WebWinnow: Leveraging Exploit Kit Workflows to Detect Malicious URLs

Birhanu Eshete
Fondazione Bruno Kessler
Trento, TN, Italy
eshete@fbk.eu

V.N. Venkatakrishnan
Universtity of Illinois at Chicago
Chicago, IL, United States
venkat@uic.edu

ABSTRACT

Organized cybercrime on the Internet is proliferating due to exploit kits. Attacks launched through these kits include drive-by-downloads, spam and denial-of-service. In this paper, we tackle the problem of detecting whether a given URL is hosted by an exploit kit. Through an extensive analysis of the workflows of about 40 different exploit kits, we develop an approach that uses machine learning to detect whether a given URL is hosting an exploit kit. Central to our approach is the design of distinguishing features that are drawn from the analysis of attack-centric and self-defense behaviors of exploit kits. This design is based on observations drawn from exploit kits that we installed in a laboratory setting as well as live exploit kits that were hosted on the Web. We discuss the design and implementation of a system called WEBWINNOW that is based on this approach. Extensive experiments with real world malicious URLs reveal that WEBWINNOW is highly effective in the detection of malicious URLs hosted by exploit kits with very low false-positives.

Categories and Subject Descriptors

K.6.5 [**Security and Protection**]: Invasive Software

Keywords

Exploit Kits; malware; detection; machine learning

1. INTRODUCTION

Cybercrime has seen a proliferation in recent times. This has been largely due to development of criminal, for-profit, software infrastructure for conducting attacks on end-users. In this infrastructure, Exploit Kits occupy a central role as they facilitate the infection of users through browser compromises. An exploit kit is a piece of off-the-shelf software that can be licensed from the underground market, which when installed and configured on a web server, carries out a malicious campaign targeting innocent victims. Examples of attacks launched through these kits include drive-by-downloads, spam and denial-of-service. A website hosting an exploit kit is advertised through URLs disseminated through spam links, search campaigns, social networks, web postings or website hijacking. Innocent victims that click on these URLs have their systems compromised through drive-by-download attacks, and the infected hosts are subsequently used for further criminal activities.

Given the role of exploit kits in cybercrime, it is a natural question to ask whether a given URL points to an exploit kit. This is a question that has significant implications for the safety of end-users on the Internet, given the proliferation of criminal activities in recent years. Therefore, identifying such exploit kit hosted URLs has the potential to be useful to a broad audience. For example, search engine vendors can use such detection techniques to prevent indexing of such URLs as they are usually accompanied with search term poisoning attacks. They also can pass on such blacklisted URLs to end users and to browser vendors, who can protect their customers from harmful infection. Anti-virus companies can also keep their signature datasets up-to-date by analyzing such URLs.

Previous work on detecting exploit kit hosted pages primarily relied on either looking for anomalous characteristics of the downloaded code [3, 11] or detect [9, 10] changes to the OS persistent state. The former is harder to define for new exploits included by the exploit kits (which are continuously updated), while the latter, involves the difficult issue of correct execution of malicious code.

This paper develops a new approach that uses machine learning to facilitate fast detection of exploit kit hosted pages. A unique characteristic of our approach is that it is drawn from an extensive analysis of the design and implementation of several available real-world exploit kits. This analysis is performed by installing the exploit kit software in a controlled environment, studying its source code, and by dynamic analysis through interaction with various browser personalities. In particular, we leverage on two sets of characteristics exhibited by all the exploit kits that we tested. The first set comes from their *attack-centric behavior*, that includes their methods for attacking browser clients (e.g., fingerprinting, obfuscation). The second set comes from their *self-defense behaviors*, i.e., those characteristics that are used to evade detection techniques. Based on our understanding of these two types of behaviors, we derive a set of features to train a classifier.

In addition, we augment the feature set obtained from our analysis with a set of features drawn from live exploit kits. Quite often, a real world deployment of an exploit kit (in terms of the number of redirections, domains traversed and similar customizations) often do not exhibit themselves when such kits are studied in a controlled setting. The set of features from locally deployed kits as well as from the live exploit kits, taken together, facilitate the training of a precise classifier that is used for detection.

Our system, WEBWINNOW, has a training phase and an actual detection phase. The training phase makes use of honeyclients that engage with locally installed kits and collect activity logs that are

CODASPY'14, March 3–5, 2014, San Antonio, Texas, USA.

http://dx.doi.org/10.1145/2557547.2557575.

then used for training. In a similar fashion, we extract activity logs for live exploit kit URLs. During the detection phase, WEBWINNOW, when presented with a possibly malicious URL, interacts with the URL using a honeyclient, and classifies whether the URL is an exploit kit or not.

Analysis of URLs with exploit kits using a honeyclient infrastructure can be expensive from a resource standpoint. Past studies have recommended the use of search engines [6]. As a resource saving step, WEBWINNOW includes the use of a pre-filtering engine that was built based on our analysis of exploit kits. The pre-filtering engine scans a URL dataset and identifies those that are suspicious. These suspicious URLs are then forwarded to a honeyclient which then uses the classifier built for detection.

We perform an evaluation of WEBWINNOW over a dataset collected over a period of five months. Our dataset consists of a collection of Alexa top sites and publicly available malicious URL datasets. Our evaluation demonstrates a precise classifier yielding a true positive rate of 99.9% on a testing sample of malicious URLs.

This paper is organized as follows. Section 2 presents background on exploit kits. Section 3 presents a brief overview of our approach followed by a discussion of the choice of features for machine learning. Section 4 discusses the training of the classifier and the design of our training infrastructure. We discuss the detection phase in Section 5. A discussion of the implementation and evaluation appears in Section 6. Discussion of related work appears in Section 7. We conclude in Section 8.

2. BACKGROUND ON EXPLOIT KITS

Exploit kits are one of the most common methods used in cybercrime and their emergence has been well documented in recent work [4]. Cybercriminals target vulnerabilities in operating systems, web browsers, office software, and third-party plugins to infect victim machines. The advent of exploit kits dates back to 2006 when the MPack kit came in to picture for the first time [8].

A typical workflow in an exploit kit usually starts with a victim being lured to a URL (e.g., by clicking a link in a spam email) to visit a seemingly benign web page. After a series of redirections, the victim reaches a landing page. The kit then gathers identifying information of the victim in pursuit of vulnerabilities to exploit. At this stage, a kit analyzes the User-Agent information of the victim to identify the type and version number of the operating system, browser, and third-party plugins. If the exploit succeeds, a malicious payload (malware) is silently downloaded and executed on the victim's machine. In addition to delivering the exploit payload to the victim, the exploit kit also updates the infection statistics accessible to the kit administrator.

Exploit kits come with pre-written exploit code and they usually include installer scripts, a number of exploits, configuration details, administration features for the kit owner, and payload which includes botnets, fake anti-virus engine, and trojans. Exploit kits are sold (rented) in the underground marketplace and like legitimate software there are regular bug fixes, feature enhancements, customer support, end-user license agreements, and even competitions among the kit developers. Most exploit kits are written in PHP (some in C/C^{++}), use fingerprinting code, use extensive obfuscation, and use a number of third-party code (e.g., JavaScript and CSS). Examples of popular Exploit Kits include RedKit, CrimePack, CrimeBoss, Cool, Blackhole, SweetOrange, Eleanore, Fragus, NuclearPack, Sakura, NeoSploit, Siberia Private, and Styx.

According to the 2013 Internet Security Threat Report by Symantec [13], Blackhole had a share in 41% of all Web-based attacks in 2012. There was also the release of an updated version of the kit (Blackhole 2.0) hinting that its impact is likely to continue. Another exploit kit is Sakura which took 22% of overall kit usage in 2012. The next 3 in the top 5 kits in 2012 are Phoenix (10%), RedKit (7%), and Nuclear (3%).

3. FEATURE DESIGN

In this section, we discuss how, based on our analysis of 38 distinct exploit kits, we derive distinguishing features based on which we train our classifiers for detection. Before discussing these details, we give an overview of our approach in WEBWINNOW.

In a nutshell, our approach is formulated as a machine learning based technique for the detection of malicious URLs hosted by exploit kits. At the core of our approach is that, based on the workflows of exploit kits, we leverage their *attack-centric* and *self-defense* behaviors to design distinguishing features based on which we train precise classifiers to detect malicious URLs. In this regard, our approach differs from systems (such as [3, 9]) that combine honeyclients and learning-based approaches to analyze the side-effects of malicious activities by inspecting pre-execution and post-execution snapshots of system properties. The following three steps summarize the pipeline of our approach:

- We first analyze source code and runtime behavior of exploit kits by installing them in a controlled setting. In addition, we probe live exploit kits on the Web to capture their activity and content they deliver to the client. By combining these observations, we identify attack-centric and self-defense mechanisms of exploit kits. Based on a careful examination of the attack-centric and self-defense behaviors, we then design features that are precise enough in distinguishing URLs hosted by exploit kits.

- Next, we extract these features from samples of (1) exploit kits we installed locally (2) live exploit kits and (3) non exploit kit URLs. Using the features, we train and evaluate learning algorithms from which we generate detection models. We discuss the training in detail in Section 4.

- Finally, given an unknown URL, we query the detection models to give the verdict as to whether the URL is an exploit kit URL or not. In Section 5, we describe the detection in detail.

Exploit kits manifest different characteristics in order to fulfill their major mission —to infect victim machines with malware. At the same time, we have observed that exploit kits have to do a great deal of self-defense to complicate the analysis task of detection systems. From an anonymous source, we were able to have access to the source code of 38 distinct exploit kits. With the purpose of leveraging these characteristics of exploit kits for detecting URLs linked with them, we analyzed these exploit kits. For these exploit kits, our analysis involved semi-automated source code inspection, installation, and then probing of the kits in a laboratory setting. For exploit kits hosted on the Web, as we do not have access to their source code, we relied on probing them in a contained environment. In the rest of this section, we describe how we take advantage of the attack-centric and self-defense behaviors of exploit kits to develop features we use to train our classifiers for detecting malicious URLs hosted by exploit kits.

3.1 Attack-Centric Behaviors

From the point of view of detecting malicious URLs that lead to exploit kits, we noticed that the attack chain of an exploit kit when it is visited by a victim client gives useful insights to characterize exploit kit behaviors. From our analysis, we identify the following four attack-centric behaviors of exploit kits.

3.1.1 Client Profiling

From our analysis, we noticed that an essential component in all exploit kits is the use of a fingerprinting routine in a form of obfuscated JavaScript code that is unpacked and executed on the client browser. Fingerprinting in most exploit kits has two components performed in the order –*identification* and then *validation*. Identification is about collecting the client profile which includes the type and version of the operating system, browser, and installed browser plugins. In particular, most exploit kits (34 out of 38) check the presence and version number of Acrobat Reader, Flash Player, and the Java WebStart plugin. All the 38 exploit kits we analyzed perform identification of the visiting client.

Validation involves further inspection of the client personality to check if the client is a real user as opposed to a bot. As per our analysis, 36 out of the 38 exploit kits perform validation with the exception of Eleonore and RDS which proceed to preparing the exploit based only on the identity of the client.

The crucial features we capture from client profiling are routines invoked when an obfuscated JavaScript is unpacked and executed on the client side. These are fingerprinting specific features manifesting themselves as routines from the common PluginDetect[1] library that is widely used by exploit kits to collect identifying information of the client. For example, de-obfuscation of the obfuscated JavaScript code used by exploit kits for fingerprinting consistently shows the occurrence of the flagship functions such as `getjavainfo.jar`, `pdpd()`, and `getversion()`. We confirmed the presence of these functions in 28 of the 38 exploit kits while the other 10 exploit kits used function aliases to wrap these functions. The feature *PluginDetect Routines* under the Interaction-Specific feature class in Table 1 is used to capture the unique number of these functions during client profiling.

3.1.2 Chain of Redirections

Live exploit kits we probed silently take clients through a chain of redirections to ultimately land them on the page that hosts the actual exploit. Depending on the remote resource they want to interact with (access), they use different redirect types. These include window-open, HTTP (with different status codes like HTTP 3XX), script source (mostly used to point to remote JavaScript), params (redirection based on URL parameters), applet (remote invocation of Java applet), JNLP (Java Network Launch Protocol invocation), iframe (specially hidden and small iframes as legitimate iframes are large enough to be visible), and HREF-based redirection. According to our observation when probing live exploit kits on the Web, we confirmed that exploit kits such as RedKit, SweetOrange, Blackhole, and Cool are particularly notorious in using complex redirection chains by combining these redirect methods.

3.1.3 Exploit Obfuscation

Except 2 exploit kits (BleedingLife and Cry217), all the rest (about 95%) of the exploit kits we analyzed obfuscate the malicious JavaScript code they use in client profiling, exploit preparation, and exploit delivery. Obfuscation renders static analysis techniques ineffective and further complicates the task of dynamic analysis. Usually, the obfuscation technique is one of the components to be revamped when a new version of an exploit kit is released. Because, by the time the kits are popular on the web, the obfuscation technique is likely to be uncovered and it is no more effective.

3.1.4 Allowing Additional Exploits

According to our analysis, 7 out of the 38 exploit kits allow uploading an additional exploit binary. To confirm the addition of new exploits, we first added a key-logger executable to CrimePack3.1.3 and Fragus1.0 exploit kits. Then, we visited the kits from a vulnerable client. In both cases, the key-logger executable was successfully downloaded to the client.

3.2 Self-Defense Behaviors

The primary purpose of exploit kits is to be effective in infecting victims with malware. However, to continuously infect victims, they need to escape detection techniques, for which they have to be armed with self-defense functionalities. In the following, we discuss six self-defense behaviors we observed in our analysis.

3.2.1 IP Blocking

In an attempt to escape from automated analysis techniques, almost all exploit kits we analyzed are equipped with IP blocking mechanisms against bots (e.g., GoogleBot) and IP addresses from ToR networks. About 87% of the exploit kits we analyzed have features to block IP addresses of known services from Anti-Virus companies or security researchers. The only exploit kits that do not use IP blocking are Bleeding Life, El Fiesta, NeoSploit, SEOSploitPack, and the Yes exploit kit.

3.2.2 Blacklist Lookup

Exploit kits check if their URL is blacklisted in one or more public blacklisting services. If so, the kit administrator changes the URL and the kit continues to operate until it is discovered and blacklisted again. In total, 6 of the 38 exploit kit in our analysis check their URL against one or more blacklists. CrimePack, for instance, has a feature that allows to lookup the kit URL in 8 major blacklists including the Google Safe Browsing service.

3.2.3 Signature Evasion

Anti-virus engines and Intrusion Detection Systems use string signatures to detect malicious content. Some exploit kits lookup the signature databases of online malware (virus) scanning services (such as virtest[2], virustotal[3]) so as to check if their signatures belong there. For example, the latest version of Blackhole (version 2.1.0) checks for its own signatures in *virtest* and *scan4you* online virus-scanning engines. The 5 exploit kits that check for their own signatures in online ant-virus engines are Adrenalin, Blackhole, Fragus, SpyEye, and ZeusKit.

3.2.4 Cloaking

We confirmed that 15 out of the 38 exploit kits use cloaking. When visited after a successful exploit delivery, most (13) of them respond with the HTTP 404 (page not found) error while a few (2) responded with a blank page with no content and/or action. There are two cases in which an exploit kit might use cloaking. One is up on failure to exploit a client either because the client turns out to be invulnerable or it is a bot. In this scenario, the kit responds with HTTP 404 error or redirects the client to a harmless page. Some exploit kits throw back to the client a random exploit just to try their chance (we noticed this in 0x88, Eleonore, Fragus, and Sava). The other case in which cloaking is used is when an already infected victim visits the exploit kit page. In this case, most exploit kits respond with HTTP 404 while some show a blank page.

[1] `http://www.pinlady.net/PluginDetect/`

[2] `http://virtest.com`

[3] `http://virustotal.com`

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

Table 1: Features used in WEBWINNOW.

3.2.5 Blocking of Robots

We found the `robots.txt` file in 14 of the 38 exploit kits and in all the cases the kits disallow any indexing attempt. Search engine crawlers rely on the `robots.txt` file placed in the public directory of websites to check permission for indexing of web pages. Some exploit kits (e.g., CrimePack, Eleonore, Fragus, Nuke, Phoenix, Siberia) disallow indexing by blocking all bots. This is done in an attempt to escape from a more advanced analysis by search engines to filter-out (from their index) suspicious or malicious web pages. Although not all exploit kits block robots, our analysis shows that, when combined with attack-centric characteristics (e.g., client profiling, redirections), this behavior contributes to identify exploit kit URLs in practice.

3.2.6 Code Protection

Our analysis shows 3 distinct outcomes as to the code protection scheme used by exploit kits. About 8 (24%) use `IonCube`, about 5 (13%) use `ZendGuard`, and about 4 (10%) use custom code protection scheme (e.g., custom cryptography). The motivation in code protection is primarily to prevent code stealing while at the same time giving hard time for detection techniques that use code analysis. Despite these code protection mechanisms, source code is sometimes leaked. To combat source code leakage, some exploit kits (e.g., Blackhole) are shifting their "business model" to only rental —exploit-as-a-service paradigm.

3.3 Features

The features we draw from the attack-centric and self-defense behaviors of exploit kits are summarized in Table 1. We designed a total of 30 features of which 5 are *Aggregate-Behavioral*; 11 are *Interaction-Specific*; 4 are *Connection-Specific*; and 10 are *Content-Specific* features. Of all the the features, the interaction-specific features are the most discriminating features because we characterize fingerprinting and the various redirection attempts using fine-grained details of interaction types. In the following, we briefly describe these features. In Section 6, we evaluate the statistical significance of these features based on the training dataset.

3.3.1 Aggregate-Behavioral

These 5 generic features provide a course-grained characterization in terms of the number of known *exploits* (from `http://exploit-db.com`), aggregated count of *redirections*, total number of *connections* made to other destinations, total number of *locations* from which remote content is fetched, and total number of distinct *files* dropped on the client.

3.3.2 Interaction-Specific

In this class of features, 8 out of the 11 features have not been used by previous work. Only *HTTP Redirects*, *Script-SRC Redirects*, and *Link Redirects* have been used in previous work to detect malicious web pages. We noticed that these fine-grained redirection features are effective in identifying exploit kit URLs. Another particularly useful feature is the *PluginDetect Routines*, which counts the number of fingerprinting routines used by exploit kits.

3.3.3 Connection-Specific

The 4 features in this class are captured when connections are made to remote sources. They include *ActiveX* loading attempts (e.g., presence of Microsoft XMLHTTP ActiveX), *heap-spray* attempts (e.g., when the argument to the `eval()` function is too large), unique number of *countries* traversed and *Top Level Domains (TLDs)* involved in the connection. Such features are attributed to the strategy of cybercriminals to complicate the traceability of the exploit kit infrastructure (e.g., during take-down operations by law enforcement).

3.3.4 Content-Specific

We extract a total of 10 features from the content related activity logs. Some MIME-types (e.g., PDF, Shockwave file) are particularly attractive for exploit kit URLs as they target vulnerable plugins that render such file types. More precisely, we extract the the size of content delivered to the client for common MIME-types used on the web with emphasis on content-type used by exploit kits. In particular, these features include size of HTML, CSS, JavaScript, Octet-Stream (e.g., long byte pattern), command, plain text, compressed content (e.g., *. zip, *.gz, *.tar), XML, PDF, and Postscript content. Notice that where there is obfuscation these content-specific features are extracted after de-obfuscation so as to obtain feature values that can distinguish content delivered by exploit kits from content delivered by benign web sites.

4. TRAINING

We now present the training phase of WEBWINNOW by describing the training infrastructure for exploit kits we installed locally, live exploit kits on the Web, and non exploit kit URLs.

Based on the features derived from the behavioral characterization of exploit kits we described in the previous section, the goal of the training is to generate a model that is used to detect exploit kit URLs. To enrich our training set, we capture as much

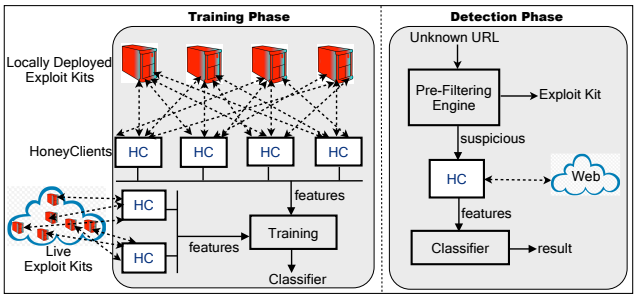

Figure 1: WEBWINNOW Training and detection pipeline.

activity-centric details as possible of both exploit kit and non exploit kit URLs in a contained environment. To this end, we divide the acquisition of our training dataset into three parts. The first part probes locally deployed exploit kits. The second part deals with probing of live exploit kit URLs. The third part involves probing of non-exploit kit URLs. In what follows, we describe these three contained probing steps in detail.

4.1 Locally Installed Exploit Kits

To inspect the attack-centric and self-defense behaviors of exploit kits, we installed exploit kits for which we were able to get the source code. The advantage of having locally installed exploit kits is twofold. First, it gives a deeper understanding of the operational mechanics of the exploit kits, specially how they orchestrate attacks. Second, it gives insights as to how the infection statistics is collected and the mechanisms that the exploit kits employ to evade detection techniques. In a virtualized environment with multiple exploit kits deployed on one side, we schedule honeyclients on the other side to visit the exploit kit URLs with a vulnerable user-agent personalities (details in Section 6.1). The honeyclients are directly pointed to and visit the landing page of the exploit kits so as to collect whatever the exploit kits throw back to the client.

More precisely, we collect HTML and CSS content, script loaded (mainly JavaScript), exploit payload dropped into the honeyclient, remote content fetched, and shell-code execution attempts, and more importantly the whole HTTP transaction. After a few seconds, we repeat the visit in order to compare the activity logs with the first visit. In most exploit kits, if the first visit succeeds in dropping an exploit payload, the subsequent visits from the same address of the honeyclient are usually served with benign-looking response or error codes.

4.2 Live Exploit Kits on the Web

For probing the activity log of live exploit kits on the Web, we rely on our data sources of live exploit kits in the wild. Given a live exploit kit URL at hand, we launch one of our honeyclients and probe the server to collect its activity logs with details similar to the exploit kits installed locally. The honeyclients we use for live exploit kits are separate from the ones we use for the locally installed kits. More importantly, we reset the honeyclients after collecting the activity logs of a given exploit kit before moving on to visit the next exploit kit URL.

An attentive reader might wonder about the distinction between the contained probing of locally installed and live exploit kits on the Web. There are two fundamental variations pertinent to the typical steps involved in exploit kit workflow we described in Section 2. In the first place, live exploit kits are engaged in a series of redirections before reaching the landing page while the locally installed ones are not. Secondly, compared to the versions of the kits we deployed locally, we presume that live exploit kits are likely to be recent versions which give us a fresher insight of their workflow.

4.3 Non Exploit Kit URLs

For this class of URLs, we use publicly known benign URLs to monitor their execution activity using the same configuration of honeyclients used for locally installed exploit kits and live exploit kits. The difference in this case, however, is that we do not repeat the probing as we assume that these URLs have a fairly stable activity log apart from changes in page content and client-side code, which we do not analyze directly. We also use separate honeyclient instance to probe non exploit kit URLs to ensure sanity of logs.

4.4 Model Generation

To verify the exploit kit samples (of both locally installed and those live on the Web), we semi-automatically inspect the activity logs we collected to ensure that they drop into the honeyclient at least one exploit payload. Similarly, for non exploit kit samples, we manually check for the presence of suspicious samples that might arise due to compromised legitimate websites that might have been hijacked by cybercriminals and injected with an iframe leading to an exploit kit URL. After verifying the samples collected, we extract the 30 features described in the previous section. Finally, we label the samples as EK (Exploit Kit) for samples from locally installed and live kits on the Web or as NEK (Non-Exploit Kit) for those which are known to be benign.

Using the labeled samples, we use established supervised learning algorithms to train classifiers and evaluate them to decide which classifier to use for detection. The metrics to evaluate the classifiers is True Positive Rate (TPR) and False Positive Rate (FPR) on the training set using 10-fold cross validation. We used 3 tree-based learning algorithms namely: J48 Decision Tree, Random Tree, and Random Forest. In addition, we also used one stochastic learning algorithm (Bayes Network) and one function-based learning algorithm (Logistic Regression). Performance evaluation of these algorithms during training and testing is discussed in Section 6.

5. DETECTION

Given an unknown URL, to detect if it leads to an exploit kit site, WEBWINNOW's detection phase works as follows. If resource is a bottleneck, a pre-filtering step is invoked to check if the URL under examination can be quickly matched against pre-compiled rules based on URL anatomy of popular exploit kits. If not, the URL is directly analyzed using the learning-based classification. In the rest of this section, we discuss these two components of the detection in more detail.

5.1 Learning-Based Classification

Given an unknown URL, first a honeyclient probes the remote server the URL points to in order to collect activity logs from the beginning to the end of the interaction. From the transactions of the interaction, activity-centric features are extracted and an already trained model is queried to classify the URL as an exploit kit URL or not. The main assumption in the activity-centric characterization to distinguish exploit kit URLs from other types of URLs is the intrinsic workflow in exploit kits that involves client fingerprinting, series of redirections, attempts to deliver and then execute exploit payload, along with typical self-defense behaviors (e.g., cloaking).

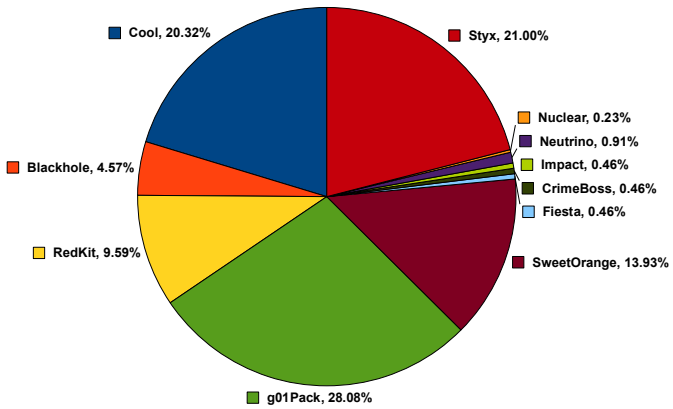

Figure 2: Distribution of live exploit kits probed.

5.2 Implementation Overview

To probe remote URLs, we use Thug[4], an open source honeyclient written in Python. Thug is configured on a virtual machine image that we replicate as needed. We preferred using Thug because the level of granularity in the activity logs captured when probing a remote URL is suitably detailed for the workflow modeling in exploit kits.

Before probing exploit kit URLs, identifying parameters of the honeyclient personality such as user-agent string (e.g., browser type and version, operating system type and version), referrer, DOM events, and plugins are properly configured. We do this not only to simulate a realistic browsing behavior but also to maximize the chance of capturing the end-to-end infection chain that is orchestrated by exploit kits.

6. EVALUATION

This section discusses the evaluation of our approach in light of accuracy of the models generated, accuracy of classifiers on a separate test set, comparison of our approach with similar techniques, and analysis of false positives. More precisely, our evaluation is aimed at answering the following questions:

RQ$_1$: *Does workflow analysis of exploit kits and characterizing them with distinguishing features based on their attack-centric and self-defense behavior improve the detection accuracy of existing techniques in malicious web content detection?*

RQ$_2$: *Since client-side attacks such as drive-by-downloads are mostly initiated by exploit kits, can we build a mechanism to combat exploit kits in real time?*

RQ$_3$: *In modeling the dynamic behavior of exploit kits, can we leverage interaction of a potentially vulnerable client and an exploit kit server so as to build better models?*

6.1 Data Collection

To collect the dataset we use for our evaluation, we analyzed exploit kits in the period April-August 2013 for about 5 months in a contained environment. In this period, we scheduled the honeyclients to periodically visit and record the interactions involved with the exploit kits we installed locally. At the same time, we also probed on a daily basis live exploit kit URLs reported by online detection services and blacklists.

Using the locally installed exploit kits on the server side, we deployed 4 virtual machines as clients with the Thug honeyclient configured with Internet Explorer 6.0 on Windows XP SP 3 with plugins for Adobe Acrobat Reader (version 9.1.0), Java plugin (version 1.6.0.32), and Shockwave Flash (version 10.0.64.0). We noticed that most exploit kits check for such combination of browser, operating system, and plugins. As for the DOM events' configuration, we used the default which enables *load()* and *mousemove()* events.

With the same honeyclient configuration, for live exploit kits, we relied on the URLs reported as exploit kit URLs by the URLQuery[5]service. When we find these exploit kit sites live, we use our honeyclient to probe the URL and record the HTTP transaction. For each live exploit kit URL that succeeds in dropping an exploit binary to the honeyclient, we re-probe it after 10 seconds in case of different behavior (e.g., to check if it performs cloaking). It is important to note that collecting the execution dynamics of live exploit kits was not an easy task as exploit kits have a very short lifetime (usually less than 24 hours). This is because the kit owners relocate them once their URL is blacklisted. During the data collection, we frequently noticed that these URLs become unreachable in a matter of few minutes after they are publicly reported.

The data collection from live exploit kits involved 11kits on 500 exploit kits for which we semi-automatically verified the delivery of at least one exploit binary. Figure 2 shows the percentage distribution of each exploit kit in the collected dataset for training. In the top 3 are g01pack (28%), Styx (21%), and Cool (20%) comprising nearly 70% of the dataset followed by SweetOrange (14%), RedKit (10%), and Blackhole (5%). Interestingly, on June 26, 2013 McAfee released a report [12] that shows correlation with the high percentage of the Styx exploit kit in the data we collected.

Purpose	Non exploit kits	exploit kits	Total
Training	500	500	1000
Testing	1117	512	1629

Table 2: Dataset for training and testing.

6.2 Dataset

To train the classifier, we rely on samples collected from the execution dynamics of the exploit kits we installed and executed in a contained environment and exploit kit URLs that we probed on the Web and few others from [2, 14, 15, 17]. For non exploit kit samples, we used the Alexa top sites as we assume that these sites are less likely to host exploit kits. The collected dataset is divided into labeled training and testing set as shown in Table 2. The training set is used to train different classifiers and evaluate their performance while the testing set is used to evaluate the performance of the trained models on a dataset disjoint with the training set.

6.3 Setup

Training. Using the features collected on the training set, the WEKA machine learning suite [5] was used to train five supervised learning algorithms to derive detection models. The algorithms are: J48 Decision Tree, Random Tree, Random Forest, Bayes Network, and Logistic Regression. The training was done with 10-fold cross validation for all the algorithms.

Testing. On the testing set, the trained classifiers were independently ran and the classification results were analyzed for accuracy and false positives.

6.4 Results

6.4.1 Analysis of Features

Based on the training dataset we used in this work, we now discuss the significance of the features we presented in Section 3.

[4]https://github.com/buffer/thug

[5]http://urlquery.net

Classifier	TPR	FPR
J48 Decision Tree	100%	0.000
Random Tree	97.1%	0.029
Random Forest	99.8%	0.003
Bayes Network	100%	0.000
Logistic Regression	99.9%	0.001

Table 3: Performance of models on the training set.

Classifier	TPR	FPR
J48 Decision Tree	99.9%	0.001
Random Tree	89.9%	0.032
Random Forest	99.7%	0.002
Bayes Network	99.8%	0.003
Logistic Regression	99.4%	0.005

Table 4: Performance of classifiers on a separate testing set.

Aggregate-Behavioral. In the training set, the average number of redirections (of all types) for exploit kit URLs is 102 while for non exploit kit URLs is 82. On average, we found 1 exploit in the exploit kit dataset as opposed to zero in non exploit kit dataset. The average number of connections made to remote sources is 25 for exploit kit dataset while it is 20 in non exploit kits. These aggregate values of features, in fact, are fairly discriminative in putting apart exploit kit and non exploit kit URLs.

Interaction-Specific. While we observed an average of 1 JNLP-based redirection in exploit kit URLs, we could not come across even a single JNLP-based redirection attempt in non exploit kit URLs. The average number of HTTP 3XX redirections shows that the average is 5 in exploit kits compared to an average of 2 in non exploit kit dataset. One of the useful features in this class is the *PluginDetect Routines* with an average count of 2 for exploit kit dataset as opposed to zero for non exploit kit URLs.

Connection-Specific. Average number of ActiveX loading attempts is 4 for exploit kit dataset while it is 2 for non exploit kit dataset. Average number of unique countries traversed and TLDs involved is 4 and 3 respectively for exploit kit URLs. Whereas in non exploit kit dataset, it is 1 for both the countries and TLDs.

Content-Specific. On average, the size (in bytes) of plain text, HTML, CSS, and XML delivered by exploit kit URLs is less than content delivered by non exploit kit URLs. On the other hand, the average size of JavaScript, Octet-Stream, and portable document is greater in the case of exploit kit URLs. For instance, Octet-Stream average size in exploit kit dataset is 13 times that of non exploit kit octet-stream content. Similarly, the average JavaScript content delivered by exploit kit URLs is about 1.5 times bigger than JavaScript in non exploit kit URLs. These statistical variations are indicators of the discriminative power of these features in practice.

6.4.2 Accuracy of Models on the Training Set

The performance of the classifiers derived from our training set are shown in Table 3. We use the usual meanings of TPR (the percentage of correctly classified URLs for both labels), and FPR (the percentage of misclassified URLs for both labels). Using 10-fold cross validation, J48 Decision Tree and Bayes Network classifier have 100% TPR and 0% FPR. In fact, only the Random Tree classifier has the lowest TPR and the highest FPR (misclassification of 29 in 1000 URLs). Overall, the classifiers are quite precise which indicates the discriminative power of the features we derived by leveraging the attack-centric and self-defense behavior of exploit kits' workflow.

6.4.3 Accuracy of Classifiers on the Testing Set

Table 4 shows the accuracy of the classifiers on an independent test set. Overall, all the classifiers except Random Tree achieved above 99% accuracy with very low FPR. As shown in the results, the the classifiers are precise enough to correctly classify samples disjoint with the training set, which shows that our system is effective for realtime detection of exploit kit URLs.

6.4.4 Error Analysis

Here, using manual analysis, we reason out as to what caused the misclassification of URLs on the testing set.

The J48 Decision Tree classifier misclassified 2 exploit kit samples as non exploit kit URLs. One of these URLs is a file-sharing website from which users download content by interacting with the website and it has no redirections up on first encounter. However, when we manually clicked on the "Download" link, we witnessed 3 redirects to lead to a file to download. For the other URL, the (limited) data we had was not sufficient to infer a reason for the misclassification. The Random Tree classifier, misclassified 23 exploit kits as non exploit URLs. As shown in the results, this classifier was not as precise as the others. A manual inspection showed that the feature values for octet-stream and JavaScript content are less than the average size of these features for exploit kit URLs in the training set, which, we speculate, might have slightly skewed the classifier. The Random Forest classifier misclassified only 3 exploit kit URLs as non exploit kit URLs. In this case, 2 of the URLs span a single country and top-level domain, which is below the average feature value of 4 and 3 respectively for exploit kit URLs in the training set.

6.4.5 Comparison with Similar Systems

To compare our system with similar systems, we prepared a separate comparison set of 100 URLs reported to be linked with exploit kits by the URLQuery service at `http://urlquery.net`. We then submitted the dataset to WEBWINNOW and a public service, Wepawet [16], at about the same time. The results are summarized in Table 5. Overall, WEBWINNOW classified 31 URLs as exploit kits while Wepawet classified 19 as malicious and suspicious combined (2 and 17 respectively). Of the 31 URLs flagged as exploit kits by WEBWINNOW, Wepawet flagged 2 as malicious, 4 as suspicious, 23 as benign, and 2 as error. Manual analysis of the activity logs of these 31 URLs shows that they have dropped a binary payload to WEBWINNOW's honeyclient. We speculate that the large number of mismatch with Wepawet's output is probably attributed to the fact that it is a public service and the exploit kits might have blacklisted it to serve it with benign response. This mistmatch partially confirmed with Wepawet's detailed reports for 7 of the URLs classified as benign to be pages with 404 (page not found) response.

Wepawet classified 67 URLs as benign and WEBWINNOW classified 63 as non exploit of which 33 URLs match the benign, and 12 URLs match the suspicious classification of Wepawet. WEBWINNOW was not able to finish the analysis of 6 URLs while Wepawet terminated with time-out error on 14 URLs. Overall, this experiment demonstrates the higher precision and accuracy of WEBWINNOW over existing tools in detecting malicious URLs hosted by exploit kits.

7. RELATED WORK

Kotov and Massacci [7] present a preliminary analysis of the source code of 30 different exploit kits. As per their analysis, the key strength of exploit kits is the functionalities to support the kit

Class	WebWinnow	Wepawet [16]
Benign	63	67
Malicious	31	19
Error	6	14

Table 5: Comparison summary.

owner to manage exploits, escape detection, and monitor traffic. On the other hand, they conclude that exploit kits use limited number of and unsophisticated vulnerabilities focusing on large volume of infection traffic to maximize the rate of successful infection of victims. One of the findings is that exploit kits have similar functionality, which also coincides with our preliminary analysis. Our motivation in this paper is to leverage exploit kit workflow in order to design a detection technique while the goal of the approach described in goal is to purely analyze and characterize exploit kits as software artifacts.

In another similar work, Allodi et.al [1] present, MalwareLab, an experience from a controlled experimental evaluation of resilience of 10 exploit kits with respect to changes in software configuration (e.g., browser personality of victims). As per the findings, there seem to be two types of exploit kits with regards to resilience to changes in software configuration. One type is those exploit kits designed to be effective for an extended period of time at the expense of lower infection rate. Another type is those exploit kits designed to be effective in infecting as many victims as possible within a very short period at the expense of low resilience. The contained analysis of exploit kits used in our work is similar to MalwareLab in some aspects such as the use of virtual machines and automation of execution when feasible. However, our goal is to have a deeper insight as to how the exploit kits function to perform attacks and to evade detection techniques.

Grier et.al [4] conducted a large-scale analysis on the emergence of "Exploit-as-a-Service" paradigm on the malware ecosystem by examining the current landscape of drive-by-downloads on the Internet. For the analysis, the authors analyze 77,000 malicious URLs from Google Safe Browsing and a crowd-sourced feed of blacklisted URLs known to lead to exploit kits. These URLs led to over 10,000 distinct executables. The results of contained execution of these binaries show about 32 families of malware (among which are the prominent ones) are carried around on the Web via drive-by-downloads supported by exploit kits (with Blackhole accounting for 29% of all malicious URLs). In addition, the authors also used passive DNS data to conclude that the infrastructure that hosts these exploit kits is short-lived (receives traffic only for about 2.5 hours). Although we did not measure the lifetime of exploit kits, we were also able to notice that they do not live long.

Honeyclient based detection systems (e.g., [3, 9]) rely on pre-execution and post-execution of the snapshot of the system properties to look for changes in system properties (e.g., new processes, strange memory allocation attempts, disk access attempts) to pinpoint malicious activity. In our approach, we take a different path by focusing on the actual execution and leverage it to derive distinguishing features to detect malicious URLs that host exploit kits.

8. CONCLUSION

Exploit kits are prevalent on the Internet and they contribute to a significant portion of web-based threats. Understanding how they function, attack victims, and evade detection systems is quite crucial in designing proactive techniques to detect them in real-time. In this paper, we leveraged the firsthand observation of the attack-centric and self-defense behavior of exploit kits to develop a machine learning approach that precisely detects malicious URLs hosted by exploit kits. Our evaluation shows that the classifiers we trained based on the features derived from the exploit kits' workflow perform very well with very low false positives.

Acknowledgments

This work was performed during the first author's visit to the University of Illinois at Chicago. This research was supported in part by U.S. National Science Foundation grants CNS-0845894 and CNS-1065537 and DGE-069311.

9. REFERENCES

[1] L. Allodi, V. Kotov, and F. Massacci. Malwarelab - experimentation with cybercrime attack tools. In *USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2013.

[2] M. Blacklist. `http://www.malwareblacklist.com/showMDL.php`, June 2013.

[3] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 281–290, 2010.

[4] C. Grier et al. Manufacturing Compromise: The Emergence of Exploit-as-a-Service. In *Proceedings of the 19th ACM Conference on Computer and Communication Security*, October 2012.

[5] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. Weka 3: Data mining and open source machine learning software in java. `http://www.cs.waikato.ac.nz/ml/weka/`, July 2013.

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

[7] V. Kotov and F. Massacci. Anatomy of exploit kits - preliminary analysis of exploit kits as software artefacts. In *ESSoS*, pages 181–196, 2013.

[8] R. Lemos. Mpack developer on automated infection kit. `http://www.theregister.co.uk/2007/07/23/mpack_developer_interview/`, July 2007.

[9] Y. min Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *In NDSS*, 2006.

[10] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iframes point to us. In *Proceedings of the 17th conference on Security symposium*, SS'08, Berkeley, CA, USA, 2008. USENIX Association.

[11] K. Rieck, T. Krueger, and A. Dewald. Cujo: efficient detection and prevention of drive-by-download attacks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, 2010.

[12] S. Surgihalli and V. Krishnasamy. Styx exploit kit takes advantage of vulnerabilities. `http://blogs.mcafee.com/mcafee-labs/styx-exploit-kit-takes-advantage_-of-vulnerabilities`, June 2013.

[13] Symantec. Internet security threat report. `http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v18_2012_21291018.en-us.pdf`, July 2013.

[14] S. E. Tracker. Spy eye tracker. `https://spyeyetracker.abuse.ch/monitor.php?browse=binaries`.

[15] Z. Tracker. Zues tracker. `https://zeustracker.abuse.ch/monitor.php?browse=binaries`.

[16] UCSB. Wepawet. `http://wepawet.cs.ucsb.edu`, Sep 2013.

[17] VxVault. Vx vault. `http://vxvault.siri-urz.net/ViriList.php`.