</table>

*Table 18: Multi-class classification: model performance comparison*

The overall performance of the multi-class classification is to be considered slightly worse than the binary classification. Again as with the binary classification, the Random Forest Classifier
had the best overall performance. Support Vector Machine shows decent performance, but with the trade-off of inconsistency. The performance of the Multinomial Naïve Bayes classifier shows that it handles multi-class classification much poorer than what it did with the binary classification. We consider that the Multinomial Naïve Bayes is not an algorithm suited for our data set based on the performance results. All in all, the Random Forest Classifier showed the most satisfactory performance and consistency by applying multi-class classification on our data set.

### 5.4 Classifier performance - testing

We have now finished presenting the results of our training and validation phase. As we mentioned in our machine learning chapter we will now proceed to do the testing. The goal of the testing phase is to analyze the performance of the machine learning system more in depth. We will test the system with a separate data set that will include other samples than what we used for the training set.

For the rest of our thesis we will proceed to use binary classification in combination with Random Forest Classifier to discuss our research. The reason we have taken this decision is because binary classification and Random Forest Classifier had the better classification performance. In addition, the Random Forest Classifier had the best results in terms of consistency. We consider consistency important in the testing phase as there will only be one data set and no-cross validation. The binary classification showed a slightly better performance than the multi-class classification. We also rate the machine learning system’s ability to differentiate the exploit kit traffic and non-exploit kit traffic to be more important than having a class for each exploit kit.

### 5.4.1 Data set

The data set we have used for testing the machine learning system consists of a data set we have manually verified and consist of data that are not included in the training set. In total there are 1500 samples whereas 500 are exploit kit samples and 1000 are non-exploit kit samples. We have tried to include data we have considered might be challenging for the machine learning classifier to test its performance.
5.4.2 Detection rate

We will proceed to present the testing results of the machine learning system. The samples of the testing set is divided into two classes: exploit kit and non-exploit kit traffic. The testing is done by parsing through the testing set and the machine learning system will attempt to correctly classify each sample. We have a separate label set which contains the correct label for each sample, but the machine learning system will not be presented this label set. By each classification we will compare the machine learning system’s answer to the label set. The performance of the machine learning system is presented in the following table:

<table>
<thead>
<tr>
<th></th>
<th>Correct</th>
<th>Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploit kit</td>
<td>499</td>
<td>1</td>
</tr>
<tr>
<td>Non-exploit kit</td>
<td>958</td>
<td>42</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1457</strong></td>
<td><strong>43</strong></td>
</tr>
</tbody>
</table>

*Table 19: Testing results*

Based on the numbers of correct and incorrect classifications, we can conclude that the classifier does well with the exploit kit data. However, the amount of incorrect non-exploit kit data is significantly higher. By inspecting the incorrect classified samples we have found that the machine learning classifier is challenged by non-exploit kit samples with the following traits:

- Session features equal to the exploit kit data
- Includes subdomains
- High entropy
- Complex URLs

Many of the incorrect classifications were towards the same domain. For example, there were multiple requests towards a domain related to auto-updates of legitimate software. However, these requests had features that could resemble exploit kit samples and were as a result incorrectly classified.
The overall accuracy of the testing phase was 97.13%. If we break down the accuracy based on each class then the accuracy of the non-exploit kit samples was 95.80% and the exploit kit samples was 99.80%.

### 5.4.3 Feature importance

The features we chose were decided based on our observations of exploit kit features compared to non-exploit kit traffic. In this section we want to look closer on the importance of the features we have chosen and see what impact they had for the classifier.

**Domain name**

Importance: 16%

The domain consists of the domain name and the TLD. By analyzing the data sets we wanted to see if we could find a distinct difference between the exploit kit samples and the non-exploit kit samples. One clear difference is the distribution of the domain’s TLD. The exploit kit domains widely use alternative TLDs that are not particularly common, and a domain containing an uncommon TLD associated with a bad domain would then be weighted towards being classified as an exploit kit sample.

**Entropy**

Importance: 10%

Entropy had a 10% importance based on the testing classification. The entropy feature calculated the entropy of the domain and potential subdomains. We wanted to include this feature because exploit kit domains tend to be complex in a specific way. Malware authors use domain generation algorithm (DGA) in order to create malicious domains and some malware tend to generate arbitrary characters to form the domain name. However, some exploit kits tend to be more sophisticated as they try to resemble a legitimate domain name. For example, the domain name can consist of multiple randomly picked dictionary words and as a result the entropy increases. Initially we believed this feature would be an important feature, but it had a little less importance of what we expected. However, the 10% importance indicates that it plays a role in separating the exploit kit traffic and the non-exploit kit traffic.
URL
Importance: 9%

We wanted to use the URL as it potentially contains many keywords that would be valuable for the machine learning classification. Exploit kits may contain certain keywords in the URL and we believe this is the reason why the URL feature had 9% importance.

The length of the URL was also included as one of the chosen features. However, it turned out that the length had no impact on the outcome of the machine learning classification. Many exploit kits have URLs with high character count, but the result here concludes that it is also normal for legitimate traffic and the feature does not contribute separate the two defined classes.

Symbol count
Importance: 15%

The nominal values that counted the URL for specific characters were included because of an observation that many of the exploit kits used the included characters. The count for the forward slash had the highest impact with 10%.

Session request count
Importance: 18%

The session request feature was created due to an observation that exploit kits are restricted to the events of the exploit kit chain and that they are limited in terms of the total amount of requests. We decided to create a threshold of five or less requests and the threshold would determine the binary value of the feature. The importance of the session requests reflects that it was a good feature to include. Domains that are legitimate do often have many requests per session due to high amount of content that is required to load website. The feature flags the different type of sessions that has been made and will help the machine learning system distinguish between a single request that has a lot of associated session requests, and the session where there are only a few requests towards a specific domain.
Redirect

Importance: 32%

The redirect is the first step of the exploit kit chain. We wanted to include the feature because it would help the machine learning system identify sessions towards a domain initiated due to a redirect from a third-party site. If the session would be an exploit kit chain then the third-party site would be the compromised host. The feature importance indicate that the feature had a high impact of the classification.

Feature importance ranking

The following table shows the ranked feature importance from the testing:

<table>
<thead>
<tr>
<th>Ranking</th>
<th>Feature</th>
<th>Impact %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Redirect</td>
<td>32%</td>
</tr>
<tr>
<td>2</td>
<td>Session count</td>
<td>18%</td>
</tr>
<tr>
<td>3</td>
<td>Domain</td>
<td>16%</td>
</tr>
<tr>
<td>4</td>
<td>Entropy</td>
<td>10%</td>
</tr>
<tr>
<td>5</td>
<td>URL count /</td>
<td>10%</td>
</tr>
<tr>
<td>6</td>
<td>URL</td>
<td>9%</td>
</tr>
<tr>
<td>7</td>
<td>URL count &amp;</td>
<td>3%</td>
</tr>
<tr>
<td>8</td>
<td>URL count -</td>
<td>2%</td>
</tr>
<tr>
<td>9</td>
<td>URL Length</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 20: Feature importance ranking
The session features have the highest impact on the classification. An interesting observation is that both of these features are binary features. We wanted to create the session features to be what would separate the exploit kit traffic and the non-exploit traffic, which seems to have paid off result wise. Another observation is that the subset of URL features had the least importance. Initially we believed the URL would have a higher impact on the result of the classification since many keywords in forms of tokens can be extracted from it.
6 Discussion

6.1 Introduction

Based on our machine learning results we will now proceed to discuss the performance and impact this would have had in a real network deployment scenario.

6.2 Base-rate fallacy

In the previous chapter we presented the detection rate of the machine learning system. Even though the percentage accuracy can be considered high then we need to remember the focus and limitations of our thesis:

- The machine learning system is developed with focus on IDSs
- Only lexical data is to be processed which we have available through the IDS HTTP logs

Our training set consists of approximately 25 000 data samples which we have assembled from HTTP logs. In addition, we had a test set of 1500 samples. When we performed the testing of our machine learning system in the previous chapter then the results showed 1457 correct and 43 incorrect classifications. It is necessary for us to analyze the numbers in order to get a better understanding of what impact this could have on an IDS and its detection capabilities. An IDS installed in a high activity network can process thousands of HTTP requests per minute. The issue of an IDS’s effectiveness has previously been addressed by Axelsson in 2000 [31] and is still a relevant topic today. We can learn more about the performance in terms of the false detection rate by analyzing how the base-rate fallacy affects the machine learning system.

In order to understand the concept of base-rate fallacy we will first present an example:

A doctor is meeting up with a patient after having performed medical tests for a fatal disease. The doctor informs the patient that he has tested positive for the disease and that the test is accurate 95% of the time. Initially, one might think that there is now a 95% chance of actual having the disease. However, the doctor also has good news about the result and informs that the prevalence for the disease is low. The reason is because out of a population of 100 000 persons, only 1 person will have the actual disease. The chance of having the disease is
therefore not 95%. How can the patient calculate his actual percentage of having the rare disease? Let us look closer on our own research problem and find out how we can apply the same logic to our own thesis.

Base-rate fallacy is based on Bayesian probability and is related to the Bayes' Theorem which we described earlier in our thesis regarding the Naive Bayes algorithm in section 4.5.3. The Bayes’ Theorem equation is expressed as the following:

\[ p(x|y) = \frac{p(y|x)p(x)}{p(y)} \]  

(6.1)

The traditionally equation of Bayes’ Theorem does however not properly take the concept of base rates into the account. In practical examples, as we have for our thesis, the base rates are needed [30]. Instead of having p(x) and p(y) denote prior probabilities in the equation then we can denote two additional parameters a(x) and a(y) as base rate parameters. With the new parameters we can adjust the Bayes Theorem to be more specific:

\[ p(x|y) = \frac{a(x)p(y|x)}{a(y)} \]  

(6.2)

An issue is that equation 6.2 hides the fact that a(y) is a marginal base rate (MBR) that needs to be expressed as a function of the base rate a(x) [30]:

\[ p(x|y) = \frac{a(x)p(y|x)}{a(x)p(y|x) + a(\neg x)p(y|\neg x)} \]  

(6.3)

By expressing MBR in the Bayes’ Theorem then we can have the theorem distinguish between priors base rates and posterior probabilities. This is what is defined as abductive reasoning [30]. Abduction is a term used in subjective logic for reasoning in the opposite direction to the available conditionals.
6.3 Base-rate fallacy of exploit kit events

We need to consider the base-rate fallacy of exploit kit events to analyze how it could potentially affect our case and system. The base rates of our classification problem can be calculated by investigating the logs on IDSs and count the number of occurrences between exploit kit requests and non-exploit kit requests. We decided to choose an IDS at random to use as an example. After an IDS was chosen we proceed to restrict the timeframe of the search to be within two weeks. From the available logs during the chosen timeframe we wanted to find the total amount of exploit kit requests and the total amount of non-exploit kit requests. The given numbers are the approximate values as we have decided to simplify them slightly:

<table>
<thead>
<tr>
<th>2 weeks of stored HTTP logs</th>
<th>Non-exploit kit data</th>
<th>Exploit kit data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>60 000 000</td>
<td>50</td>
</tr>
</tbody>
</table>

It is important to note that the values presented here are not static. The values depend on dynamic circumstances such as internal users’ outbound network traffic activity and attackers external exploit kit activity. However, we want to use the values here as an example for a randomly chosen network and to present how we can expect the machine learning system to perform under a certain circumstance in a set time frame. The value ratio presented here implies that for every observed exploit kit request there will be 1 200 000 non-exploit kit requests. This is a highly unbalanced ratio and will potentially have a high impact on the end result and conclusion of the machine learning system. We want to use equation 6.3 in order to calculate the likelihood of exploit kit traffic given a positive alert by the classifier. In order to do that we will make some denotations first based on our machine learning system.

- Let \( E \) and \( \neg E \) denote a sample with exploit kit and non-exploit kit activity.
- Let \( A \) and \( \neg A \) denote the presence or absence of an alert generated as of detecting an exploit kit sample.
In terms of detection rate we can then denote the following conditional probabilities:

\[ P(A \mid E) \text{ – An alert was generated by an exploit kit sample. True positive rate (TPR)} \]
\[ P(\neg A \mid \neg E) \text{ – No alert was generated by a non-exploit kit sample. True negative rate (TNR)} \]
\[ P(A \mid \neg E) \text{ – An alert was generated by a non-exploit kit sample. False positive rate (FPR)} \]
\[ P(\neg A \mid E) \text{ – No alert was generated by an exploit kit sample. False negative rate (FNR)} \]

By calculating the TPR and TNR we can measure the quality of the testing result. TPR and TNR is also known as sensitivity and specificity. The terms are defined as follow:

- **Sensitivity**: The machine learning model’s ability to avoid false negative
- **Specificity**: The machine learning model’s ability to avoid false positives

Sensitivity and specificity can be calculated by the following equations:

\[
\text{Sensitivity} = P(A \mid E) = TPR = \frac{TP}{TP + FN} = \frac{499}{(499 + 1)} = 0.998
\]
\[ (6.4) \]
\[
\text{Specificity} = P(\neg A \mid \neg E) = TNR = \frac{TN}{TN + FP} = \frac{958}{958 + 42} = 0.958
\]

For the quality measurements of the machine learning model we want the conditional probabilities sensitivity and unspecificity, where unspecificity is the complement of specificity.

\[
\text{Unspecificity} = P(A \mid \neg E) = 1 - P(\neg A \mid \neg E) = 1 - 0.958 = 0.042
\]
\[ (6.5) \]

We want the unspecificity as low as possible since it indicates the probability of a non-exploit kit generating an alert (i.e., a false positive). Even though the most important aspect from a security point of view is that the malicious requests are detected then false positives might be regarded as a trifle. However, when incident respond handlers and security analysts are hired to manually analyze all the alerts in an organization then it can become problematic. If a security analyst is to investigate hundreds of security incidents a day and over half of them are false positives then it will affect the overall quality of the investigations. Firstly, the security analyst can become exhausted and overwhelmed by the false positives. If the person has investigated ten false positive incidents in a row then the expectation can be that the next incident will also
be a false positive. If however the next incident is a true positive, but it resembles the false positives from the former incidents then it can be disregarded as another false positive. Also, an overwhelming amount of false positives will exhaust resources in form of time and attention that could have been used on actual security incidents.

The unspecificity is 4.2% which is high if we consider the base rate of exploit kit and non-exploit kit events. We want to calculate the likelihood of an exploit kit given alert, expressed as $P(E \mid A)$. We can apply equation 6.3 to calculate the conditional probability $P(E \mid A)$:

$$p(x|y) = \frac{a(x)p(y|x)}{a(x)p(y|x) + a(\neg x)p(y|\neg x)} = \frac{a(E)p(A|E)}{a(E)p(A|E) + a(\neg E)p(A|\neg E)} = \frac{1}{1200000} \times 0.998$$

Based on equation 6.7 we will change the following denotations:

\begin{align*}
   x &= E \\
   y &= A \\
   p &= P
\end{align*}

The following equation is based on 6.7 and uses the denotation we set earlier in this section:

$$P(E\mid A) = \frac{a(E)p(A|E)}{a(E)p(A|E) + a(\neg E)p(A|\neg E)}$$

By using this equation we can calculate the conditional probability:

$$P(E\mid A) = \frac{1}{1200000} \times 0.998 = 0.000198$$

From the abductive reasoning we get a value that indicates the quality of the current state of the machine learning system. The result indicate that for each alert then the confidence that it will be an actual exploit kit event is very low. This is an unfortunate value as it indicates that the amount of false positives will be high.
6.4 Estimated performance in a network deployment

The detection performance of the machine learning system would be important if it is to be complementing an IDS and used to analyze traffic in large-scale networks. Even though the TPR (sensitivity) is high and the system adapted well to detect new exploit kit samples then we need to have the base rate fallacy in mind as we discussed in the previous section.

In the example from the previous section, the unspecificity implies that for every true positive alert we will also have 50 400 false positive alerts. The false positive rate would need to be drastically lower if the machine learning system were to complement an IDS in a network deployment. Manually analyzing a high amount of alerts where the majority of them are false positives would be impractical. In order to reduce the amount of false positives we would need to enhance the system’s detection capabilities.

The current feature set show that it has a satisfactory TPR, but the FPR is too high considering the ratio and base rate fallacy of the network traffic. We have presented suggestions on how to enhance the system and to reduce the FPR in section 7.2.
7 Conclusion

The thesis has focused on implementing a machine learning system for detection of exploit kits by classification. We wanted to create a system that can be used with traditional IDS technologies and to limit the data features by what we had available in HTTP logs. The system was developed in a lightweight manner to increase performance and so it can potentially be integrated with an IDS.

We investigated the concepts of machine learning and exploit kits in two separate chapters in order to have a good understanding of the different fields of studies. We then proceed to design and develop the machine learning system. Three different machine learning algorithms were chosen: Support Vector Machine, Random Forest Classifier and Multinomial Naïve Bayes. We experienced the challenges of gathering enough relevant data for balancing the training set, and explained the methods we used to find the relevant exploit kit traffic. The data set was used to extract features to input to the machine learning models. We categorized the chosen features into two groups: lexical features and session features. Since the data we had available were limited to only lexical features in HTTP logs then we created a session handler module. The session handler allowed us to see each single HTTP request in a context-based order from the HTTP logs and to generate more features based on our analysis of exploit kits.

In order to evaluate the machine learning part of the system we planned three different stages: training, validation and testing. The training and validation were done by cross-validation techniques in order to validate the full training set. The validation had varying results, but the Random Forest Classifier showed the most promising and consistent results up to 99%. We decided to take the algorithm with the better performance of the cross-validation and to use in the test phase. The test phase was done with a separate testing set in order to see how the machine learning model would adapt to new data. The result showed that the model was able to adapt well to new exploit kit samples, but it had a considerate amount of incorrect classifications for the non-exploit kit data set.

We evaluated the impact of the testing results by discussing how it would affect the performance and amount of alerts if the system were to be integrated with an IDS.
The classification performance of the machine learning system indicates that it would impractical to use it to generate alerts in its current state. The reason is because of the base rate fallacy and that the unspecificity is too high. However, the classification performance showed good results in terms of detecting exploit kit traffic, but if it were to be used to generate alerts for IDS logs then the FPR would need to be significantly lower.

7.1 Goal fulfillment

In the introduction of our thesis we presented some research questions that form the red line for what we wanted to accomplish. In this section we will directly answer the research questions we made during the planning of the thesis work.

Q1) What are the most relevant features of recent modern exploit kits?

A1) Exploit kits follow a static chain of events in order to successfully infect a client. By identifying the different stages of the exploit kit chain we could create features based on what we observe in a network session. In addition, both the domain and URL contains features that can be identified with certain exploit kits.

Q2) How can we design a lightweight machine learning system to process, analyze and classify malicious exploit kit activity?

A2) By limiting the resources of features to only HTTP logs we were able to create a machine learning system which preprocesses, trains and tests HTTP data with high performance.

Q3) What accuracy of classification can we expect of the machine learning system with the data we have available?

A3) The accuracy depends on the machine learning algorithm that is chosen. Some algorithms work better than others for certain data sets and for our machine learning models we had an accuracy of up to 99% by cross-validating the training set. Afterwards we proceed by using a separate test set to validate the result with new data. We concluded that the model did well on adapting to new exploit kit data, but it did perform slightly unsatisfactory on adapting to new non-exploit kit data.
Q4) How effective would the machine learning system perform if it were to be integrated with an IDS?

A4) The machine learning system, in its current state, would have worked well in terms of detecting exploit kits due to low FNR. However, when it comes to the amount of false positives then it would have had a high negative impact. The FPR is too high to be applied in its current state and would have resulted in an overwhelming amount of false positive alerts. The reason is due to the ratio between each exploit kit event and non-exploit kit event, and this became clear as we considered the base rate fallacy. If the system were to be integrated with an IDS then the FPR would need to be significantly lower than what our test results showed. However, if the feature set is improved or if each new classification is cross-validated with other external resources then we could potentially create a more robust system with less false positives.

7.2 Future work

While working on this thesis we have discovered several ideas that could be of interest, but due to the limited time and scope of a Master’s project then we could not investigate everything. In this section we will list some of the interesting points that can be used for future work and improvements.

Expanding data and features

The final result of our thesis was based on a feature set that was mainly focusing on the domain, URL and session. As described in chapter 5, we experienced challenges with gathering exploit kit data. As a result of this we had to use older HTTP logs with limited amount of data fields. If all HTTP logs we had available would be of newer date then the amount data fields would have been significantly improved due to new updates of the IDS log system. By having more data fields we could have enrichen the feature set we used for the machine learning model, which could potentially improve the performance of the machine learning system’s detection rate.

The impact of the session handler showed promising results and the two session features we made were ranked best in the feature importance ranking. There could potentially be other features that can be included from a session that does not take on too much resources in terms of preprocessing. One example is by looking at the total amount of bytes in the HTTP requests,
or by comparing the different bytes for each step in an exploit kit chain. If a session that resembles an exploit kit chain has a high amount of bytes in the last request then it could potentially indicate a request for the payload. By working on the feature set and experimenting by adding more features then the detection rate could potentially be improved.

The feature set could also be expanded by external resources than what is available in an IDS. If the machine learning system were to analyze live-traffic from an IDS, then the external resource should be lightweight as well since we do not want it to throttle the performance of the IDS. Here we will list some of the external sources that can be considered in combination with the machine learning system:

1) Passive DNS

For our thesis we were considering expanding the feature set with passive DNS data. Passive DNS is a technology invented by Florian Weimer in 2004 and is used to build historic data of DNS. By saving the queries and metadata of all DNS lookups it is possible to correlate domain with associated IP addresses. For each HTTP request towards a domain we could have made a query to a passive DNS database to find metadata about when the requested domain was first and last observed. The passive DNS data also includes data about subdomains and would had been very useful for detecting exploit kits as the malicious server is often hosted on a newly created domain or subdomain. We could then for example have used the difference in time between the first seen request and last seen request to create a new feature.

2) Reputation data

Another external resource could be to cross-validate the alert with reputation data. If the requested IP or domain has earlier been associated with malicious traffic then we can consider the confidence in an alert to be of higher value. Reputation can be a good source for evaluating network events, but it has some flaws. Exploit kits servers are frequently moved to new domains and the reputation data would potentially not have any effect if no reputation exists for the newly created domain.

3) Whitelisting

The result of testing the machine learning system in our thesis showed that it did well on detecting exploit kit traffic, but it had a considerate amount of false positives in form of
incorrectly classified non-exploit kit traffic. What we observed by analyzing the content of incorrect samples was that many of them came from legitimate domains related to known entities. By building a whitelist of trusted domains then the false positives from these domains could have been ignored. The issue with this approach is that it would require a significant time effort in terms of building the whitelist and to tune the domains that were alerted as exploit kits. However, there are certain techniques that could be applied, by for example whitelisting the most used domains in a network or by using external services that provide data about the most visited domains on the Internet.

**Expand to more types of malicious traffic**

For our thesis we wanted to focus on detection of exploit kit activity. However, there are many other types of malicious network traffic that the system can be expanded with, either by expanding the detection scope to more than only exploit kit traffic or by creating a separate system that will focus on classification of another specific type of traffic. It could be of interest to see how well the machine learning system could adapt to detection of general malicious traffic compared to benign traffic.

**Deploy machine learning system with an IDS**

The system developed for this thesis was created in a lightweight manner. We did this because we wanted the system to process high amount of data at a low performance cost. It would be interesting to test whether the system can do classification of new data at a speed similar to new data that is processed in an IDS. The machine learning system we created for this thesis was constructed to train and test data for each time it was executed. Training takes time and would not be practical to do on an IDS since it requires processing power. There are ways to save the trained model and distribute it to multiple IDSs, but by doing this it would be necessary to update the model with new data regularly so the model would remain adaptable in detecting exploit kits. This could have been solved by training the model with new data at a specific time window once a week or whenever it would be necessary.

We believe that the system we created would be interesting and useful to deploy with an IDS. It does potentially have different use cases in terms of either detecting exploit kits by itself, or it could also possibly be used as a validation of other IDS detection mechanism. However, the current state of performance that we presented in chapter 5 shows that the system needs to
enhanced in order to be effective in a real-traffic scenario due to high FPR, but with the suggestions of adding light external resources such as passive DNS and adding more quality features based on more data then we believe the system could potentially be a good complement to many modern IDSs.
References


[9] Check Point Threat Intelligence & Research, Check Point Software Technologies Ltd. 2016. «Inside Nuclear’s Core: Analyzing the Nuclear Exploit Kit Infrastructure – Part I»


[27] N. Biasini. 2016. «Connecting the Dots Reveals Crimeware Shake-up», URL:


Appendix A - List of Acronyms

AI - Artificial Intelligence
CMS - Content Management System
CVE - Common Vulnerabilities and Exposure
CVSS - Common Vulnerability Scoring System
DGA - Domain Generation Algorithm
DNS - Domain Name System
GUI - Graphical User Interface
HTML - Hypertext Markup Language
HTTP - Hypertext Transfer Protocol
IDS - Intrusion Detection System
IPS - Intrusion Preventive System
JS - JavaScript
MaaS - Malware as a Service
PPC - Pay-Per-Click
SaaS - Software as a Service
SOC - Security Operation Center
TLD - Top-Level Domain
URL - Uniform Resource Locator
Appendix B – Session handler

The following code was used for the session module:

```
# Aggregates session events and returns a string array which contains tags
# based on the session
# Returns index array with appended tokens
def aggregate_session_events(filename):
    tmpTokens = []
    tmpIndex[]
    cnt = 0

    # Tokenize events and store them with tag identifier <domain + user agent + src IP>
    with gzip.open(filename, 'r') as f:
        for line in f:
            tmpArr = line.split(" [**] ")
            domain = tmpArr[0].split(" ")[1]
            userAgent = tmpArr[2]
            referrer = tmpArr[3]

            tmp_ip = tmpArr[8]
            ipAddress = tmp_ip.split(" -> ")
            src = ipAddress[0]
            src_ip = src.split(".")[0]

            domref = domain + " " + referrer
            token = domain + " " + userAgent + " " + src_ip

            tmpIndex.append(domref)
            tmpTokens.append(token)

    cntTokens = []
    cntIndex = cntSession = 0

    # Loop for identifying the session requests and appending tags for array index
    # Ignore sessions towards a domain with 6 or more requests (threshold)
    for ele in tmpTokens:
        if tmpTokens.count(ele) < 6:
            tmpPath = ""
            for f in tmpTokens:
                if ele in f:
                    if cntIndex is not cntSession:
                        tmpPath = tmpPath + "," + str(cntIndex)
                    cntIndex = cntIndex + 1
                    cntIndex = 0
                    cntSession = cntSession + 1
                    cntTokens.append("1" + tmpPath)
                else:
                    cntTokens.append("0")
        cnt = cnt + 1

    cnt = cntSession = tmpRedirectFlag = 0
    tmpSessionLabel = "1"
```
# Loop for checking session redirect from third-party domain
for f in cntTokens:
    if '1' in f[:2]:
        indexes = f.split(",")[1:]
        for ele in indexes:
            for ele2 in indexes:
                if ele is not ele2:
                    if tmpIndex[int(ele)].split(" ")[0] not in tmpIndex[int(ele2)].split(" ")[1]:
                        tmpRedirectFlag = 1
                        cntSession = cntSession + 1
                        if tmpRedirectFlag is 1:
                            tmpSessionLabel = tmpSessionLabel + ",extRedirect"
                            cntTokens[cnt] = tmpSessionLabel
                            tmpSessionLabel = "1"
                            tmpRedirectFlag = 0
                    cnt = cnt + 1

# Return array containing session tags for each request
return cntTokens