University of Piraeus

Department of Digital Systems

Postgraduate Program
«Techno-economic Management & Security of Digital Systems»

Master's Thesis

Classification of Exploit Kits

Tsiampa Paraskevi
MTE1540, vivian.tsiampa@gmail.com

Under the supervision of:

Dr. Christoforos Dadoyan, dadoyan@unipi.gr

Piraeus 2015-2017

This Thesis is dedicated to my beloved husband and son and my parents for their encouragement, patience and faith in me. It is also dedicated to all working women who try to combine motherhood with passion for science.
Special thanks to Dr Dadoyan for his guidance.

Contents

Table of Figures

Table of tables

Abstract

In this Thesis we examine all possible solutions presented by researchers across the world in the field of threat intelligence and more specifically in the direction of identifying exploit kits in the field. For every solution presented there is a brief report among with our point of view for each one. Moreover, we proceed with the implementation of a deep learning algorithm in Windows 10 environment, using Anaconda IDE. The goal is to create an algorithm able to identify pcap files that include exploit kit activity. We present our results and we conclude with future works goals.

1. Exploit Kits Background

Exploit Kits are software that consists of two parts. The client-side part and the server side part which is the control panel of the EK. Upon visiting a compromised link or a webpage that was originally developed for malicious cause, the user is redirected to the exploit kit's landing page. The first step of every exploit kit is to fingerprint the user's environment e.g. software, version and type of web browser, plugins etc so that it checks whether it meets a known vulnerability. Then, according to the vulnerability found, it chooses the best option of exploit to be delivered. If the exploitation is successful the malware is downloaded and executed on the victim's system. This process is usually not triggered on user's consent and that is why this type of attack meets the term "drive-by-download" attack. The malware communicates with the administration interface providing functionalities such as remote access. It is worth mentioning that, usually, the developer of the exploit kit software is different from the developer of the malicious code. [1]

All the process of compromising a user's system is fully automated in a single interface that allows people with few technical skills to become a hacker. This launching platform that can be used to deliver all type of malware e.g. a backdoor, trojan, spyware or other type of malware is user friendly and it is a characteristic on which exploit kit authors and distributors are competing for customers. The better and more automated the interface, the greater the interest from potential buyers.

Exploit kits are also considered to extend geographical boundaries, since they can be developed in Country A, be sold in Country B and used in Country C, while the servers are located in Country D. This seems to be a difficulty for security defenders, since every country has its own laws about privacy and law enforcement.[2]

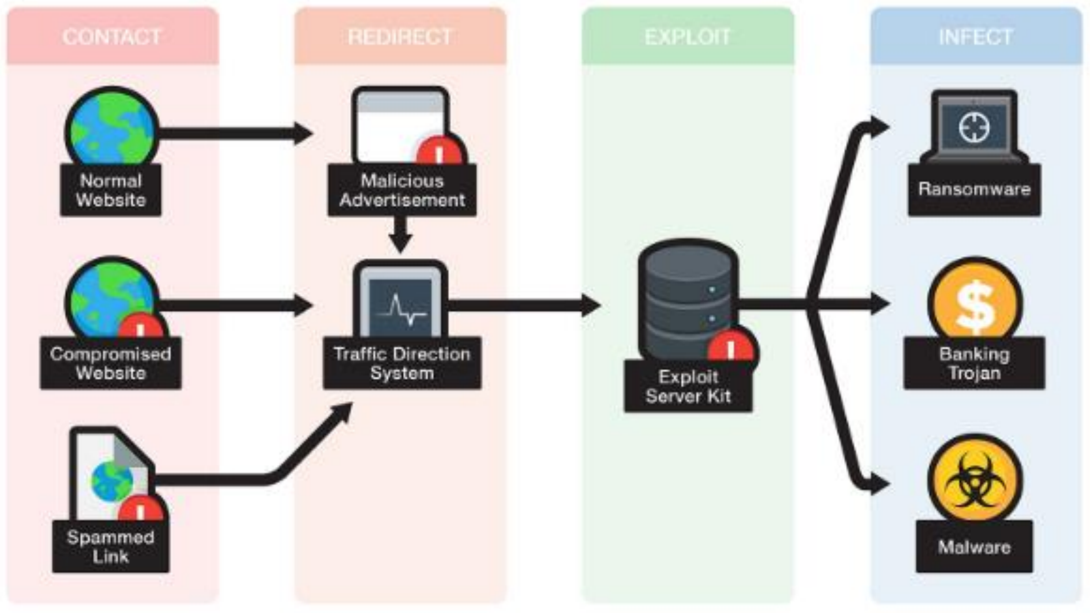


Figure 1: Stages of Exploit kit infection (Trend Micro)

Related work analysis

In this section, the goal is to present all techniques applied from researchers all over the world that target to the identification of exploit kits, either statically or dynamically.
These techniques are divided in those who apply machine learning technology and those who do not.

Machine Learning Techniques

Kizzle : A signature Compiler for detecting Exploit Kits [3]

Kizzle is an interesting approach on defensive security, introduced by Ben Stock and is practically a helpful tool for AV analysts since it aims at compiling signatures for exploit kits. Its development initiated the observation that the core part of Eks evolves much slower than the outer part of them, due to the reuse of the core code of exploit kits, after the consolidation in malware production.

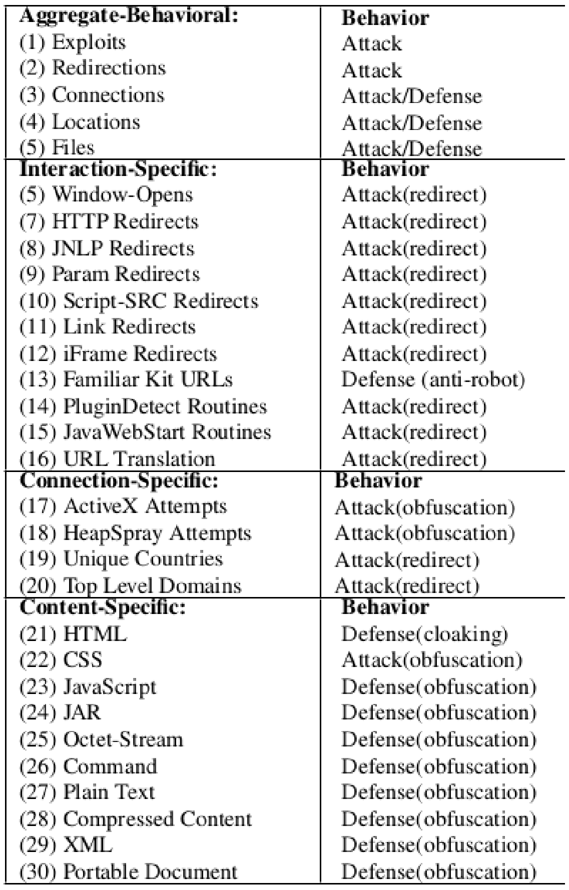
More specifically, the exploit kits consist of an outer part, the unpacker and the eval trigger and the core part, the plugin and AV detector and the payload. Since the exploit kits incorporate a range of CVEs, researchers studied the evolution of the parts of EKs and agreed to the following: EKs change their unpacker frequently, while they more seldom append new exploits or borrow code from other kits, which consist the inner part of the kit – AV detector and payload.

Kizzle compiles new signatures for kits' detection in the following manner. The machine takes as input sets of new samples along with existing malware samples which are in fact HTML pages with their inline script elements. These inputs are tokenized and the tokens are clustered by a clustering (machine learning) algorithm, DBSCAN. Then winnowing (a technique to detect plagiarism in code) is used to label the clusters as malicious of benign comparing their winnowing histogram to the one of known malware samples. The final step for the tool is to create the signature. For that cause a common subsequence of tokens is found among packed samples of a malicious cluster at first, while afterwards the algorithm determines the remaining strings found in the different samples unique for each of them. Finally a regular expression is generated to capture all matching token samples, including the common part, which was observed on the first step, and this is the final signature.

Kizzle was evaluated against 4 more prevailing exploit kits and was compared by a modern antivirus solution. The results showed that false negative rates of Kizzle – under 5% - are smaller than those for AV, while its false positive rates are under 0.03%. The weak part of the solution is the possibility of drastic change of malwares in a very small period of time e.g. in a few hours.

WebWinnow: Leveraging Exploit Kit Workflows to Detect Malicious [4]

Webwinnow is introduced by Esthete to the same purpose with the distinguished characteristic that it uses 4 different classifiers to eliminate false positives, while its implementation is based on preleminaryanalysis of the attack – centric and self-defense behavior of exploit kits. The attack centric behavior consists of client profiling obfuscation and chain of redirections while the self-defense behavior includes IP Blocking, blacklist lookup by the exploit kit, signature evasion, code protection, blocking of robots.txt and cloaking. Analyzing the above characteristics, the researchers concluded to four categories of features: aggregate – behavioral, interaction- specific, connection – specific, content specific. These features are listed below:



Aggregate-Behavioral:	**Behavior**
(1) Exploits	Attack
(2) Redirections	Attack
(3) Connections	Attack/Defense
(4) Locations	Attack/Defense
(5) Files	Attack/Defense
Interaction-Specific:	**Behavior**
(5) Window-Opens	Attack(redirect)
(7) HTTP Redirects	Attack(redirect)
(8) JNLP Redirects	Attack(redirect)
(9) Param Redirects	Attack(redirect)
(10) Script-SRC Redirects	Attack(redirect)
(11) Link Redirects	Attack(redirect)
(12) iFrame Redirects	Attack(redirect)
(13) Familiar Kit URLs	Defense (anti-robot)
(14) PluginDetect Routines	Attack(redirect)
(15) JavaWebStart Routines	Attack(redirect)
(16) URL Translation	Attack(redirect)
Connection-Specific:	**Behavior**
(17) ActiveX Attempts	Attack(obfuscation)
(18) HeapSpray Attempts	Attack(obfuscation)
(19) Unique Countries	Attack(redirect)
(20) Top Level Domains	Attack(redirect)
Content-Specific:	**Behavior**
(21) HTML	Defense(cloaking)
(22) CSS	Attack(obfuscation)
(23) JavaScript	Defense(obfuscation)
(24) JAR	Defense(obfuscation)
(25) Octet-Stream	Defense(obfuscation)
(26) Command	Defense(obfuscation)
(27) Plain Text	Defense(obfuscation)
(28) Compressed Content	Defense(obfuscation)
(29) XML	Defense(obfuscation)
(30) Portable Document	Defense(obfuscation)

Figure 2: Categories of features for Webwinnow solution

The training dataset of the algorithm derives from the log collection of 4 Thug honeyclients that probe locally installed exploit kits and two virtual honeyclients that probe live exploit kits on the web. Also a honeyclient probes benign URLs to capture their log activity also. The exploit kit samples are verified and the 30 features of exploit kits are extracted, while the samples are labelled as EK for those derived from locally installed and live exploit kits and NEK (non EK) for those provided by the benign URLs. The above mentioned features are then used to train WEKA, a machine learning suite consisted of five learning algorithms.

The detection phase follows, where an unknown URL is put on the test. A pre – filtering engine decides if the URL matches popular exploit kits and if not, a honeyclient probes it, gathering its features and provides them to the classifier, which concludes to whether it is malicious or benign.

The results of Webwinnow are quite impressive since the J48 Decision Tree and Bayes Network classifier have 100% correctly classified URLs and 0% false classified URLS ratio.

This particular method shows high accuracy and is important that it trains 5 different classifiers.

On the assumption that exploit kit traffic, though, consists of many redirections, Taylor's experiments show that e.g. Blackhole, Nuclear, Fiesta, Goon DotkaChef, Fake, and Sweet Orange consistently had a single indirection to the exploit kit server.

2.1.3 ZOZZLE : Fast and Precise In-Browser JavaScript Malware Detection [5]

Zozzle is a another solution introduced by Curtsinger that promises the static detection of malware pages and is integrated in the browser's Javascript engine.

Also here, machine learning is used and more specifically the datasets are gathered using the Nozzle heap spraying detector, through the implementation of a web crawler used to gather the potentially malicious URLs. Zozzle requires deobfuscation of the javascript code embedded in the URLs and for that purpose a Nozzle and Zozzle deobfuscator is augmented in the Javascript engine of the browser, which lead to multiple fragments of javascript code, named *javascript contexts*. The next step is to label the samples as malicious. An exploit needs to be unpacked to run its code. Practically this action occurs when the eval function is used or when new code is included in an <iframe> or <script> tag and in order to detect this code behavior, the Detours binary instrumentation library was required to intercept the calls to the compile function in the Js Engine. All the javascript contexts were manually examined to provide the malicious dataset for training the algorithm.

On the other hand, the benign training dataset of javascript contexts was created using the Zozzle deobfuscator on the Alexa.com top 50 URLs.

Through the above mentioned process the javascript fragments are labelled and the feature extraction follows to determine which type of fragment is related to malicious or benign content. The researchers extracted specific features of the Javascript's Abstract Syntax Tree (AST) such as expressions and variable declarations which were either fed to the x^2 algorithm to select the ones that are most predictive for malicious or benign intent or were hand-picked based on experience and intuition. The dataset derived from this process is the training dataset of the algorithm.

Zozzle is implemented with the naive Bayesian classifier, which repeatedly applies the rule of conditional probability. The classification of a javascript fragment is performed in linear time and depends on the the size of the code fragment's AST, the number of features that are examined and the number of possible labels.

In contrast to the Nozzle detection method, which observes the codes behavior, this method seems to be more precise, since it fragments the code to a deeper level for eliminating the false positives. The evaluation of the method shows that automatic selection of the features concludes to almost zero false positive ratio, compared to handpicked selection. The false negative rates, however, indicate that the hand-picked selection outperforms the automated one.

2.1.4 Cujo: Efficient Detection and Prevention of Drive By Download Attacks [6]

Cujo by Rieck which is a shortcut for “Classification of Unknown JavaScript Code” is a detection method embedded on a web proxy and its methodology is divided in three sections: JavaScript analysis, feature generation and learn-based detection.

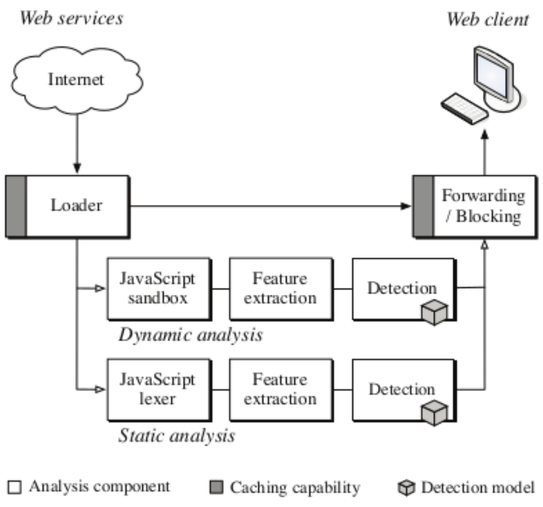


Figure 3: Overview of Cujo

The JavaScript analysis is either static or dynamic. In both static and dynamic analysis the code of the HTML or XML page is inspected for occurrences of JavaScript code which means the code inside HTML tag <*script*> or event handlers such as *onmouseover* or *onload and* also the external code referenced in the web page e.g. iframes, scripts. These pieces of code, different for each web page are then analyzed dynamically or statically.

In terms of static JavaScript analysis each web page’s code is subdued to lexical analysis. This means that the JavaScript code is divided in tokens according to the following manner similar to YACC grammar: the JavaScript consists of identifiers, punctuators, key marks, literals. The identifier is replaced by a generic token e.g. ID, the numbers by the token NUM and the strings by the token STR.01 if the string has up to 10 characters, or STR.02 if the string has up to 100 characters etc. With this technique it is easier to trace the actual function of an obfuscated JavaScript code.

Dynamic analysis makes use of AD Sandbox, a JavaScript sandbox which takes the JavaScript code and executes it in the JavaScript interpreter SpiderMonkey (Mozilla). The result is a report that contains all operations that are performed during the execution of the web pages java script code.

The second step of this method is the feature extraction. Because of the diversity between the statically and dynamically analyzed reports the Q-gram method is applied. According to this method, both types of reports are separated into fixed length parts (q-grams e.g. 3-grams if the token of every part consists of three elements) and the result for each report is a set of q-grams. Each report is processed with q-gram normalization where for each report the most important q-grams are stored for the final step.

The final step is the learn-based detection where the technique of Support Vector Machines (SVM) is applied. The SVM takes vectors of two classes as training data and creates a hyperplane with a margin, above of which the malicious pages are noted in the space vector and below of this the legitimate once. At the detection stage a function f(X) is applied at a report and a positive outcome classifies the web page associated with it as malicious or benign otherwise.

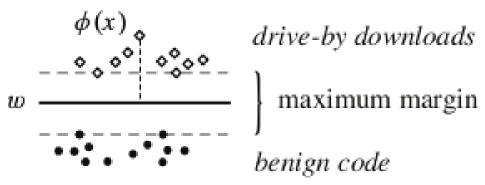


Figure 4: SVM technique margin creation

Cujo was compared against two antivirus tools, ClamAV[26] and AntiVir[27] where it demonstrated higher true positive rates – 90.2% for static and 86% for dynamic analysis – while ClamAV reported 35% and AntiVir 70% true positive ratio. False positive ratio for this method is near 0.001%, which is respectively low.

2.1.5 Exploit Kit Website Detection Using HTTP Proxy Logs [7]

Nikolaev presents a simple way of detecting exploit kit presence by using http proxy logs. The idea behind the related work is to build a global detection model in two stages of algorithm, by relying on five indicators :

- the MIME type of the served files to the browser will mostly be the ones an exploit kit would use to deliver a malicious payload after profiling the client such as Adobe Flash, pdf, Java, while other MIME types e.g. CSS, Font, will be absent.
- Number of redirects to the malicious suspected website provided by the Referrer and Location HTTP field
- Small duration of communication
- Repetitive similar requests due to the absence of keep-alive sessions found in benign sites
- Browser user – agent requests require a type of browser e.g. Firefox, because exploit kits target on existing web browsers.

The first stage of the algorithm gives an output of a cluster of hosts suspected to be compromised, while the second stage deals with the communication of users to the suspected hosts. The method consists also of a small final step, where a regex pattern is applied at the output of the second stage of the algorithm. This regex pattern is trained to describe most common exploit kits.

In order to train the classifier, a large number of pcap files was used, provided by Malware Traffic Analysis Website, which they indicate communication with exploit kits. The classifier was trained upon these samples after their convertion to http logs. The precision of the algorithm was measured upon real network traffic from various enterprises. After the detection algorithm provided a result, this was manually tested against Virus Total, blacklists, Public feeds to confirm the accuracy of the result.

Although this method is simple it gives a general idea of detecting exploit kits activity

This method presents an idea of detecting exploit kits' activity through the simplicity of http proxy logs. However, there are many evasion techniques for attackers e.g. mimicking MIME types of benign websites that make the above project difficult to implement.

2.1.6 Shady Paths: Leveraging Surfing Crowds to Detect Malicious Web Pages [8]

SpiderWeb is a detection method for malicious URLs proposed ny Stringhini (2013) with takes into consideration HTTP redirections to specify if a webpage is malicious or not. This is accompliced by constructing new entities called redirection chains which each specify a particular webpage and sequentially aggregate to redirection graphs, a cluster of redirection chains with similarities on the page level.

The next step is to classify a redirection graph as malicious or not. This classification takes into consideration specific parameters:

1. Client features. The user population that followed the redirection chain, because a big diversity by accounting a large number of countries visited the website suggest the potential presence of an exploit kit, since legitimate webpages usually have a low metric on this field.

2. Referrer Features. The less number of referrer pages point to a particular URL, the more the chances it is a malicious one, since legitimate pages are advertised by a wide number of other benign sites.

3. Landing pages features. Legitimate landing pages differ little from the referrer

4. Final page features. Benign redirection graphs contain many GET parameters, as a means to provide more services to their visitors, while a large number of distinct URLs as a final page suggests the same URL generation from attackers through the kits.

5. Top level domain features due to the different cost of registration of a new domain among top level domains.

6. Redirection graph features e.g. long redirection chains which show many redirections, which is a characteristic of a malicious page, redirection to same domains – feature of legitimate pages, presence of a hub etc

The above features were used to train the classifier, provided by WEKA machine learning tool, which was based on the Support Vector Machines(SVM) , trained with Sequential Minimal Optimization. Upon determination of redirection graph as malicious by the SVM, SpiderWeb marks all final pages that belong to this graph as malicious.

The datasets for training the algorithm were provided by a popular Antivirus Vendor.

The results suggest that the redirection graph features that classify a page as malicious are the country ratio, the presence of hub in the graph, the IP as a final page while legitimate pages are distinguished by the country ration (small), the redirections to same domain feature and the top level domain characteristic.

SpiderWeb has some limitations e.g it requires a large number of graphs for an accurate classification, it fails to group URLs that e.g have different web page for each request to the same graph and finally at same cases the attacker redirects the victim to a benign URL after compromising, which whitelists the particular redirection chain, while it should not.

Also, Taylor has proved over experiments that some exploit kits e.g Blackhole redirect directly to the exploit kit server.

2.1.7 Prophiler: A Fast Filter for the Large-Scale Detection of Malicious Web Pages [9]

Prophiler demonstrates a method for large scale detection of malicious web pages, based also in machine learning techniques.

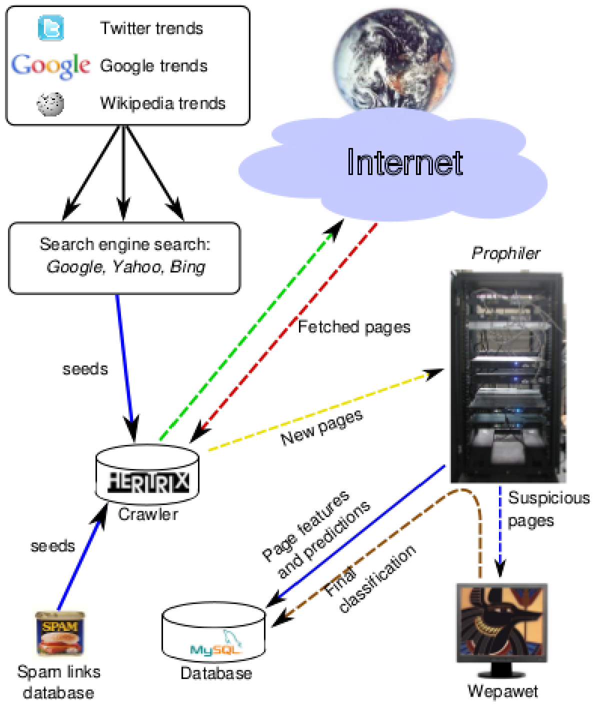


Figure 5: Prophiler architecture

Prophiler’ s architecture is composed by a modified instance of Heritrix, a web crawler, whose referrer header is set to the search engine to deceive the attackers that the compromised page was visited after a search result. The web crawler visits a list of URLs provided by three search engines (Twitter, Google and Wikipedia) after a search for trending terms to capture pages that are a part of a SEO campaign and a list of URLs derived from a feed of spam mails.
The outcome of the web crawler is the input to the Prophiler. Prophiler includes three models that analyze every web page provided to them, by extracting specific features according to their role.
The HTML features. The features selected for extraction are 19 in total and are based on statistical information and structural information. The pages are parsed with Neko HTML Parser and HTML parser. Such selected features are: number of elements with small area, since attackers try to hide elements that lead to exploitation, number of included URLs, number of misplaced elements as a sign of SQLI, presence of double documents e.g. head, title as a side-effect for exploitation and others.

The Javascript features. A total of 25 features are selected to be extracted which are related to exploitation behavior. Some of these are: presence of shellcode, number of direct assignments which is typical for malicious scripts because of deobfuscation and decryption, number of iframe strings or suspicious strings e.g. “evil”, “crypt” etc
URL and host-based features. In this case 33 features are collected and they are either syntactical, or DNS- based, or whois-based. Some of them are: the presence of an IP in the URL, the short TTL of the DNS query since malicious pages do not live long and change domains frequently, recent registration date as a sign of potential exploit server and others.
Then three models, in correspondence to each feature set, are trained with the training dataset using machine learning techniques and more precisely the WEKA machine learning platform. The training dataset is a list of URLs both benign and malicious provided by Wepawet’s Database and Alexas 100 sites. Among the existing algorithms in this platform, the Random Forest classifier presents better results for HTML features, the J48 for the javascript features and the J48 for the URL and host-based features.

After the classification process of the URLs the malicious results are sent to dynamic analysis tools to prove if Prophiler’s outcome was correct or false. The dynamic analysis tool used was WEpawet.
After the training process, Porphiler is validated against a new dataset, named the validation dataset, to evaluate its effectiveness. On this dataset, also provided by Wepawet, Prophiler exhibited a false positive rate of 10.4% while the false negative rate rose to 0.54% only. By discarding also many benign pages as a result of correct classification, Prophiler saved resources from the dynamic analysis tool.
Results from a large scale evaluation aka almost 19000000 pages within 1 month show that Prophiler saved 400 days of processing from the dynamic analysis tool by classifying only 14.3% of them as malicious, while feeding the malicious outcome list to wepawet this method demonstrates a false positive rate of 13,7% which is relatively high in comparison to other detection approaches. In addition false negative rate – pages that are evaluated to be benign while in fact are malicious – rises only to 0,01%.

2.1.8 Detecting Malicious HTTP Redirections Using Trees of User Browsing Activity [10]

Mekky introduces the building of an HTTP redirection tree for each user to identify malicious behavior. The analysis is based on the fact that the adversaries often use a large number of redirections as a measure of evading their activity.

A browser activity tree is a single user’s activity during one session that as calculated lasts 30 secs, in which the parent node is connected with an edge to a child node when the later is a result of an HTTP response to the first. Also every edge is labelled with a specific type e.g. HTML meta tag redirection (<meta http-equiv=’refresh’ url=’http://www.domain.com”), referrer header, <iframe>, <link> tags etc. depending on the method that triggered the visit to the child node.

When the 30sec period ends, the new URL us considered as a new root for the tree, while when a URL is not included to any previous HTTPs respondes, a new tree is also created.

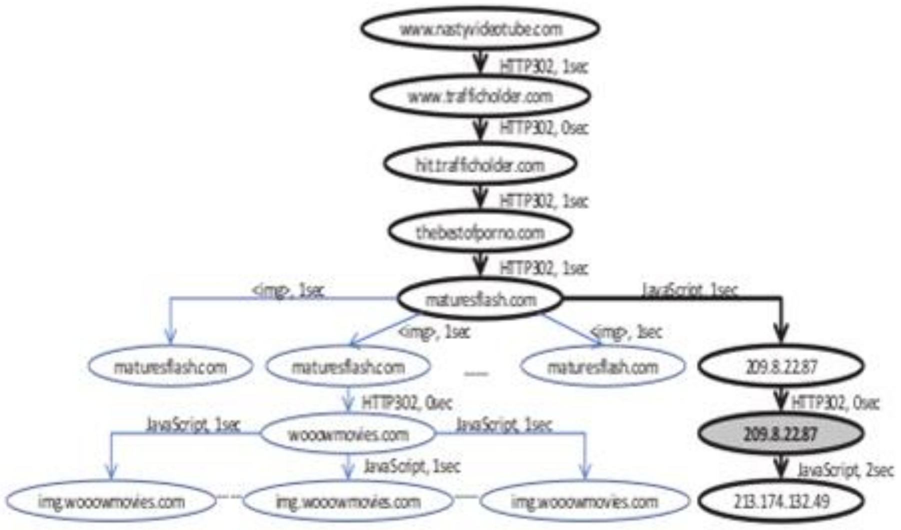


Figure 6: Browsing Activity Tree. The path with thicker edges is malicious

Two datasets of 24 hour each of network activity provided by a large ISP was provided for the implementation of this method, one from April2011 and one day from August2012. To establish a ground truth on the browser activity algorithm, an add-on on Firefox was created by the researchers that documents the path of URLs a user visits. Then, firefox was instructed to visit both benign and malicious URLs and the results of the add-on was compared to the one's coming from the browser activity algorithm after applied to the 24h traffic dump. Results confirm the accuracy of the algorithm.

To distinguish the legitimate redirection paths of each tree from the malicious ones, these paths are compared to the analysis' results of a known IDS solution. Whenever a malicious URL is found in a path according to the IDS, that path is malicious.

Furthermore, by making a few observations, the researchers conclude the fact that malicious browsing activity is related to: large number of visited domains and large number of redirections. Due to this they agree to specific features to train the J48 decision tree classifier used in this method: number of 3xx HTTP redirections, number of different domains, number of different domain HTTP 3xx redirections, number of consecutive HTTP 3xx redirections, number of consecutive different domain HTTP 3xx redirections, number of short edges, path length and edge duration.

This methods prototype consists of an HTTP parser, which provides the HTTP header and payload to the URL extractor, that extracts the URLs from the above and labels the type of the source e.g and <iframe> and then the results are sent to the activity tree updater that constructs the activity tree. Finally, for each path a feature vector according the above mentioned features is built to send it to the J48 classifier in order to determine whether it is a malicious or benign path.

To train and evaluate the classifier two different methods where used. The 10-fold cross validation, where the data are split into 10 sets and the classifier is trained with the first 9 and next evaluated at the 10th and the 60% percentage split where tha data is ramdomly split into two subsets, 60% for training purpose and 40% for evaluating.

After evaluation against Google safe Browsing that this method demonstrates better results identifying suspicious activity. Moreover by analysing the false positives generated by this method, according to the IDS's suggestions, after further analysis they are found to be potentially malicious, which is not detected by the IDS.

Although this approach is very interesting it lacks the live detection scenario since it is implementation is based in statical analysis.

2.1.9 Revolver: An automated approach to the detection of evasive web-based malware [11]

Due to the fact that nowadays attackers have implemented techniques that evade potential detection of their malicious code, Revolver seems to address to this problem and present a new project to this cause.

Revolver is based on AST analysis of the javascript code to find similarities between pairs of javascript code in order to conclude whether an evasion technique is used or not. This way it addresses to the problem of obfuscation of malicious code performed by adversaries.

An Oracle is primarly used - Wepawet honeyclient in this case - to deliver lists of malicious and benign webpages. Then Revolver executes each page using a browser emulator based on HtmlUnit which extracts all Javascript code of the page and the AST of the webpage is saved for analysis. In case of "code in a code" with the use e.g of the eval() function, Revolver saves the relation between the parent and child code, while it records whether each code related to a node is finally executed. Then depending of the outcome of the Oracle, Revolver lists this AST as malicious or legitimate.

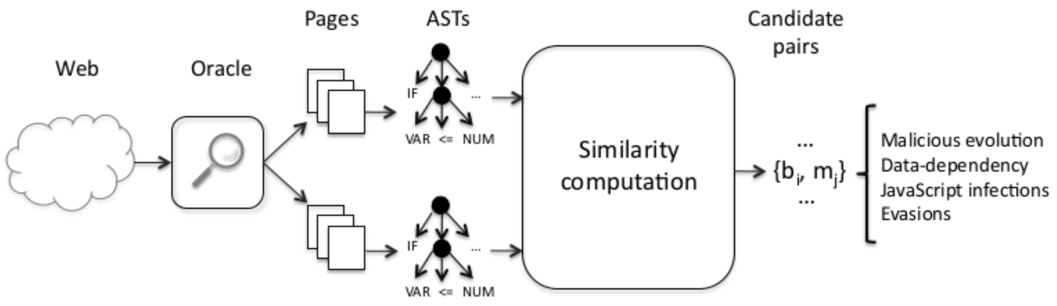


Figure 7: Revolver architecture

The next step is to transform the AST of each code to a normalized node sequence which is the sequence of nodes when one traverses the tree pre-orderly.

Similarity search between node sequences is later performed. In order to minimize the effort of similarity search because of the vast amount of node sequences, deduplication on the dataset of the node sequences is essential. Moreover, Revolver groups the node sequences to sequence summaries and for each sequence summary it provides its malicious sequence summaries. The laters derive from the application of the k-nearest neighbor search algorithm. This addresses to the same problem, meaning it lowers the computation effort.

Finding the similarity between two sequence summaries evolves comparing the normalized node that belongs to a summary to the sequence summary of the malicious neighbours, by use of pattern matching algorithm by Ratcliff (*) that is based on defining a threshold score when exceeding it two sequences are similar.

The outcome is hundreds of pairs of similar scripts. Revolver classifies each pair in the following manner: if the Oracle has concluded that both of them are malicious, then the similarity is a particular classification category, e.g. an evolution of the code. If the Oracle concludes that one is malicious and the other is benign, this pair is a candidate pair and stored for further analysis.

The categories are data-dependency category (packers that are used both for benign and malicious purposes), control flow, java script injection, evasions and general evolution cases.

Control Flow: This candidate pair is examined for control flow node differences. Control flow nodes (try, catch, while, if, else) may suggest the presence of an evasion technique applied by the adversaries. It depends whether the control flow is added in the benign or the malicious script a candidate pair might be classified as injection or evasion. The execution bit is taken into consideration in this instance since it shows whether the code was executed or not to transform the malicious code into a benign.

JavaScript injection: it concerns injecting malicious code to benign java script libraries to avoid detection. The malicious code has control flow nodes that turn the benign script to malicious.

Evasions: control flow scripts are included in benign code are considered to be evasions. The script was modified to benign while the Oracle has classified it as malicious

General evolution cases include all pairs that do not match to the above categories.

Revolver, despite its success to find evasions has also some limitations. At first it needs an Oracle to provide the verdict upon a URL since it does not perform URL classification and to have sample codes. Secondly if the attackers write a script from scratch this method will not be effective on finding similarities since the new script will not have a similar one. Then since the classification into categories is performed on pairs of code, the method is effective when the primary and the evasive code are given. If one of them is missing, Revolver fails. Finally, when cloaking is used by adversaries Revolver cannot perform correctly.

2.2 Non - machine learning techniques

2.2.1 AFFAF: A New Technique for Counteracting Web Browser Exploits [12]

Byungho Min and his team in the “New technique or Counteracting Web BrowserExploits” introduces an interesting way to protect browsers and browser plugins from exploits kits with a tool called AFFAF (a fake for a fake).
Due to the fact that the primary action of an exploit kit is reconnaissance, which translates to identifying the victim environment Min suggests that tricking this process by the exploit kit’s side would lead to the failure of the exploit attempt. AFFAF is a methodology in which the browser’s or plugin’s version is advertised as higher than it actually is and the reason for that lies in the following observation by the author: Both web developers and attackers take into consideration of the browser or plugin version for developing their code. Web developers set the lowest version to their code, that a web application needs to run, while the attackers set the maximum version in exploit kit coding, from which and below a plugin/ browser can be exploited. Therefore, advertising a higher version both exploit kits fail, while users can enjoy java applications in web sites that check the minimum required version.
AFFAF is a proactive defensive method for exploit kits, targeting to the protection of browsers and plugins, it requires no configuration on the user’s side and protects without disabling any version the client uses. AFFAF has been checked against 50 known exploit kits and a locally developed one and demonstrates according to the researchers zero false positive ratio.

2.2.2 Next Generation Of Exploit Kit Detection By Building Simulated Obfuscators [13]

Another characteristic of exploit kits is that they insert obfuscated code to the compromised web pages to avoid detection. This observation Is the base to structure a different detection approach from Palo alto network associates.

Researchers Luo and Jin focused their investigation on the rebuild of the obfuscator which was used on samples of malicious payload, as an easier way to de-obfuscate a web page’s contents and reveal whether it is benign or not.

Their sample data contained more than 20.000 samples over a 2-year period and the first step to this task was Javascript normalization, by a Javascript lexer, which parses the source code and tokenizes into a sequence of tokens. Then It assigns each type of token to a unique character. At the end of this procedure, the unique number of source code samples decreases.
In order to specify whether a set of sample data come from the same obfuscator variant or version, hierarchical clustering algorithm was used, which results a number of clusters with sample data of the same obfuscator origin. The threshold – the distance among the clusters – was applied after manual observation of few samples.

The final step of the project is the rebuild of the obfuscator. By comparing the common structure of the samples in the same cluster, the researchers rebuild the template of the obfuscator, while rebuilding the logic of the software requires to reverse engineering the encoder and applying the same at the reconstruction of the obfuscator.

2.2.3 Defending Browsers against Drive-by Downloads: Mitigating Heap-spraying Code Injection Attacks [14]

Heap spraying code injections are a type of an attack where the attacker creates multiple instances of a shellcode combined with NOP sledge at the area of the heap memory of a java program e.g. a browser plugin where is the most probable place of the memory that a variable is allocated. Obviously with this method the attacker inserts malicious arguments to the java program.
The idea of this detection approach lies exactly at this attacker's behavior: the monitoring of string (variable) allocation and the presence of shellcode in it. This is implemented using SpiderMonkey, Firefox's java script engine in the following manner:
First, it is essential to track the memory space where strings are allocated, because a shellcode needs sub sequential memory spaces to be put and the string variables are perfect for this goal. To define which part of the memory is allocated for string variables, the researchers added code to SpiderMonkey (open source) that defines the start of a string variable and the length of it.
The next step is to check for parts of shellcode in the above mentioned memory parts. In order to identify shellcode libemu - a C library - is used that offers basic x86 emulation and shellcode detection. By applying a minimum threshold of 32 bytes as a minimum length of shellcode sequence, libemu checks whether there is a valid sequence of instructions in this memory area.
For optimization reasons, data to be freed by the garbage collector of java are excluded from the list of objects to be emulated by libemu, since it will not contain shellcode. Also, strings that result from concatenation and come out to be clean, shows that there is no need to emulate the parts they come from. Both techniques reduce the number of memory parts need to be inspected.
This method was evaluated for false positive rates by checking many benign URLs from Alexa ranking and demonstrated no false positive at all and it also was evaluated to specify detection ratio which was approx. 94%.
The method focuses in heap spraying attacks while there are many more drive by download attack types such as the use of visual Basic scripts and others that do not make use of memory exploits, which limits its success ratio.

2.2.4 Detecting Malicious Exploit Kits using Tree-based Similarity Searches [15]

Taylor's approach on the detection of exploit kit traffic lies on the building of web session trees. According to this every HTTP traffic is converted to web session trees, aka http web requests originating from a single root request e.g. visiting a website over a time window. The web session tree is built by applying the root request and underneath the nodes are mapped to the attributes related with the primary request for example an image that loads to the website.

For the building of web session trees all HTTP flows are grouped, comparing specific attributes such as same referrer and location and in the specified time frame.

The Malware index is a group of web session trees that derive from converting recorded HTTP traffic from a honey client that visits malicious sites, to web session trees.

After transforming the HTTP flow into a tree, its features are extracted and indexed. These are content based (node level) indexed features and structure based.

The first ones consist of three types: token features, URL structural features and content based features. Token features of each node derive from breaking down the URL into its parts: domain, path, etc. URL structural features are an abstract categorization of URLs depending on the data types it has, e.g. numeric, alphanumerical, base-64 encoded etc. Content based features are associated with the HTTP headers or payloads and include content length, types (e.g. Accept) and response codes.

Structure based features are strings which consist of a number of substrings as many as the nodes the tree has, since every node of a tree has a particular identifier that includes the identifier of the tree it belongs to. The string is built by visiting each node in preorder traversal and appending its identifier at the end of the string.

In order to put the method to the test, HTTP traffic from an enterprise network is monitored and web session trees are built so as to compare them with the malware index described above.
This comparison includes two steps: node level similarity search and structural similarity search.
Node level similarity search is based on comparing the token, URL and content based features of the tested session tree against the same features of the malware indexed nodes. For this purpose the Jaccard Index and the weighted Jaccard indexed similarity approach is applied were a score derives that shows whether the two nodes are relevant or not.
Structural similarity search follows to determine whether the suspected “malicious” nodes are connected the way the malware index suggests, meaning are part of the same subtree as in the malware index, by using the tree edit distance * and finally concluding whether the tested HTTP traffic is related to an exploit kit, since every indexed tree is labeled with one exploit kit.

This method has been compared with Snort IDS solution and the results show that it outperforms Snort since it demonstrates far less false positives, due to the application of structure similarity search.

2.2.5 BLADE: An Attack-Agnostic Approach for Preventing Drive-By Malware Infections [16]

Blade (Block All Drive-by-download Exploits) is a different detection technique proposed by Lu that focuses on the prevention of unconsented content execution in browser level. That means that it monitors the IU interface and when a download process is initiated without the user’s approval through dialog box, the downloaded content is transferred to supervised secure disk space.

The above are implemented according to the following principals

- Real time user authorization capture and interpretation: monitoring users consent for downloading an object real time
- Robust correlation between authorization and download content: correlation of the users approval for downloading the specific object with the expected binary stream downloaded by the browser
- Enforcement of execution prevention: prevention of downloading files without authorization from users
- Browser agnostic enforcement: The browser’s rest processes are not affected by BLADE’s intervention

Blade architecture consists of the Screen Parser, the Supervisor, the Hardware Event Tracer, the Correlator, the I/O Redirector.

The Screen Parser is responsible for capturing the user’s consent upon downloading an object through e.g. a browser’s dialog box. This component is triggered only if a dialog box appears regarding downloading a file and this is succeeded by installing an agent in browser to pre-filter irrelevant windowing events.

Once the Screen parser is triggered, the Supervisor, which is the responsible component for organizing the rest of BLADE's components notifies the H/W Event Tracer, that looks for hardware device signals invoking user's authorization for the downloading process is triggered ready to monitor a potential user's authorization e.g. by clicking the OK button on the dialog box. At the same time the Correlator is triggered to start the recording process by the Supervisor.
The Correlator provides 1-1 mapping between download authorizations and downloaded files. This process starts when a file that has been written to browser's local storage and the same file – according to its name – that is of pending authorization. The Correlator starts the validating process which includes the matching of the content of the candidate file from a specified source IP upon authorization with the content of the download candidate that exists in a logged TCP stream from the same source address. When there is a match the file is validated and the correlation stage is completed.
The I/O Redirector is the component which stores the correlation candidate file upon signal of the Supervisor to the secure zone of the disk. This zone restricts execution of files which are not allowed by users. After a successful correlation the file is transferred to the filesystem.

BLADE was successful on blocking almost 8000 drive by downloaded attacks while it produced zero false alarms and malicious files were stored to the quarantined zone. These malicious files and especially their binaries were detected by virustotal.com only by 28.43%. BLADE was also checked for false positives by attempting to download trusted applications from legitimate sites and it demonstrated zero false positive ratio.

However this technique demonstrates a few limitations:

- It does not prevent social engineering attacks, where the victim is lured to download the malicious file
- It does not prevent memory execution scripts e.g. inserted by JavaScript code
- If the download consent User Interface is disabled by user on purpose or accidentally BLADE cannot prevent the aforementioned attacks.
- BLADE does not prevent from interpreted scripts, only from binaries.

2.2.6 EVILSEED: A Guided Approach to Finding Malicious Web Pages[17]

Invernizzi presents Evilseed an approach for detecting malicious web pages, which is based on a particular number of URLs provided as seed to this architecture and after a certain process the machine provides an larger list of URLs that are suspected to be malicious based on the performance of certain core elements, named gadgets while at the same time, it leverages search engine results.
The system's architecture consists of the seed, the prefilter, the Oracle and the Gadgets.

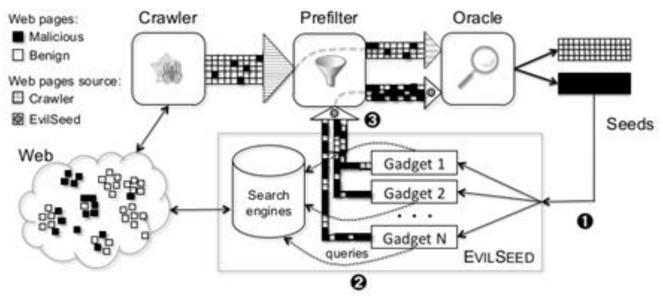


Figure 8: EVILSEED overview

The seed is the primary list of URLs that are considered to be malicious and are fed to the machine. This list can be obtained by a traditional crawler. The results are sent to the prefilter

that makes an initial distinction between benign and malicious web pages from the primary list and then the malicious – considered ones are sent to the Oracle. The Oracle consists of three components: Google Safe Browsing, Wepawet (not supported currently) and a custom built detector for pages that host fake AV.
The malicious URLs provided by Oracle are sent to the gadgets. The gadgets are the core part of the system. The are a cluster of tools that query search engines and afterwards they provide the extracted information back to the Oracle. A circle of events is initiated this way and at each loop the malicious list of URLs grows.
The Gadgets are the following:
1. The Links gadget. Its purpose is to locate malware hubs, meaning pages that contain several links to malicious pages, whether they are compromised sites or pages that list malware links. After the seed input as described above, the gadget deploys queries using the link operator to Google, Yacy and Bing and extracts the outgoing links of the returned pages, only to send them to the Oracle.

2. The Content Dorks Gadget. This gadget leverages the features of a google dork. A google dork is a list of terms that are likely to be found into vulnerable web pages, e.g. “powered by PhpBB 2.0.15” which is a vulnerable bulletin board software. This gadget is interested only in landing pages from the initial seed because they provide more indexable content and they remain live longer. For reason web pages that are not longer active are excluded from the primary seed and secondly a number of evaluated fields of HTML content that is related to malicious URLs such as script code after an html tag is taking into consideration for this distinction.
The custom google dork this gadget provides to the Oracle derive from term extraction and n-gram extraction process. The term extraction process leverages the Yahoo’s Term Extraction API and the n-gram extraction process provides sequences of n grams of words (n=2 up to 5) ranking the grams comparing them to their similarity with terms appearing on a malicious web site.

3. Search Engine Optimization gadget. A SEO campaign is the process of the producing multiple compromised web pages by the adversaries, by a php script that fetches popular search terms from search engines and additional features such as images combines them together. This gadget needs at least one URL that is a part of such campaign and apply cloaking detection heuristic. By accessing the page using three different ways (as a Google indexing bot, as a user who was redirected there after a search query and as a user who typed it to the browsers address bar) cloaking is detected when it redirects to 2 or more different domains. After cloaking detection a candidate list is formed by domains that can be extracted by this page as a part of the same SEO campaign, links that are included to it after visiting it as a bot and pages that were optimized for the same search terms. The candidate list is forwarded to the Oracle.

4. Domain registration Gadget. It extracts all domains from the initial seed and flags the web pages that are registered before and after the malicious. It creates a candidate list and sends it for further Analysis to the Oracle.

5. DNS Queries Gadget. DNS traces expose the connections of compromised pages that contain links to malicious URLs. By monitoring the traffic using a sensor in front of a RDNS server, the gadget scans for DNS queries to domains included in the evil seed. Once such query is detected a candidate list is created by URLs that are queried by the same client at a preset time window. This list is considered to have malicious suspected URLs and is sent to the Oracle for investigation.

EVILSEED was compared with a web crawler and web searches with specific strategies to find malicious web pages and the results show that it outperforms them. It finds approximately 200 new malicious detections per day and 25% of the queries sent to the search engines lead to a compromised or landing page.

However, it is essential for the presence of a SEO campaign in order the SEO gadget to perform and also, regarding the DNS gadget, the attackers may easily evade this technique by redirecting to the landing page after a time delay. In addition, caching and pinning of DNS responses by modern browsers may compromise the results and the effectivenesses of this gadget and the whole project does not apply in other attack vector e.g. spam mails.

2.2.7 EKHUNTER: A Counter-Offensive Toolkit for Exploit Kit Infiltration [18]

EKHUNTER is a radical approach for mitigating the risk of exploit kits by attacking them, proposed by Esthete. This method was proposed after research on 30 available exploit kits in the lab. In a similar way as adversaries, the researchers performed vulnerability assessment on them to exploit them.

This method consists of three steps: exploit kit detection, exploit kit identification, exploit execution.

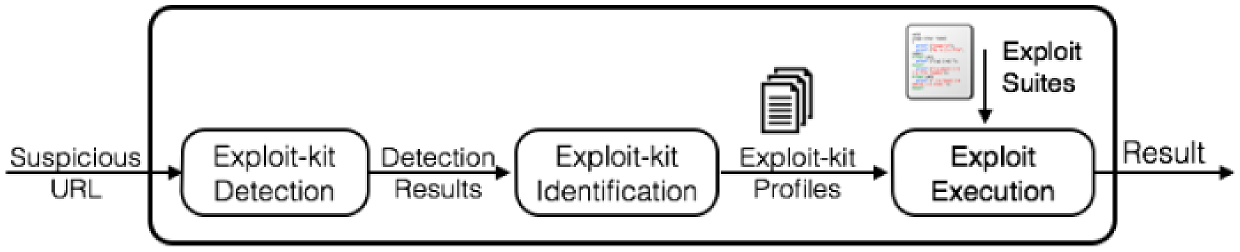


Figure 9: EKHUNTER architecture

In the first phase of exploit kit detection EKHUNTER is being provided by a set or URLs that are expected to be malicious. This set is either derived by Webwinnow or by publicly available lists of compromised URLs. The tool analyses every URL to conclude whether it points to exploit kits.
In the second phase, EKHunter performs identification of the EK family the exploit kit belongs to. This is achieved by either structural information about the exploit kit, gathered through the analysis process of the 30 exploit kits, or behavioral information provided by Webwinnow.
The third part consists of the vulnerability assessment and the payload delivery to the server of the exploit kit.
The vulnerability assessment focuses on these vectors Detection of Access Control Vulnerabilities, which detect the path constraints that lead to a successful privileged escalation attack, Detection of SQLI vulnerabilities, which provide the path constraints and an analysis of the DB of the exploit kit, and detection of multiple vulnerabilities using RIPS[28].
The above findings are used in the phase of the exploit generation. More specifically, by enumerating the above constraints, Z3-str [29] a string solver is used to return a successful exploit string. This exploit string will be used in the forming of the HTTP request towards the exploit kit server. At this point the cyber analyst has a number of exploits he can leverage tailored to the specified target.
EKHunter has achieved to compromise the Eleonore, the FirePack, the Fragus and the Exploit Kit exploit kits.
The above proposed method is unsuccessful when the source code in the server side of the exploit kit is not available, since it focuses on its compromise. Also, when the source code is not written in PHP is an issue, because the implementation is based on this feature. Finally since this approach is about an offensive countermeasure, ethical issues arise since it may violate end users licence aggrements, when it concerns blackhat software.

2.2.8 Pexy: The other side of Exploit Kits [19]

De Maio attempts to contribute to the security analyst’s main cause by analysing the server side code of selected exploit kits, after their source code is available to researchers due to shutting down of the machines hosting them by law authorities.
Pexy is the result of this contributions and it aims to milk all possible exploits by an exploit kit, depending on the different input parameters by the victims to produce s single exploit kit signature, that can be latteraly used by e.g. honeyclients or other defensive tools.
The architecture of the system consists of the data flow analysis of the server side code (Pixy) and the extended behavioral analysis.

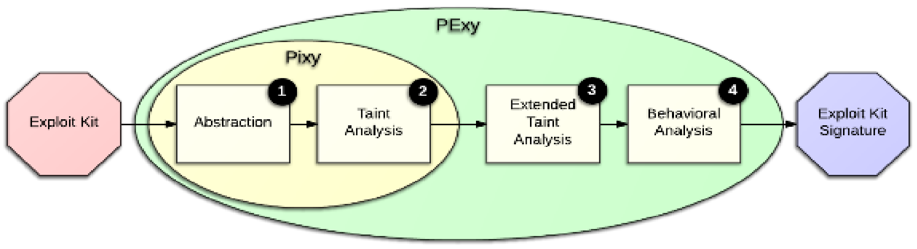


Figure 10: Pexy architecture

The first phase of the analysis targets to discover flows and tain style vulnerabilities. To achieve that, a abstract tree is built upon the php form that represents the input of the platform. Each function that is encountered is described as control flow graph, that indicates its inputs and actions and the whole php form is considered as a Parse Tree.
The taint analysis focuses on entry points from the user e.g. HTTP headers, sensitive sinks, meaning routines that return data to the browser such as print() and sanitization routines that destroy potentially malicious characters e.g. htmlspecialchars(). Vulnerability is considered to be a path of the above mentioned parse tree that begins from an entry point, concludes to a sensitive sink and there is no sanitization process performed during this flow. The result of this step is a number of the condition sink of the program with the relative taint information.

The next phase is the extended taint analysis, where the results from taint analysis are further analyzed to improve precision of the method. The number of data flows produced by phase I is subjected to finding indirect taints, through the observation that a tainted variable may affect another variable that may not seem tainted at first place. The output of this step is a set of Sensitive sink encountered with the indirect taint information whose outcome depends on the user’s request parameters.
The branch classification follows to identify the behavior of the exploit kits code depending on the input parameters applied to the sensitive sink set described above. Each sensitive sink flow is considered to be a conditional branch, because it depends on the user’s input. In order to accomplish this, four behavior elements are documented: embedded PHP code, printing statements, file inclusion, header manipulation. The branch classification phase returns the correlation of each behavior element found on this page to a set of conditional branches.
Next step is to extract the type, name and value of HTTP parameter for every conditional branch that satisfies its condition (behavior element action). The final step is to discover the exact value of the HTTP parameter that satisfies the condition and therefore it is possible to retrieve the required string that applies to the tainted HTTP parameters.

The output of the method is a string signature that can be used to provoke an exploit kit to reveal its presence since the above mentioned extracted string will satisfy the conditions built for this cause.

The limitation of this approach is that it requires a primary inspection of the server-side source code of the exploit kit which can only be obtained by law enforcement. Moreover, it does not investigate the client side of the exploit kit as previous mentioned approaches.

2.2.9 Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code [20]

Cova introduces a tool named JSAND (JavaScript Anomaly-based aNalysis and Detection) which focuses in analyzing specific features of potentially malicious JavaScript code, similar with fore-mentioned techniques.
To support the robustness of the tool, the researchers claim that they focus on characteristics that imitate the life cycle of an exploit kit operation and according to this idea, it is more likely to detect this type of operation when two of three of the following vulnerability classes are observed: plugin memory violations, unsafe APIs, browser memory violations. The below 10 features are noticed in these classes and are the following:

- Redirection and cloaking related
  1. Number of target of redirections: an unusual large number of target domains indicates malicious activity.
  2. Browser personality and history-based differences: visiting each page twice by changing the client visiting the URL while retaining the same IP leads to observing any anomalous behavior in terms of network or the instantiation of plugins
- Deobfuscation
  3. Ratio of string definitions and string uses: the high number of variable definition is related to evasion through obfuscation technique
  4. Number of dynamic code executions: a large number of function calls to interpret JS code and code changes is connected to the previous feature so that the input of the variables is interpreted.
  5. Length of dynamically evaluated code: obfuscation techniques are noticed when assigning to the eval function a large string, which is usually encoded shellcode.
- Environment Preparation
  6. Number of bytes allocated through string operations: assignments that lead to a memory allocation larger than 100MB are a common characteristic of exploit kit behavior.
  7. Number of likely shellcode strings: the existence of shellcode is identified by parsing the javascript code and finding strings that exceed a certain threshold (256 bytes)
- Exploitation
  8. Number of instantiated components: when many different components are triggered it is more likely that an exploit kit tests different vulnerabilities.
  9. Values of attributes and parameters in method calls: large number of characters in strings passing as parameters in method calls strongly suggest the presence of shellcode.
  10. Sequence of method calls: a common sequence of methods that indicate malicious activity is downloading and later executing malicious code. These types of sequential methods are monitored.

The proposed solution is implemented based on libAnomaly, a library to develop anomaly detection systems and uses the following models:

- Token finder: determines whether the values of certain features are limited in an expected range since in legitimate scripts certain features have few possible values, while discovering a different unexpected value may indicate malicious activity.
- String length and Occurrence accounting: this model is responsible for counting the string length – since large strings are suspected for malicious operations – and the possible number of redirections
- Character distribution: strings in Javascript code are meant to be read by humans and the characters are originated from a well-defined set e.g. ASCII while the presence of non-printable code used for binary encoding suggests a potential attack.
- Type Learner: this model was added to the original libAnomaly library and is used for determine whether the value of a feature is of the expected type, e.g. a plugin method expects a URL as input.

Additionally, HtmlUnit was used as a framework to emulate browser environment and it was configured so as to extract the above mentioned features.
The system's analysis consists of two steps: exploit classification and signature generation. The exploit classification has two phases: In the first phase, JSAND analyses the samples according to the previous models and for the ones that labels as malicious, it extracts four exploit features: the name of the plugin, the name of the method, the position of anomalous parameters, the type of identified anomaly e.g. long string value. Manual search to vulnerability repositories follows for each set of four features, so as to connect each feature set with known vulnerability and create a so called exploit class.
The next phase runs automatically and classifies each new feature set with a known – from previous step – exploit class. JSAND collects the potentially malicious events, creates the four-feature set for each of them and then a Bayesian classifier, connects the set with one of the previous found exploit classes.
Moreover, in the second step JSAND generates signatures for signature based detection tools **.**
JSAND was evaluated against known detection tools: ClamAV, an open Antivirus platform, Phoney C , a browser honeyclient and Capture-HPC a high interaction honeyclient.
One known – good dataset as called is a dataset consisted of benign URLs that was used to train the model. Four known bad datasets, which consist of malicious URLs were used to evaluate the tool and export the False negative rate while a set of two unknown datasets, which derived from pages gathered through a crawling session and pages that were submitted to Wepawet service between October and November 2009. For the last ones there is no outcome about being benign or malicious and this final dataset is used for calculating False positive rates, where the outcome of the tool is "malicious" while in fact the web page is legitimate.
Comparing JSAND with the the other three tools, concerning the False positive ratio, the researchers conclude that when tested against the known good dataset, JSAND delivers zero false positive ratio, while when tested against the unknown pages dataset, JSAND has a minimum of 0.001% of false positive ratio (15 URLs out of 115K were wrongly classified as malicious)
To determine the false negative ratio, JSAND was tested against the three known-bad datasets, as well as the other three fore mentioned tools were tested too. JSAND gives a minimum of 0.2% false negative rate while the other tools deliver much higher false negative ratio.
The JSAND tool might fail to detect a malicious page, if the attackers evade detection by using new type of attack classes that might not be found in a web repository, which is common in 0-day attacks.
Also, by fingerprinting the emulated environment of the JSAND tool, attackers may evade detection, by delivering for example a benign page instead of the actual landing page.

Below you will find two tables that compare the characteristics and the success ratios among the for mentioned suggested solutions. The success ratios derived from the papers of the authors according to their findings.

Comparison Tables

	Web browser analysis oriented	Involves Machine learning	Based on EK's behaviour Analysis	Static detection	Dynamic Detection	Code oriented analysis	Compared with existing Anti malware Solutions
AFFAF	✓				✓		
Kizzle		✓	✓	✓			✓
Webwinnow		✓	✓	✓			✓
Zozzle		✓		✓		✓	✓
Simulated Obfuscator				✓		✓	
Cujo		✓				✓	✓
Detecting Heap Spraying Attacks				✓		✓	
HTTP Proxy Logs		✓	✓				
SpiderWeb		✓		✓			
Tree Based Simirarity Searches				✓			✓
BLADE	✓				✓		
EvilSeed			✓	✓			
EKHUNTER			✓	✓			
Prophiler		✓		✓			
HTTP redirections Using Trees		✓		✓			
Pexy			✓	✓			
JSAND				✓		✓	
Revolver		✓		✓		✓	

Table 1: Classification of presented solutions

	Success Rate	False Positive rate	False negative Rate
AFFAF	100%	0%	N/A
Kizzle	N/A	<0.03%	<5%
Webwinnow	99.9%	0.001%	0.1%
Zozzle	99.48%	0.02%	9%
Simulated Obfuscator	N/A	N/A	N/A
Cujo	90%	0.001%	9.89%
Detecting Heap Spraying Attacks	93.9%	0%	6%
HTTP Proxy Logs	95%	N/A	5-8%
SpiderWeb	N/A	0-3.4%	16.1-84.78%

Tree Based Simirarity Searches	**99%**	**0%**	**0.001%**
BLADE	**100%**	**0%**	**0%**
EvilSeed	**N/A**	**N/A**	**N/A**
EKHUNTER	**N/A**	**N/A**	**N/A**
Prophiler	**N/A**	**10.4%**	**0.54%**
HTTP redirections Using Trees	**95%**	**0.1%**	**N/A**
Pexy	**N/A**	**N/A**	**N/A**
JSAND	**N/A**	**0%**	**0.2%**
Revolver	**N/A**	**N/A**	**N/A**

Table 2: Ratios of presented solutions

3. Implementation

At this section we attempt to develop a deep learning algorithm as a concept of proof of machine learning application in threat intelligence field. Before proceeding with the implemented solution a brief information on neural networks follows which is essential for correct comprehension of the algorithm.

3.1 Deep Learning Theory

The idea behind neural networks is to imitate human's brain to make such connections and correlations of data so as to provide a prediction.
The human's nerve cells called neurons are connected with other thousand cells by axons. Stimulation from external environment is accepted by dendrites, creating electric pulses which travel through the neural network. In a similar way, artificial neural networks are composed of nodes connecting and interacting with each other through links. Each node performs operations on input data according to a specific function determined by the us and passes the result to other nodes, until it meets the output node.

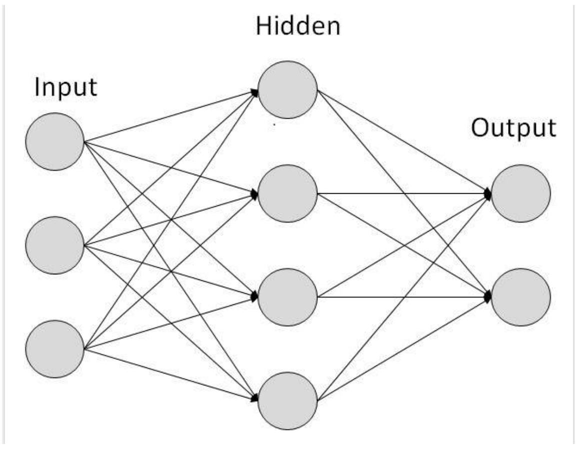


Figure 11: Neural network

Each link is associated with a weight value. The learning procedure is associated with changing of the weight's value until it meets the correct output.

There are several types of learning strategies

- Supervised learning: The teacher feeds the data and knows already the answers. The NN makes the guess, the teachers provide the correct answer and the NN makes the proper adjustments.
- Unsupervised learning: The NN searches for patterns since there is no input data with known answers. Then it divides the elements into groups according to the patterns.
- Reinforcement learning: The NN makes a guess according to the input data and if the guess is wrong the NN adjusts the weights to make a different required decision next time.

The Back propagation algorithm is the training algorithm. After a wrong observation the flow of the NN goes back from output node to input node to adjust the weights so as to produce the desired output. [21]

Below are the essential steps for building a neural network:

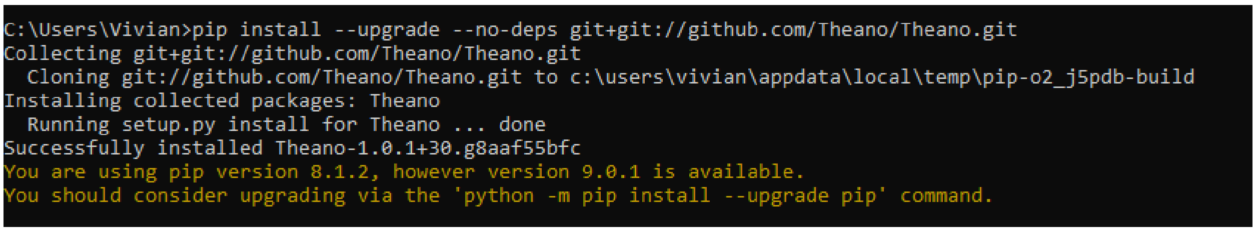
1. Initialize weights randomly
2. Input the first observation into input layer
3. Forward propagation : all neurons are activated by a specific function and the result reaches the output node
4. The output is compared to the actual result
5. Back propagation: From right to left the weights are updated
6. Repeat of forward and back propagation
7. When all training set of observations passes the NN it is an epoch. Epochs are repeated.

3.2 Installation Prerequisites

For the creation of the Neural Network it is vital to install all the essential software for it and this is the following packages: Theano, Tensorflow and Keras. We have already installed Anaconda IDE [22]
Theano and Tensorflow contain binary libraries that handle neural networks and they are considered to be highly demanding in terms of coding expertise. Keras on the other hand is a more simple package that includes the aforementioned ones. Nevertheless, it is vital to install all three. [23, 24, 25]

The installation was performed on Windows 10 operating system following the commands found in tensorflow official webpage

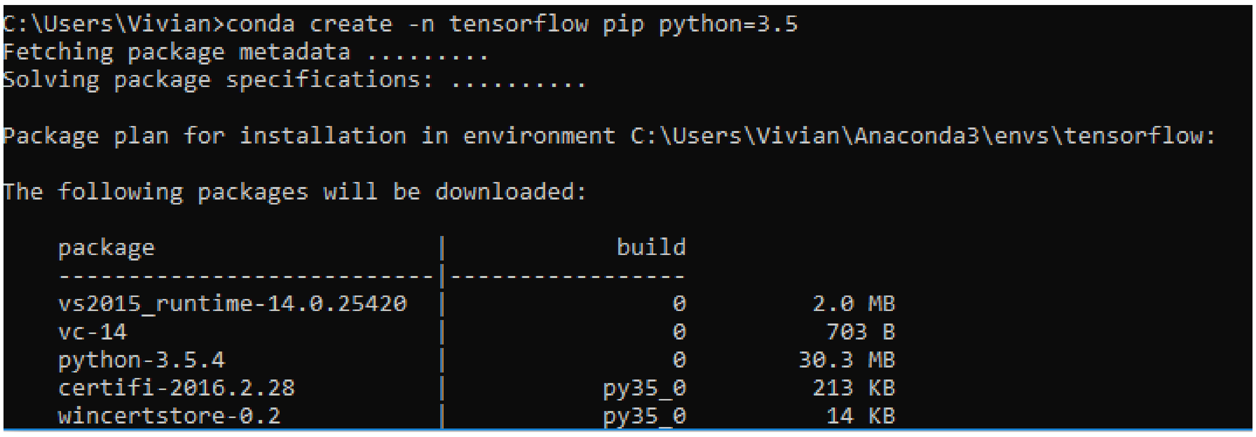
1.First we begin with the Theano installation with the following command



```
C:\Users\Vivian>pip install --upgrade --no-deps git+git://github.com/Theano/Theano.git
Collecting git+git://github.com/Theano/Theano.git
  Cloning git://github.com/Theano/Theano.git to c:\users\vivian\appdata\local\temp\pip-o2_j5pdb-build
Installing collected packages: Theano
  Running setup.py install for Theano ... done
Successfully installed Theano-1.0.1+30.g8aaf55bfc
You are using pip version 8.1.2, however version 9.0.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

Figure 12: Theano Installation

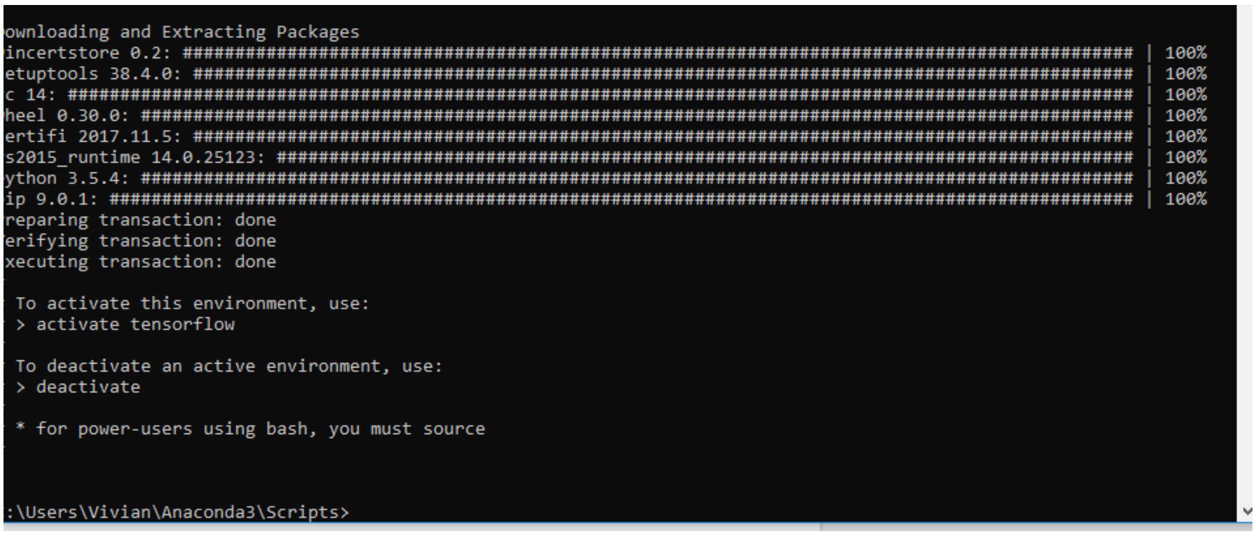
2. Next we proceed with Tensorflow,creating an enviroment for it



```
C:\Users\Vivian>conda create -n tensorflow pip python=3.5
Fetching package metadata .........
Solving package specifications: ..........

Package plan for installation in environment C:\Users\Vivian\Anaconda3\envs\tensorflow:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    vs2015_runtime-14.0.25420  |                0         2.0 MB
    vc-14                      |                0          703 B
    python-3.5.4               |                0        30.3 MB
    certifi-2016.2.28          |            py35_0         213 KB
    wincertstore-0.2           |            py35_0          14 KB
```

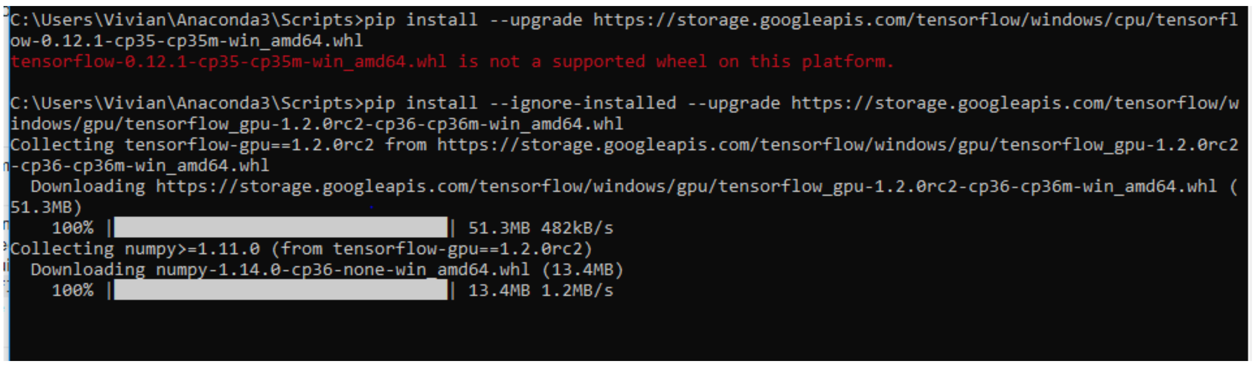
Figure 13: Tensorflow Installation

With the following result:



```
ownloading and Extracting Packages
incertstore 0.2: ################################################################################## | 100%
etuptools 38.4.0: ################################################################################# | 100%
c 14: ############################################################################################# | 100%
heel 0.30.0: ###################################################################################### | 100%
ertifi 2017.11.5: ################################################################################# | 100%
s2015_runtime 14.0.25123: ######################################################################### | 100%
ython 3.5.4: ###################################################################################### | 100%
ip 9.0.1: ######################################################################################### | 100%
reparing transaction: done
erifying transaction: done
xecuting transaction: done

 To activate this environment, use:
 > activate tensorflow

 To deactivate an active environment, use:
 > deactivate

 * for power-users using bash, you must source


:\Users\Vivian\Anaconda3\Scripts>
```

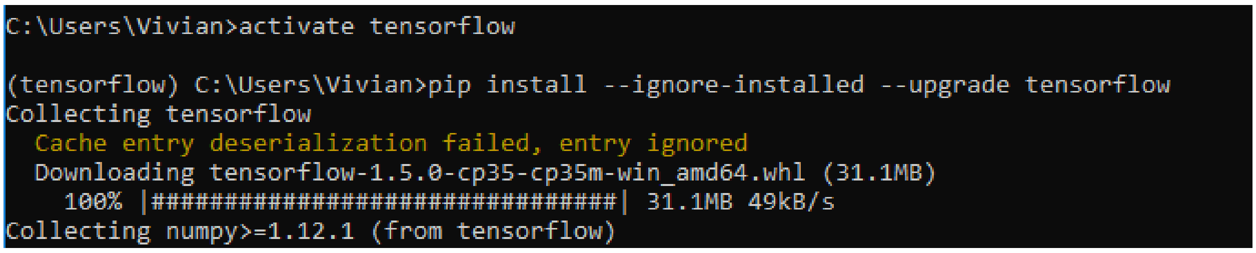
Figure 14: Tensorflow Installation



```
C:\Users\Vivian\Anaconda3\Scripts>pip install --upgrade https://storage.googleapis.com/tensorflow/windows/cpu/tensorfl
ow-0.12.1-cp35-cp35m-win_amd64.whl
tensorflow-0.12.1-cp35-cp35m-win_amd64.whl is not a supported wheel on this platform.

C:\Users\Vivian\Anaconda3\Scripts>pip install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/w
indows/gpu/tensorflow_gpu-1.2.0rc2-cp36-cp36m-win_amd64.whl
Collecting tensorflow-gpu==1.2.0rc2 from https://storage.googleapis.com/tensorflow/windows/gpu/tensorflow_gpu-1.2.0rc2
-cp36-cp36m-win_amd64.whl
  Downloading https://storage.googleapis.com/tensorflow/windows/gpu/tensorflow_gpu-1.2.0rc2-cp36-cp36m-win_amd64.whl (
51.3MB)
    100% |████████████████████████████████| 51.3MB 482kB/s
Collecting numpy>=1.11.0 (from tensorflow-gpu==1.2.0rc2)
  Downloading numpy-1.14.0-cp36-none-win_amd64.whl (13.4MB)
    100% |████████████████████████████████| 13.4MB 1.2MB/s
```

Figure 15: Tensorflow for CPU

We activate tensorflow with the below command and the prompt changes to (tensorflow)

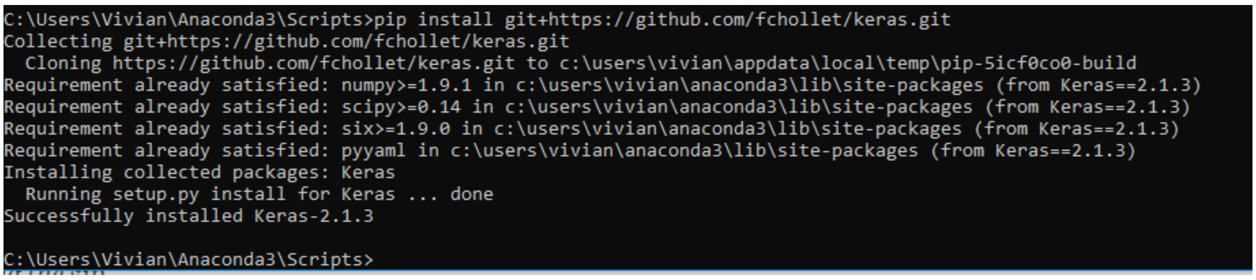


```
C:\Users\Vivian>activate tensorflow

(tensorflow) C:\Users\Vivian>pip install --ignore-installed --upgrade tensorflow
Collecting tensorflow
  Cache entry deserialization failed, entry ignored
  Downloading tensorflow-1.5.0-cp35-cp35m-win_amd64.whl (31.1MB)
    100% |################################| 31.1MB 49kB/s
Collecting numpy>=1.12.1 (from tensorflow)
```

Figure 16: Tensorflow activation

We installed the CPU installation option because we will create a simple neural network with minimum processing requirements.

3. Installing keras:



```
C:\Users\Vivian\Anaconda3\Scripts>pip install git+https://github.com/fchollet/keras.git
Collecting git+https://github.com/fchollet/keras.git
  Cloning https://github.com/fchollet/keras.git to c:\users\vivian\appdata\local\temp\pip-5icf0co0-build
Requirement already satisfied: numpy>=1.9.1 in c:\users\vivian\anaconda3\lib\site-packages (from Keras==2.1.3)
Requirement already satisfied: scipy>=0.14 in c:\users\vivian\anaconda3\lib\site-packages (from Keras==2.1.3)
Requirement already satisfied: six>=1.9.0 in c:\users\vivian\anaconda3\lib\site-packages (from Keras==2.1.3)
Requirement already satisfied: pyyaml in c:\users\vivian\anaconda3\lib\site-packages (from Keras==2.1.3)
Installing collected packages: Keras
  Running setup.py install for Keras ... done
Successfully installed Keras-2.1.3

C:\Users\Vivian\Anaconda3\Scripts>
```

Figure 17: Keras Installation

3.3 Data Preprocessing

Neural networks require a dataset to proceed with their estimations on the outcome of a problem.
In this case we processed a number of pcap files using bash shell scripting to provide a csv file that gathers the features selected for this algorithm.
As seen in the previous section, the researchers gathered specific features to create their matrixes to use the as input for the machine learning algorithm. Furthemore, they used a large number of features to conclude to a more precise estimation. Our goal is to show the method in the deep learning field of machine learning and for that cause we only choose 3 features from the pcap files: large strings, iframes that contain large strings and number of redirections. The malicious pcap files were provided by malware-analysis.net website while the legitimate once were captured on a linux machine using wireshark.

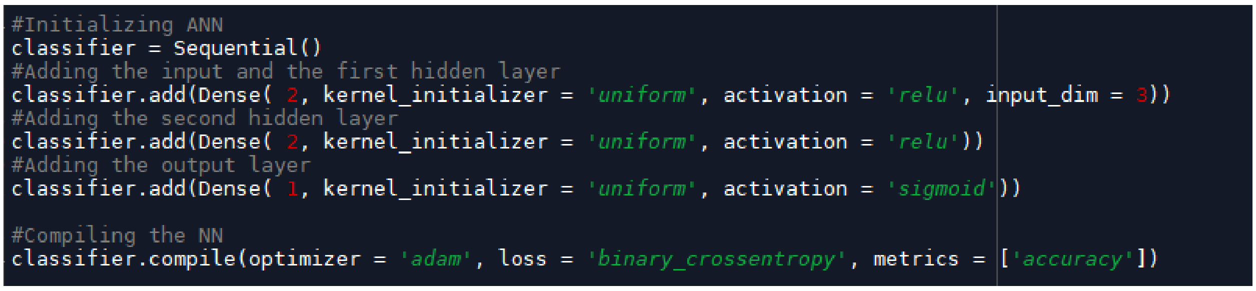
Before proceeding with designing the neural network, it is crusial to preprocess the dataset because of the fact that the deep learning algorithms are accepting numbers as input and therefore a categorical problem with output “malicious” or “benign” as described in the dataset could not be understood by the algorithm. Moreover metrics have to be in the same scale, which improves the algorithms efficiency.
In this case the independent variables (malicious - legit) are replaced by 1 and 0 accordingly.

We also needed to implement feature scaling because some values have distance from one another, so that an independent variable with greater value will not dominate another one. Also the preprocessing stage inserts the dataset to the algorithm and divides to training set and test set.

The columns of the dataset are the dependent variables, meaning the features of the pcap files, and the independent variables which is the outcome whether a pcap file host malicious or legitimate traffic.
By importing the dependent variables, it will allow the neural network to make the correct correlations and estimate the correct weights for each connection between the neurons to provide the correct prediction, by evaluating which features have the most impact.

3.4 Building the Neural Network

We choose to build the NN beginning with the input nodes which are as many as the features of each observation, 3 in our case and we choose to have hidden layers of 2 neurons in each hidden layer. Also, the rectifier function is chosen for every node in the input and hidden layer and the sigmoid activation function is preferred for the output layer, because we need to measure probability of a pcap being malicious. Activation functions are the functions applied on the input values combines with the weights of every synapsis.

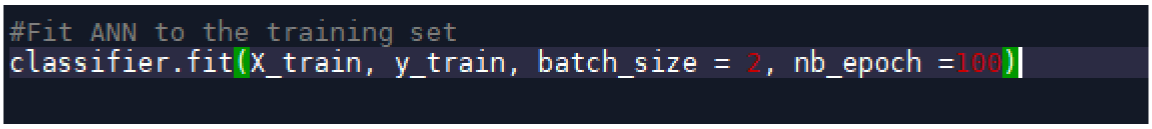


```
#Initializing ANN
classifier = Sequential()
#Adding the input and the first hidden layer
classifier.add(Dense( 2, kernel_initializer = 'uniform', activation = 'relu', input_dim = 3))
#Adding the second hidden layer
classifier.add(Dense( 2, kernel_initializer = 'uniform', activation = 'relu'))
#Adding the output layer
classifier.add(Dense( 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

#Compiling the NN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Figure 18: Building the NN

3.5 Training The Algorithm

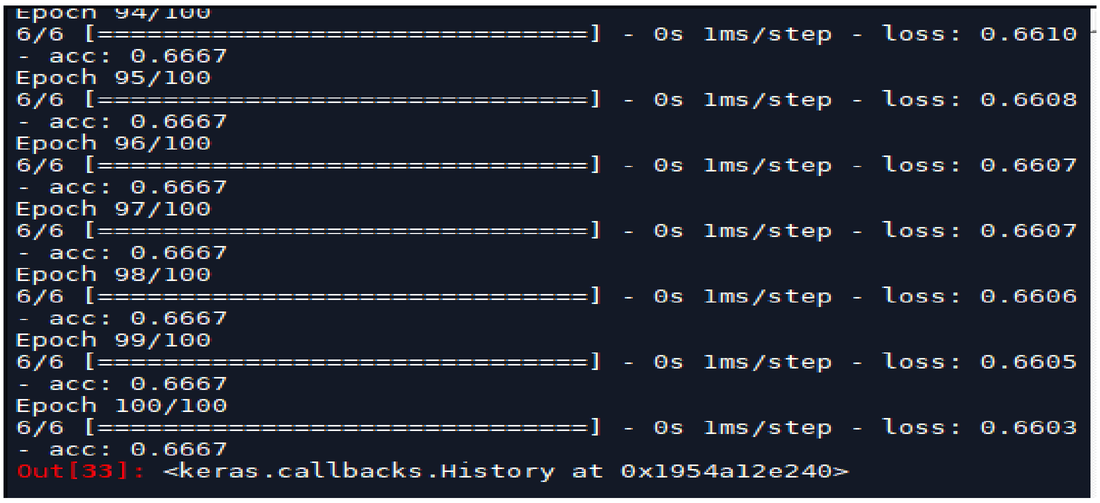
This is the most interesting step of the process because it gives immediate results and provides an accuracy metric for the algorithm.
We choose to fit in the classifier the training dataset and feed t with batches of 2 observations per time and 100 epochs. This means that the weights are reculculated after 2 observations and the whole training dataset goes through the algorithm 100 times.
The execution of this command, regardless the size of batch chosen each time, provided **66%** accuracy on the algorithm.



```
#Fit ANN to the training set
classifier.fit(X_train, y_train, batch_size = 2, nb_epoch =100)
```

Figure 19: Command for training the algorithm

Result:



```
Epoch 94/100
6/6 [==============================] - 0s 1ms/step - loss: 0.6610
- acc: 0.6667
Epoch 95/100
6/6 [==============================] - 0s 1ms/step - loss: 0.6608
- acc: 0.6667
Epoch 96/100
6/6 [==============================] - 0s 1ms/step - loss: 0.6607
- acc: 0.6667
Epoch 97/100
6/6 [==============================] - 0s 1ms/step - loss: 0.6607
- acc: 0.6667
Epoch 98/100
6/6 [==============================] - 0s 1ms/step - loss: 0.6606
- acc: 0.6667
Epoch 99/100
6/6 [==============================] - 0s 1ms/step - loss: 0.6605
- acc: 0.6667
Epoch 100/100
6/6 [==============================] - 0s 1ms/step - loss: 0.6603
- acc: 0.6667
Out[33]: <keras.callbacks.History at 0x1954a12e240>
```

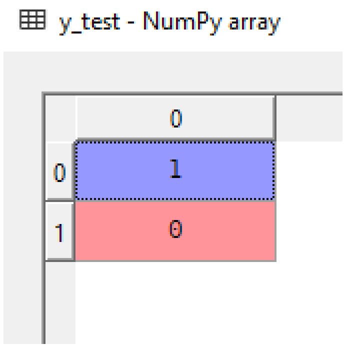
Figure 20: Output for training the algorithm

This accuracy metric, which is considered to be low, was highly expected since it is recommended to provide the algorithm a large dataset in order to obtain better results. As discussed in the beginning of the section, the implementation of the deep learning algorithm serves the purpose of presenting the methodology, which can be used with minimum alterations or future work.

3.6 Validation of the Algorithm

The validation of the algorithm is in fact the test of the algorithm on new observations.

Where 1 corresponds to malicious traffic (true) and 0 to benign traffic.

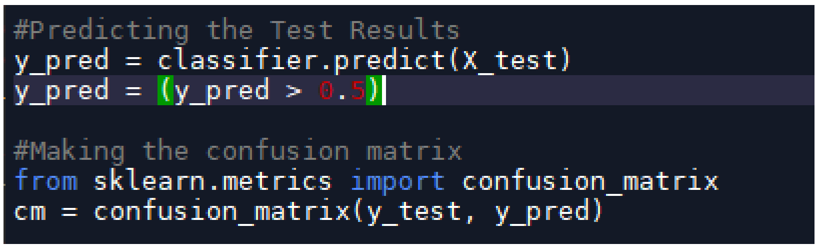


y_test - NumPy array

	0
0	1
1	0

Figure 21: y_test array in Spyder

This task will be completed with the help of the test dataset, which was defined at the beginning of our python program. We choose a threshold of 50%, above which the pcap file is predicted to be malicious.

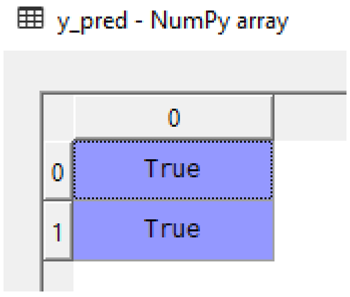


```
#Predicting the Test Results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

#Making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Figure 22: Predicting the test results

The Confusion matrix gives the accuracy of the algorithm on new observations.

The result is quite as expected since it classifies all the observations as malicious while in fact the one is malicious and the other is benign.



y_pred - NumPy array

	0
0	True
1	True

Figure 23: y_pred array

The algorithm gives a validation rate of 50% which is close to the 66% predicted accuracy.

The low accuracy rate occurs because of the following reasons:

1. Pcap files that represent legitimate traffic might as well include the same number of large strings as malicious
2. The features selected are very few.
3. The low number of samples

4. Conclusions

The above study indicates that there is a promising future on implementing deep learning algorithms in the threat intelligence field. In this thesis, we developed a simple deep learning algorithm by using few pcap files as input data as a sample but there is greater work to be done. As indicated by similar studies mentioned in this thesis, additional input data could be provided by large organizations or online free material, to be properly analyzed and fed to the algorithm. Eventually, a high accuracy algorithm can be built to recognize malicious traffic with minimum false positives.

5. References

[1] International Conference on Security and Privacy in Communication Networks, 10th International ICST Conference, SecureComm 2014, Beijing, China, September 24-26, 2014, Revised Selected Papers, Part II

[2] https://zeltser.com/what-are-exploit-kits/

[3] Stock, B., Livshits, B., and Zorn, B., 2015. “Kizzle: Asignature compiler for exploit kits”. In International
Conference on Dependable Systems and Networks.
[4] B. Eshete and V. N. Venkatakrishnan. Webwinnow: Leveraging exploit kit workflows to detect malicious urls. In ACM Conference on Data and Application Security and Privacy, 2014.
[5] Curtsinger, C., Livshits, B., Zorn, B. G., and Seifert, C., 2011. “Zozzle: Fast and precise in-browser javascript malware detection.”. In USENIX Security Symposium, pp. 33–48.

[6] K. Rieck, T. Krueger, and A. Dewald. CUJO: Efficient Detection and Prevention of Drive-by-Download Attacks. In Proceedings of the Annual Computer Security Applications Conference (ACSAC), 2010

[7] I. Nikolaev, M. Grill and V.Valeros. Exploit Kit Website Detection Using HTTP Proxy Logs Conference: Fifth International Conference on Network, Communication and Computing At: Kyoto, Japan, December 2016

[8] G. Stringhini, C. Kruegel, and G. Vigna. Shady paths: leveraging surfing crowds to detect malicious web pages. In ACM Conference on Computer and Communications Security, 2013.

[9] D. Canali, M. Cova, G. Vigna, and C. Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. In World Wide Web Conference, 2011.

[10] H. Mekky, R. Torres, Z.-L. Zhang, S. Saha, and A. Nucci. Detecting malicious http redirections using trees of user browsing activity. In IEEE Conference on Computer Communications, 2014.

[11] Alexandros Kapravelos and Yan Shoshitaishvili, University of California, Santa Barbara; Marco Cova, University of Birmingham; Christopher Kruegel and Giovanni Vigna, University of California, Santa Barbara. Revolver: An Automated Approach to the Detection of Evasive Web-based Malware Proceedings of the 22nd USENIX Security Symposium. August 14–16, 2013 • Washington, D.C., USA

[12] Byungho Min Adv. Cyber Security Res. Centre, Macquarie Univ., Sydney, NSW, Australia and Vijay Varadharajan Adv. Cyber Security Res. Centre, Macquarie Univ., Sydney, NSW, Australia. A New Technique for Counteracting Web Browser Exploits, Software Engineering Conference (ASWEC), 2014 23rd Australian

[13] Tongbo Luo and Xing Jin, Palo Alto Networks. Next Generation Of Exploit Kit Detection By Building
Simulated Obfuscators. Blackhat 2016.

[14] P. Ratanaworabhan, B. Livshits, and B. Zorn, “NOZZLE: A Defense Against Heap-spraying Code Injection Attacks,” in USENIX Security Symposium, 2009.

[15] Teryl Taylor, Xin Hu, Ting Wang, Jiyong Jang, Marc Ph. Stoecklin, Fabian Monrose and Reiner Sailer. Detecting Malicious Exploit Kits using Tree-based Similarity Searches. Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy Pages 255-266

[16] L. Lu, V. Yegneswaran, P. Porras, and W. Lee, “BLADE: An Attack-Agnostic Approach for Preventing Drive-By Malware
Infections,” in ACM Conference on Computer and Communications Security (CCS), 2010.

[17] L. Invernizzi, S. Benvenuti, M. Cova, P. M. Comparetti, C. Kruegel, and G. Vigna. Evilseed: A guided approach to finding malicious web
pages. In Proceedings of the 2012 IEEE Symposium on Security and Privacy, SP ’12, 2012.

[18] Birhanu Eshete, Abeer Alhuzali, Maliheh Monshizadeh, Phillip Porras, V.N. Venkatakrishnan, Vinod Yegneswaran EKHUNTER: A Counter-Offensive Toolkit for Exploit Kit Infiltration, NDSS Symposium 2015.

[19] G. De Maio, A. Kapravelos, Y. Shoshitaishvili, C. Kruegel, and G. Vigna. PExy: The Other Side of Exploit Kits. In Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 2014.

[20] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In World Wide Web Conference, 2010.

[21]https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_network.htm

[22] https://www.anaconda.com/download/

[23] http://deeplearning.net/software/theano/

[24] https://www.tensorflow.org/install/install_windows

[25] https://keras.io

[26] Clam AntiVirus, http://www.clamav.net/

[27] Avira AntiVir Premium, http://www.avira.com/

[28] http://rips-scanner.sourceforge.net

[29] Y. Zheng, X. Zhang, and V. Ganesh, “Z3-str: A z3-based string solve for web application analysis,” in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ser. ESEC/FSE. 2013.