Detecting Malicious HTTP Redirections Using Trees of User Browsing Activity

Hesham Mekky*, Ruben Torres†, Zhi-Li Zhang*, Sabyasachi Saha† and Antonio Nucci†
*University of Minnesota, Twin Cities, USA. Email: {hesham, zhzhang}@cs.umn.edu
†Narus Inc., CA, USA. Email: {rtorres, ssaha, anucci}@narus.com

***Abstract*—The web has become a platform that attackers exploit to infect vulnerable hosts, or deceive victims into buying rogue software. To accomplish this, attackers either inject malicious scripts into popular web sites or manipulate content delivered by servers to exploit vulnerabilities in users' browsers. To hide malware distribution servers, attackers employ HTTP redirections, which automatically redirect users' requests through a series of intermediate web sites, before landing on the final distribution site. In this paper, we develop a methodology to identify malicious chains of HTTP redirections. We build per-user chains from passively collected traffic and extract novel statistical features from them, which capture inherent characteristics from malicious redirection cases. Then, we apply a supervised decision tree classifier to identify malicious chains. Using a large ISP dataset, with more than 15K clients, we demonstrate that our methodology is very effective in accurately identifying malicious chains, with recall and precision values over 90% and up to 98%.**

I. Introduction

With the wide adoption of firewalls and other security protection mechanisms at end systems, attackers have turned to the web as a platform to launch various malicious activities, e.g., by attracting vulnerable users via social engineering techniques and infecting their machines with malware by exploiting vulnerable web browsers or browser plug-ins. For instance, attackers may compromise certain web sites that a target user base frequently visits, and inject malicious scripts into web content via embedded objects. When vulnerable users visit such sites, the malicious scripts are executed by users' web browsers, which, through a sequence of HTTP redirections, trigger the browsers to download malware from a final malware distribution site, and install them on victims' machines by exploiting vulnerabilities in the web browsers or plug-ins (such mode of attacks is often referred as "drive-by-download"). Attackers have also known to have set up fraudulent web sites and exploit search engine optimization [1], [2] to attract user clicks for scam, or inject malicious ads via the complex online advertisement syndication system in legitimate web sites to peddle rogue software bundled with malware. When users visit infected sites, they are tricked into providing private information or paying for downloading rogue software, e.g., the so-called "fake-AV" attacks [3], where users are warned of a malware infection, and triggered to download fake antivirus software for a fee.

As will be expounded on in Section II-A, *HTTP redirection* is a key tool employed by attackers to evade detection as well as to *dynamically* and *selectively* inflict malware to targeted user bases. Attackers often command a massive malicious infrastructure consisting of subverted web sites, bots made of compromised user machines and leased hosting servers under their control. They utilize HTTP redirections (e.g., HTTP3xx redirections or redirections via dynamically executed scripts) to dynamically switch from one set of intermediate web servers (e.g., when some of which are detected or shut down) to another to evade detection by, e.g, DNS/URL blacklisting, static or dynamic Javascript and web content analysis, or honeynets [1], [4]–[10] while retaining control of their malicious infrastructure. Such multiple HTTP redirections in a sequence not only make it more difficult for security analysts to detect the malicious servers; they also allow attackers to use cloaking techniques [11], [12] to launch *stealthy, target* attacks while making them extremely hard to detection. For example, attackers can selectively attack users based, for example, on the browsers used (e.g., vulnerable versions of Internet Explorer), or the IP addresses of client machines (e.g. ignoring the known honeynets set up by security analysts or clients running virtual machines, supplying "benign" content to search engine bots, or targeting only government agencies/corporations of interest).

Using complete network traces collected at a residential network of a large ISP on two different days about 8 months apart, in this paper we explore whether it is feasible to identify malicious chains of HTTP redirections based solely on statistical properties of these chains. The goal is to develop a lightweight detection system that reduces the reliance on "heavy-duty" deep packet inspection used by most commercial Internet intrusion detection or prevention (IDS/IPS) systems, while circumventing some of the shortcomings associated with conventional DNS/URL blacklisting, static or dynamic web script and content analysis techniques mentioned above. Using labels generated by a commonly used commercial IDS as ground truth, we conduct a systematic analysis of malicious HTTP redirection chains. In uncovering such malicious chains, we build a per-user *browsing activity tree*, where a node in the tree is the requested URL and there is an edge between two nodes if the request for the URL of the child node is triggered from the URL of the parent node (e.g. via an HTTP 302 message, a JavaScript method or an HTML tag). This novel notion enables us to piece together multiple HTTP requests and reconstruct trees of related browsing activities on a per use basis. We focus our analysis on individual paths in the tree, where a path is a sequence of URLs between the root node and a leaf in the tree. Through systematic investigation of malicious HTTP redirection chains and comparing them with benign ones, we identify a novel set of features that are inherent to the network properties of the malicious redirection paths, such as the length of the path, the number of redirections

978-1-4799-3360-0/14/$31.00 ©2014 IEEE

in a very short period of time and the number of distinct domain names involved in a path. We develop a general supervised machine learning methodology using a decision tree classifier, and demonstrate that it is indeed feasible to classify malicious HTTP redirections and separate them from benign ones. Evaluation results show that our machine learning framework can achieve precision and recall values of over 90% and up to 98%. In addition, we also discover new threats, which were not flagged by the IDS.

Our contributions are two-fold. (i) We develop a supervised machine learning-based methodology for detecting malicious HTTP redirection chains. It relies on a novel set of features extracted from the chains to identify malicious and benign cases. These features are fundamentally difficult for attackers to change so as to evade detection while still being able to launch stealthy attacks and retaining control of their malicious infrastructure. In addition, our method relies solely on passive monitoring of network traffic, a much lightweight approach than deep packet inspection, static or dynamic analysis of deep web content or sophisticated JavaScript deobfuscation mechanisms. (ii) We present novel findings on new threats that were not identified by a commercial IDS, and uncover a new stealthy method that attackers employ to infect vulnerable clients, namely, the delivery of unsolicited content through traffic aggregators that is mostly used by porn web sites.

II. Background, Related Work and Datasets

A. Embedded Objects, HTTP Redirections, & Malware Exploits

Today's web sites are highly complex, often involving content or advertisements (ads) provided by multiple entities; much of the content is also dynamically generated. As a result, when a web browser sends a HTTP request to a web site for a piece of content (as identified by a URL), the HTTP response returned by the web site often contain many embedded objects, e.g., CSS links, images, embedded frames, Javascript codes. These embedded objects, some of which may be hosted at other web servers or web sites, often trigger additional HTTP requests when rendered by the browser. During this process, HTTP *redirections*, e.g., in the form of HTTP3xx redirection or redirection executed by an embedded Javascript code, often occur, for example, to redirect the user request to a different web server/site, or insert a piece of dynamically generated content or ad tailored to the user request. For instance, a user in a non-English speaking country, say, Japan, types *www.google.com'* in her browser, she is immediately redirected through an HTTP3xx redirection to *www.google.com.jp*, the Japanese Google site. After she submits a key word to perform search, a list of search results will be returned together with relevant thumbnail images, snippet videos and sponsored ads. When she clicks a link, the web page will typically call a google analytics site, e.g. doubleclick.com, either to download a piece of JavaScript code, an image, etc. We broadly refer to one HTTP request leading to another HTTP request ("automatically" as opposed to user initiated clicks) as an HTTP *redirection*; they include HTTP3xx redirection, Javascript or other script generated redirection (e.g., via the `document.location` method), HTML tag (e.g., `iframes, img` or `meta`) based redirections.

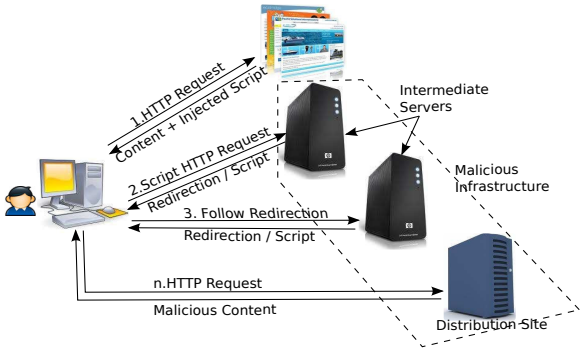


Fig. 1. Example of a malicious redirections that lead to a drive-by-download.

It is well known that attackers have been exploiting embedded objects in web pages and utilizing HTTP redirections to launch various malicious activities while hiding their tracks. For example, an attacker may compromise a popular web site (e.g., a blog site) and exploit embedded objects to inject malicious scripts. When an innocent user visits the site, her browser renders the web page, executing the malicious script, which generates a sequence of HTTP redirections to download malware and compromise the client machine. Since popular web sites are often well-maintained and generally harder to compromise, attackers often resort a variety of other schemes to launch attacks. For instance, an attacker may embed malicious scripts on third party objects, e.g., ads peddling fake anti-virus (AV) software, and buy ad space on popular web sites by exploiting the complex advertisement syndication system [1]. (It is known that Google's doubleclick.com has been exploited by malicious ad networks [2].) Attackers also set up malicious web sites and employ search engine optimization (SEO) techniques to attract user clicks and compromise their machines [13]. As will be shown later in this paper, we have also found that attackers exploit *traffic aggregation* web sites to redirect user requests to less popular and compromised web sites or malicious web sites under their control to inject malware.

HTTP redirection is a key tool employed by attackers to evade detection as well as to *dynamically* and *selectively* inflict malware to targeted user bases. For example, attackers may inject malicious scripts into compromised web servers containing "innocent" looking URLs, which point to other "legitimate" web sites that have also been compromised or set up by attackers as camouflage, so as to evade detection, e.g., by Javascript analysis or malicious URL/web page classification methods [4], [5], [7], [14]. For this purpose, attackers set up and control a malicious infrastructure consisting of compromised web servers, bot machines or their own hosting servers. Fig. 1 depicts a typical drive-by-download redirection sequence, which leads to the final malware distribution server. If any of the intermediate servers or the final distribution server is detected and shut down, the attackers can simply use HTTP redirections to direct user requests via other intermediate servers or another malware distribution site. Furthermore, multiple HTTP redirections allow the attackers to use cloaking techniques [11], [12], where they can selectively attack clients based, for example, on the browser used (e.g. a vulnerable Internet Explorer) or the IP address of the client machine (e.g., ignoring the known honeynet sites set up by security

analysts, supplying "innocent" content to search engine bots, or attacking users from certain specific IP address blocks such as a government agency or corporation of interest).

B. Related Work

We can divide the related research into two groups: (i) analysis and detection of malicious URLs or web content; and (ii) analysis and detection of malicious URL redirections under various settings. Research studies belonging to the first group include those that apply machine learning and other techiques to analyze and classify URLs or web content based on static features and behaviors such as URL lexical patterns, HTML page content, Javascript features, or host attributes [4]–[7], [14]. The first group also includes those studies which apply dynamic analysis techniques, e.g., virtual machines, to crawl and download content from individual web sites that are deemed suspicious/malicious web sites, and monitor the subsequent changes on the client machine, e.g., file system changes such as file creatio, registry alteration, antivirus alerts, to detect drive-by-downloads or other malware activities [1], [8]–[10]. Our approach is different from these existing studies in that our work focuses on malicious URL redirections, and utilize features associated with malicious URL redirections and the intrinsic relationships between redirected web sites to analyze and detect malware activities. Such features associated with URL redirections are hard to capture by per-URL, per-page or per-site analysis. In addition, as our methodology only relies on the URLs generated by malicious scripts not on the analysis of the scripts itself, we do not need sophisticated *de-obfuscation* mechanisms to untangle obfuscated Javascripts. Furthermore, our system can be installed at a client site to avoid the cloacking techniques (e.g., anti-emulation techniques) employed by attackers while protecting them from targeted attacks.

The second group of existing studies are more directly related to our work, as they also focus on malicious activities related to URL redirections. We discuss three recent pieces of research that are most relevant to us. SURF [13] focuses on identifying search poisoning events, where some of the top results returned by a search engine, are malicious pages, not related to the terms searched by the user and that could redirect clients to malware distribution sites. WarningBird [15] focuses on identifying redirections generated from users clicking on malicious URLs posted on Twitter pages. Finally, in MadTracer [2], the authors studied malicious advertisements using paths built by tracing the sequence of requests initiated by an advertisement. The common characteristic of these works is that they focus on particular kinds of redirections (triggered by a search engine, Twitter or advertisements). In contrast, we look at the problem in a far more *general* setting. In addition, we extract features only from the HTTP redirections in a path, while previous works require additional information, such as the keywords searched by a user [13], or the number of times a URL appears in a tweet [15].

C. Datasets

We utilize two 24-hour long datasets collected at the edge of a network within a large ISP in August 2011 and April 2012. The monitored network is mostly residential, with high-speed ADSL connections to the Internet. All incoming and outgoing TCP connections and UDP flows to the network were collected in these two days. To protection privacy, client IP addresses were anonymized and other sensitive information was stripped or sanitized before the datasets were used for our study. In the remainder of the paper, we will refer to the two datasets respective as as the **August2011** and **April2012** datasets. Table I summarizes the key statistics.

TABLE I
AUGUST2011 AND APRIL2012 DATASETS

	August2011	**April2012**
TCP connections	28,548,283	40,552,026
Client IPs	15,242	19,559
Web servers contacted	118,526	145,406
HTTP requests	40,227,123	61,369,864

To generate the *ground truth* for our study, we use a well-known commercial IDS with signatures to label malicious HTTP request-response pairs with one or more corresponding threat identifiers. We ran the August2011 and April2012 datasets through the IDS (using signatures produced by the commerical IDS provider in 2011 and 2013 respectively). For the purpose of this paper, we only consider those threats that are relevant to URL redirections, namely, malicious activities that are known to be launched via URL redirections. These include (i) *Fake-AV attacks*, where a user is lured to buy rogue software such as fake antivirus (AV) software; and (ii) various forms of web-based attacks such as a large number of drive-by-download attacks based on JavaScript and `iframe` signatures.

III. BROWSING ACTIVITY TREES

To proceed with our analysis of HTTP redirection sequences, we need to be able to group related HTTP requests. This is no trivial task since the HTTP requests that we collect from the network do not provide any information regarding the relation across them. In addition, given that web pages present a lot of dynamic content which can be remotely fetched from other servers, one single HTTP request may trigger several HTTP redirections, producing multiple different paths. Finally, we need to consider the time in which the requests are generated so that we can accurately capture a single activity from the user.

To overcome all these complexity, we construct a representation of all the HTTP requests triggered by the user visit to a single web page, which we call the *user browsing activity tree*. A node in the tree is simply an HTTP request and an edge indicates that the request for the child node got initiated through the parent node. The edge is labeled with the action that caused the transition from the parent to the child, such as clicking a hyperlink, a request generated by an `iframe` tag or a JavaScript AJAX request. An example of such a tree can be seen in Figure 2.

A. Tree Construction Algorithm

To construct the tree, we first group HTTP flows by the IP address of the client. Then, we partition the HTTP flows generated by the client into **sessions**, where a client session ends after a 30 second period of silence [10]. We selected a 30 second silence period because it is a reasonable threshold to separate different activities for a single user. A longer period could result in merging two or more trees into a single one and

TABLE II
URL TYPES USED BY THE ALGORITHM

URL Type	Example
HTTP 3xx redirection	Location: `www.domain.com`
HTML meta tag redirection	`<meta http-equiv='refresh' url='http://www.domain.com/'>`
JavaScript redirection (static analysis)	`document.location, location.href`
HTTP Referrer header	Referrer: `www.domain.com`
Other	`<a>, <iframe>, <link>, <img>, <script>`

a shorter period could break a single redirection path into two or more. In addition, the 30 second silence period can capture cases in which attackers deliberately delay the download of the malware by several seconds (e.g. up to 15 seconds in [10]) using JavaScript timer functions.

After identifying the sessions, HTTP records within a **session** are ordered by timestamp to facilitate the analysis. For each HTTP request-response pair, we extract the corresponding HTTP headers and all the URLs embedded in the HTTP response payload. Then, each URL is associated with a type, depending on the kind of redirection that triggered the corresponding HTTP request. The URL type is used to label the edges in the tree. Table II summarizes all URL types that are implemented by our algorithm. Note that this list is not comprehensive, and it represents the heuristics employed by our algorithm to reconstruct the user browsing activity. Additional heuristics will increase the accuracy of the algorithm, but also its complexity (e.g. semi-dynamic JavaScript analysis such as deobfuscating JavaScript code that can potentially contain hidden URLs).

After parsing the current HTTP request payload and extracting the embedded URLs, the current requested URL is looked up to see if the URL appears in a previous HTTP payload. If the current request URL is found in an earlier page, then it gets added as a child node to that page in the tree and the edge is labeled with the corresponding type e.g. `<iframe>`. For instance, assume we had an HTTP request to `url1` and the response contains a JavaScript code that have a `document.location` redirection to `url2`. Then, we save `url2` with an indication that it is a JavaScript redirection from `url1`. Then, after encountering a future HTTP request to `url2`, the saved information is used to add `url2` as a child node to `url1` in the current session tree. This applies to all edge types in Table II. In case that the current URL is not found in any previous HTTP payload, but the **referrer** field from its HTTP header matches a previous URL, then the current URL is attached as a child to that node. Finally, if all lookups fail, then a new tree is created with the current URL as the root. Hence, isolated nodes in a session (nodes with no children) are those that we were not able to find any information to link them with previous nodes (requests) in an existing tree. We use the term "path" to refer to a path from a root node to a leaf node in a single tree, and we use features extracted from paths in the activity trees to build our classification model. In the rest of the paper, we will use the terms "redirection paths" and "redirection chains" interchangeably.

B. Tree Construction Assessment

In this section, we evaluate the accuracy of the tree construction algorithm. For this, we perform three basic steps. In the first step, we build a ground truth dataset of correctly constructed browser activity trees. We develop a browser add-on for Firefox, which adds a new field to the header of every HTTP request to explicitly include the URL of the page that triggered the request. The add-on ensures that the new field is correctly included, even in the case of asynchronous JavaScript requests [10]. With this new header field present in every HTTP request, it becomes trivial to accurately construct the user activity tree from the traffic dump. In the second step, we construct the browsing activity trees from the traffic dump with our algorithm, as described in Section III-A. Finally, we compare both datasets to assess the quality of the trees constructed by our algorithm.

We collected a one-day traffic dump, running Firefox with our add-on installed. Firefox was instrumented to visit the top most popular web sites from Alexa [16], as well as popular file sharing, peer-to-peer and blacklisted web sites. In addition, many web sites were visited simultaneously to emulate a multiple-tabs environment. We generated the browsing activity trees with both our algorithm and the browser add-on and compared them. Results show that our algorithm can correctly construct most of the browsing activity trees, with 96.5% of the trees generated been identical to those generated with the add-on. We found that the remaining 3.5% of the trees belong to suspicious URLs which either obfuscate JavaScript content or manipulate the HTTP Referer information.

IV. IDENTIFYING MALICIOUS REDIRECTIONS

This section presents the details of how we design the classifier to identify benign and malicious redirection paths. First, we present a systematic study on benign and malicious redirection paths, and show some intrinsic differences between the two classes. Then we describe the features selected and the classification technique.

A. Systematic Study of Benign and Malicious Redirections

For this study, we rely on the ground truth provided by the IDS to identify malicious chains. Hence, we consider a chain as malicious if at least one of the HTTP requests in the chain has been flagged by the IDS. Otherwise, we consider the chain as benign, as long as it is not from clients with malicious TCP flows (see Section VI-A for more details).

We begin by showing a typical browsing activity tree constructed by our algorithm, with an identified malicious path in Figure 2. In this example, the client initially visits `nastyvideotube.com`, the root of the tree, which triggers other HTTP requests through different mechanisms and transition times (denoted in the edge legend). For instance, the request for `trafficholder.com` was generated 1 second after the request to `nastyvideotube.com`. As we can see from the figure, several paths can be extracted from the tree. The path that has a leaf node `213.174.132.49` and has thick edges is the malicious path, since node `209.8.22.87` was flagged by the IDS because it caused a fake-antivirus attack. There are other paths that have leaf nodes, such as `maturesflash.com` and `img.wooowmovies.com`, all of them being benign requests for images. We digged deeper into the malicious path and found that the web site `nastyvideotube.com`,

a traffic seller, is registered with `trafficholder.com`, a *traffic aggregation web site*. `trafficholder.com` decides to redirect the user to `thebestofporno.com`, a traffic buyer, which causes a redirection to `maturesflash.com`. This last web site had an injected JavaScript file pointing to `209.8.22.87`, which was eventually flagged by the IDS.

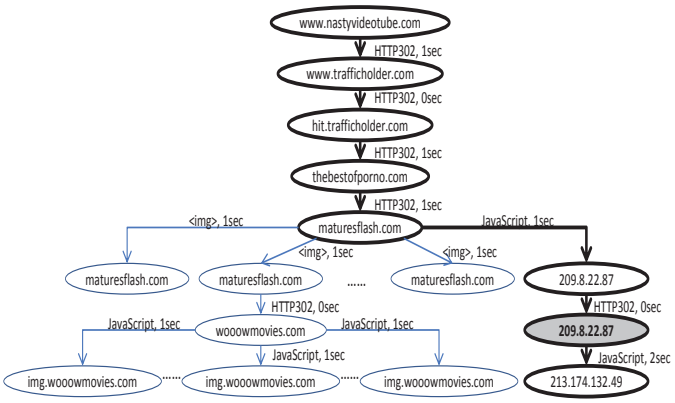


Fig. 2. Browsing Activity Tree. The path with thicker edges is malicious (HTTP request to 209.8.22.87 (dark colored) was flagged by the IDS).

Interestingly enough, we investigated *traffic aggregation web sites (TAW)*, such as `trafficholder.com`, and found that they often appeared in malicious paths in our dataset. It turns out that TAWs buy/sell adult traffic (i.e. user visits) to other sites. In particular, less popular sites buy traffic from TAWs to guarantee a minimum amount of visits. The TAWs, on the other hand, use embedded objects included in more popular web sites (traffic sellers), to redirect users from the original page requested to pages from the traffic buyers. This forces the user to visit other web sites, which could have been exploited or owned by the attacker and are used to initiate drive-by-downloads.

Next, we use a few of the observations from the example above to drive our analysis. We noticed that malicious redirection chains involve: (i) large number of redirections and (ii) include a variety of domain names visited. To generalize this result, Figure 3 shows the mean and standard deviation for four features extracted from all the redirection paths obtained from the data: path length, number of HTTP3xx redirections, number of different domains and the number of redirections between different domains. We compare benign paths and malicious paths. We can clearly see that the mean for all the features is clearly higher for malicious paths than for benign paths.

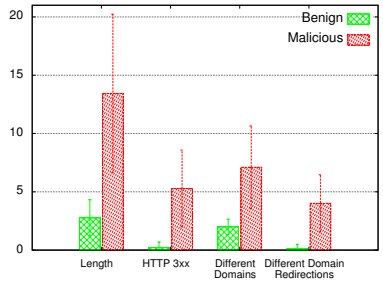


Fig. 3. Mean and standard deviation of benign redirection paths triggered by popular sites and malicious redirection paths

Further, we look at the distribution of edge types in Table III for the datasets. We can see that for malicious redirection chains, over 65% of all the edges types are either HTTP3xx or JavaScript redirections, while for benign cases, it is just over 15% in the August2011 dataset (and similar for April2012).

TABLE III
DISTRIBUTION OF EDGE TYPES FOR REDIRECTION CHAINS

Edge Type	Benign 2011	Malicious 2011	Benign 2012	Malicious 2012
HTTP3xx or `<meta>`	13.26%	48.07%	10.92%	41.54%
Javascript	2.39%	18.71%	2.78%	12.55%
Others	84.34%	33.22%	86.29%	45.9%

Finally, we must point out that considering individual metrics, such as the ones described in Figure 3, to differentiate between benign and malicious redirections may not be enough. For instance, as mentioned in Section II, benign sites may use syndicated advertisements which may cause longer HTTP redirection paths, so we cannot only use the chain length feature to differentiate the two classes. Hence, in the following subsection, we will describe the complete set of features that we use in our system.

B. Feature Selection and the Classifier

Building on the observations from the previous section, we select a small subset of features from a large pool that we tried. The features selected are the most distinctive ones to differentiate between benign and malicious redirection chains. In addition, as we have seen above, if the attacker manipulates these features (e.g. make the path length shorter), he or she will be more likely to be caught by other mechanisms. We now describe the list of selected features:

- **Number of Different Domains (F1)**. The number of different domains in a path. We consider two domains different if the top two-level domains are different, taking into consideration *c*ountry codes and other well known Top Level Domain names (TLDs). For example, `x.a.com` is the same domain as `y.a.com`, and `a.co.uk` is not the same domain as `b.co.uk` since the `co.uk` is a TLD in our list. This happens to be a good feature since in typical benign CDN or advertisement scenarios the number of different domains involved is small, and we found that in malicious cases, the number of domains involved in a single path is large.
- **Number of HTTP 3xx Redirections (F2)**. The number of edges in the path that are labeled as HTTP 3xx redirections.
- **Number of Different Domain HTTP 3xx Redirections (F3)**. The number of different domain redirections caused by HTTP 3xx headers. We examine all edges in the path, and if the edge is labeled as HTTP 3xx redirection and both the parent and child nodes are different domains, then we count this edge.
- **Number of Consecutive HTTP 3xx Redirections (F4)**. The number of consecutive edges in a path that are labeled as HTTP 3xx redirections.
- **Number of Consecutive Different Domain HTTP 3xx Redirections (F5)**. The number of consecutive edges labeled as HTTP3xx redirections where redirections happen between different domains. It is a mixture of F1 and F4 and a larger value of it makes the path more suspicious.
- **Number of Consecutive Short Edges (F6)**. Short edges

are transitions that occur in no more than 1 second. A one second threshold provides high confidence that it is not a user generated action. Here we are trying to capture any type of redirection (e.g. 3xx messages, JavaScripts, etc) that occurs in a very short period of time, and the larger this number, the more suspicious the path tends to be.

• **Path Length (F7)**. The number of nodes in a path, leveraging the observation of long malicious paths.

• **Edge Duration (F8)**. This feature captures the distribution for the edges durations in a path. We consider the median, average, min, and max edge durations as distinct values in the feature vector. The intuition is that for malicious cases, HTTP redirections happen in a short period of time, so the elapsed time between URL requests in a path should be very short.

Figure 4 shows the CDF for all eight features in the August2011 dataset (Similar patterns were found for April2012). Each plot has two curves, the solid-dotted line shows the CDF of the feature values for the malicious paths and the intermittent-starred line shows the CDF for the benign paths. For instance, Figure 4(f) shows the CDF for the number of consecutive short edges (< 1 second) in both malicious and benign paths. The x-axis represents the number of short edges and the y-axis represents the CDF value. The figure shows that only 8% of the malicious paths have 0 consecutive short edges, while 96% of the benign paths have 0 consecutive short edges. As shown in the figure, the malicious and benign behaviors are clearly different. But, the tails of the two distributions meet in all cases, which indicates that there may be false positive cases, if we decide solely based on this feature. However, as we will see in Section VI, once we combine all these features in a classifier, the false positive rate is negligible. In addition, we show that each of these features plays an important role in our classification scheme in Section VI-D.

Decision Tree Classifier. After we extract the paths, we want to predict whether they are malicious or benign. We use a decision tree, a supervised classifier, using the features described above. In particular, we used a J48 decision tree classifier, which is a Java implementation of the C4.5 algorithm [17] in the WEKA [18] data mining library. The major advantage of using decision tree lies in the interpretability and human validation of the learned models. Also it is fast and scalable. However, the performance of the classifier can be further improved using more sophisticated classifiers, like Random Forest, Boosted Decision Trees or SVM.

V. Putting Pieces Together: a System Prototype

In this section, we describe the architecture of our system prototype. The architecture consists of four components, as shown in figure 5: the HTTP Parser, the URL Extractor, the Activity Tree Updater, and the Classifier.

Starting from packet-level traces obtained from the network, the *HTTP Parser* is responsible for assembling packets and parsing the HTTP header and payload. To do this, the HTTP parser looks for the `Transfer-Encoding` and `Content-Encoding` fields in the HTTP header of the response. The `Transfer-Encoding` field is used by web servers to indicate to web browsers that the response will be sent in chunks instead of one big chunk. This changes the way the parsing is done since an extra header, which has to be removed, is added to the beginning of every chunk. After assembling the packets, the parser looks for the HTTP `Content-Encoding` header to check if the content of the response is compressed (e.g. in zip format). The content will be uncompressed if required. Finally, using the HTTP `Content-Type` response header and well known file signatures at the beginning of the response payload, in most cases, we are able to determine the file type successfully, which facilitates ignoring binary files in the URL Extractor component.

After the HTTP header and payload are assembled and parsed, they are forwarded to the *URL Extractor*, where URLs are extracted and labeled by the type of the source; i.e. JavaScript embedded URL, `<iframe>` URL, HTTP Location header, etc. We did not implement any dynamic JavaScript analysis,s since dynamic code analysis is often very complex and expensive, and our goal is to be able to cope with the speed of live traffic and identify malicious behavior quickly as discussed in Section VII. The extracted URLs are then sent to the *Activity Tree Updater*, which updates the internal data structures and the browsing activity tree as described earlier in Section III-A. Finally, we construct the feature vector for each path and send it to the *Classifier*, which will take the decision on whether the path is malicious or benign.

VI. Evaluation

In this section, we evaluate the performance of our system and demonstrate its effectiveness. We begin by describing the ground truth (i.e. labeled data), which we use to train and test the classifier. Next, we present our classification results, and dissect the main sources of false positives in our datasets. Finally, we show the relative importance for each individual feature we use for classification.

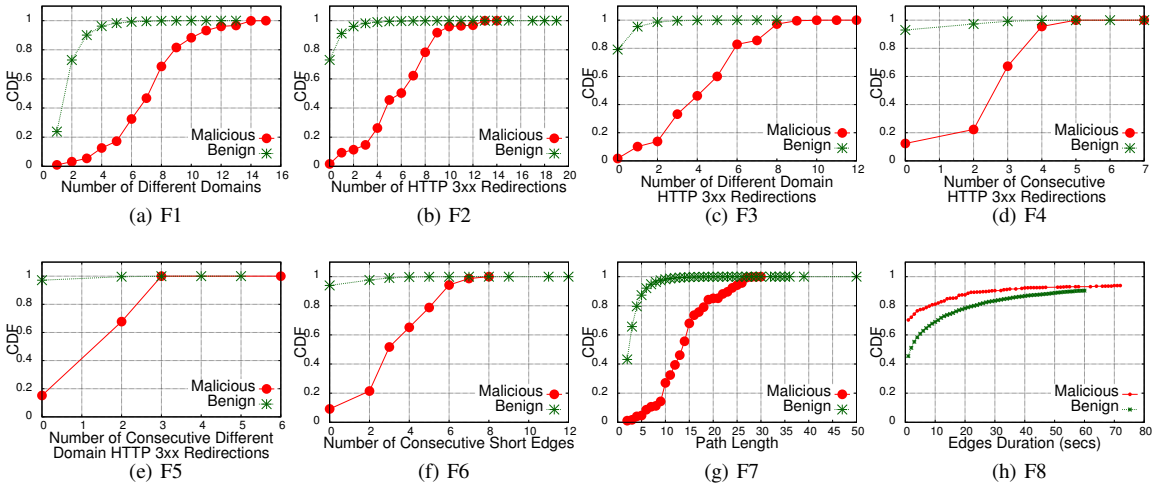
A. Ground Truth

In this section, we discuss the details of the labeling process applied to every path in the dataset.

Malicious Paths. We use a commercial IDS to find malicious HTTP request-response pairs in the dataset. If a redirection path contains at least one node (i.e. an HTTP request) that has been flagged by the IDS, the path is considered malicious. We found a total of 250 clients with malicious paths in August2011 and 300 in April2012 datasets, and then we extract the malicious paths for our evaluation dataset.

Benign Paths. Given that the IDS could miss the identification of malicious HTTP request-response pairs, we follow a series of pre-filtering steps to better identify benign paths. First, we select only a subset of the clients that have no TCP flows labeled as malicious by the IDS. Second, we remove any client who has contacted at least one blacklisted hostname or server IP address using three different blacklists [19]–[21]. We sort the remaining clients in decreasing order of number of flows that contains HTTP 3xx redirections and selected the top 250 clients with the largest number of redirection flows in August2011 and 300 in April2012, so that we have a more uniform set of benign and malicious clients. We extract all the benign paths from these clients for our evaluation dataset.

Finally, our evaluation dataset consists of $12K$ malicious paths and $185K$ benign paths for August2011 and $3K$ malicious paths and $40K$ benign paths for April2012.



(a) F1 (b) F2 (c) F3 (d) F4

(e) F5 (f) F6 (g) F7 (h) F8

Fig. 4. CDF Distributions for the Eight Features.

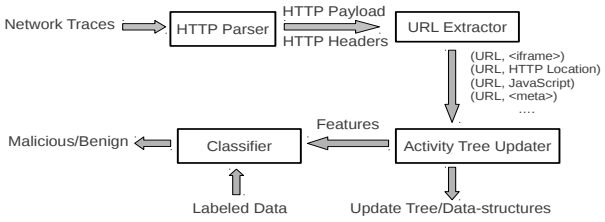


Fig. 5. Architecture of the system prototype.

Threat diversity in the August2011 and April2012 datasets. As mentioned in Section II-C, we focus mainly on two families of threats for evaluation: "Fake AV Attacks" and "Web Attacks". However, we observed that the distribution of the number of malicious HTTP flows in each family is different for the August2011 and the April2012 datasets. This will have important implications for our classifier in Section VI-B. We found that the August2011 dataset is dominated by the threat ID "Fave AV Website", which represent over 90% of the malicious HTTP flows in this dataset (the second highest threat ID is "Mass Iframe Injection" with only 4.8% of the malicious HTTP flows). In contrast, the April2012 has a more diverse set of threats, where the most predominant attack is the threat ID "Web Attack", which represents 30.1% of the malicious HTTP flows, followed threat ID "Web Iframe Injection", with 25.8% of the attacks and threat ID "Mass Injection Web", with 16.9% of the attacks. The rest of the malicious flows in both datasets are distributed across many other threats.

B. Classification Results

In this section, we evaluate the decision tree classifier described in Section IV-B. To build the classifier, we extract the features described in Section IV-B, from the evaluation dataset. Then, we apply the classifier to the dataset and analyze the false positives. We use two different evaluation methods in order to evaluate the classifier's performance: 10-fold cross-validation (CV) and 60% percentage split, on each of the two datasets. In 10-fold CV, the data is randomly splitted into 10 folds, where the classifier is trained on 9-folds and tested on the remaining fold. The classifier repeats this process 10 times, each time a different fold is used for testing. In 60% percentage split, the data is randomly splitted into two pieces; 60% of the data for training the classifier and the other 40% for testing.

TABLE IV
CLASSIFICATION RESULTS

	August2011 CV	August2011 60% Split	April2012 CV	April2012 60% Split
True Positives (TP)	11,147	4,388	2,511	1,018
True Negatives (TN)	185,728	74,340	40,734	16,272
False Positives (FP)	60	24	120	42
False Negatives (FN)	139	78	250	114
FP Rate $FP/(FP+TN)$	0.0003	0.0003	0.0029	0.0025
Accuracy $(TP+TN)/(P+N)$	0.999	0.998	0.991	0.991
Precision $TP/(TP+FP)$	0.995	0.994	0.954	0.960
Recall $TP/(TP+FN)$	0.987	0.982	0.910	0.900

August2011 Dataset Results. Table IV presents the confusion matrix results for both 10-fold CV and 60% percentage split in columns two and three, respectively. We show absolute values of true positives, true negatives, false positives and false negatives in the first four rows. We also show the derived measures false positive rate, accuracy, precision and recall in the last four rows. The table shows that, all the measures of accuracy, precision and recall present values of over 98.2%. Hence, these results show that our classifier, trained with the carefully selected set of features, is very effective in classifying paths as malicious or benign. Furthermore, we tested the classifier on a set of 200 clients (not used before in training or testing), and the classifier shows FP Rate of 0.1%.

April2012 Dataset Results. We apply the same methodology described above to the April2012 dataset. Table IV presents the confusion matrix results in columns four and five. The table shows that our precision is over 95% and recall over 90% for both 10-fold CV and 60% percentage split experiments. We notice a small difference in recall with respect to the previous dataset results. We found that this is due to a larger diversity of threat families detected in the April2012 dataset compared to the August2011 dataset as described in Section VI-A. Hence,

when the dataset is splitted for evaluation, we miss flows from threat families that the classifier does not train on. This would not happen in real life, where the classifier could use all the labelled data for training. But the result still shows that even in this situation, our algorithm is able to perform very well in terms of recall and precision.

Browsing Activity Trees vs. Google Safe Browsing. Next, we compare the performance of Google Safe Browsing (GSB) and our approach. For this, we consider a scenario where we train our classifier on some data provided by an IDS, but then we let the classifier running by itself (without the IDS). Then, we can compare the case where we use only our classifier or only GSB to identify malicious paths. To emulate this, from our 60% percentage split experiment for the April2012 dataset, we take the 40% of the dataset that was used for testing and run the corresponding URLs through GSB. Results show that while we were able to obtain a recall of 90%, as shown in Table IV, GSB could only detect 0.2% of the paths (i.e. at least one URL in the path was identified as malicious by GSB). We believe that since GSB does heavy weight analysis on samples of web sites [1], it does not visit them very often and it may miss many malicious cases that are not sampled. On the other hand, our system is running at the edge of a network and it can capture web sites as they are requested by clients, which gives a more up-to-date view of the malicious cases. Hence, we see our system as complementary to GSB, where chains detected as malicious can be reported to GSB to issue deeper analysis on them.

C. Digging Deeper into "False Positives"

In this section, we dig into the false positives (FPs) obtained from the classifier to gain a better understanding into possible classes of malicious paths that we may have missed in our ground truth data. Note that in the August2011 dataset, the number of false positives is 60 and this number is similarly small in the April2012 dataset. We manually checked each path incorrectly classified as malicious. We extracted the URLs from each path and searched for information from various sources, such as WOT [22], ThreatExpert [23] and VirusTotal [24]. If we found a hit on one of the previous sources, the corresponding path was then a true malicious case, otherwise it remained as a FP. We found the following major categories of FP: *b*enign (paths that are FP), *benign advertisements* (Ads) (paths that include known Ad web sites and are FPs), *m*alicious (paths where one or more URLs are reported as malicious), *m*alicious Ads (paths that include known Ad web sites and where one or more URLs are reported as malicious) and *s*uspicious (paths that include one or more host-names with low WOT scores but there was no clear proof of being malicious). Table V summarizes the results of these categories. The True False Positives are paths that are benign, benign Ads, or suspicious. The True FP Rate is the FP Rate after removing malicious paths from the FPs. The results show that we can maintain a very low FP Rate.

For August2011, we found that the 11 benign Ad cases, use Ad syndication, where the advertiser syndicates part of its ad area to another advertiser and hence, the values of the extracted features for these paths get very close to the malicious class. Even when this even happens very rarely in our evaluation, we envision that our system could incorporate in our scheme some of the features described in [2] to specifically identify malicious Ad sites. We also found 22 malicious Ad cases. For example, one of the FP paths included domains such as `ybrant-search.com`, which was blacklisted due to hosting malicious Ads during 2011, and in addition, WOT [22] showed a very poor reputation for it. Finally, we found 16 malicious (non-Ad) cases. Table V show similar results for April2012.

TABLE V
FALSE POSITIVES CATEGORIZATION

Category	August2011	April2012
Benign	3 (5%)	20 (17.07%)
Benign Advertisements	11 (18.3%)	23 (19.51%)
Malicious	16 (26.6%)	34 (28.05%)
Malicious Advertisements	22 (36.6%)	28 (23.17%)
Suspicious	8 (13.3%)	15 (12.2%)
True False Positives Count (TFP)	22 (36.6%)	58 (48.3%)
True FP Rate = TFP / (TFP + TN)	**0.0001**	**0.0014**

TABLE VI
FEATURE RANKING

Rank	Relative Importance	Feature
1	100%	(F3) # of Different Domain HTTP 3xx Redirections
2	94%	(F6) # of Consecutive Short Edges
3	89%	(F8) Edge Duration
4	80%	(F7) Path Length
5	55%	(F4) # of Consecutive HTTP 3xx Redirections
6	40%	(F1) # of Different Domains
7	34%	(F5) # of Consecutive Different Domain HTTP 3xx Redirections
8	21%	(F2) Number of HTTP 3xx Redirections

D. Feature Importance

To evaluate the significance of the individual features listed in section IV-B, we used the RuleFit [25] algorithm, which uses linear combinations of simple rules for building the prediction model. Then, the relative importance of each individual feature given by Rulefit, reflects its presence in rules that are more influential in the prediction model.

Table VI shows the ranking for the set of features used by our model. The most influential feature is the *number of different domain redirections*, which is expected since normally the number of affiliates involved in a path will be small, and hence a malicious path tends to have a large number of redirections between different domain names. The second most important feature is the *number of consecutive short edges*, which captures the number of consecutive redirections, either JavaScript or HTTP, occurring in a very short period of time. It is expected to see this feature at the top of the list since it captures HTTP and JavaScript redirections spanning short periods of time and occurring consecutively.

VII. DISCUSSIONS AND FURTHER IMPROVEMENTS

Deployment Scenarios: In this paper we demonstrate that it is feasible to employ a machine learning (ML) framework to detect malicious HTTP redirection chains, and develop a simple prototype to evaluate its efficacy. We envision that a "complete" malicious HTTP redirection detection system based on our framework can be installed on a client site (e.g., co-located with a firewall and an IDS system). The ML

classifier obtained from our framework can be also adapted and implemented as a browser plug-in to detect malicious HTTP redirections based on partially observed chains, alert users of potential malicious attacks and block the browser from issuing further HTTP redirections. A third possibility is to implement the system as a cloud service, perhaps in conjunction with the browser plug-in discussed earlier, where user browsers report a sequence of potentially malicious URLs visited to the cloud service, where the cloud service can either test the whole path on the classifier or it can try to do early detection and test the piece of the path that it has seen so far, since our methodology does not prevent the classifier from doing so. The advantage of the cloud-based service is that it would be able to gather a large collection of potentially suspicious HTTP redirection chains from widely dispersed vantage points, and thus can better train the ML classifiers over time. The (new) classification rules can then be fed back to user browsers as updates to protect them against evolving threats.

Integration with Existing Solutions: Our framework rely on IDS labels to train the classifier. Hence it is meant to augment IDS instead of replacing it. However, comparing with running an IDS on all incoming/outgoing network traffic at a client site, our system is far more *lightweight*, as we only need to periodically run the IDS on a small portion of traffic, e.g., on a set of random selected HTTP redirection chains, to generate labeled training data and train the ML classifier over time. Clearly, the effectiveness of the classifier will hinge on the number and variety of training samples. However, our framework is general in the sense that our system can easily incorporate labels or samples obtained from sources, e.g., Google Safe Browsing, samples obtained from infected user machines or honeynets, etc. Furthermore, as our ML classifier is trained based on *statistical* properties that are inherent to malicious HTTP redirection chains (as opposed to signatures generated from malicious web content or malware), our system can potentially detect novel, zero-day threats. As shown earlier, many of the "false positive" alerts generated by our system are in fact suspicious, if not malicious; however, they are *not* identified by the commercial IDS. In addition, our framework also detects malicious HTTP redirections where none of the web sites (URLs) in the chains have been identified by Google Safe Browsing as malicious. In fact, we can feed the alerts generated by our classifier back to IDS or other detection systems to selectively perform deep packet inspection or deep content analysis to identify new threats and generate new signatures. Furthermore, by running our system at client sites (in user browsers, client machines or client networks), we can circumvent the cloaking techniques (e.g., anti-emulation techniques) employed by attackers while protecting them from targeted attacks. Our system also avoids the need to develop *de-obfuscation* mechanisms to untangle obfuscated Javascripts, as it simply relies on the URLs generated by such scripts through passive network monitoring. In addition, when running on the client side (e.g., inside the browser), our system can also be adapted to detect malicious HTTPS traffic.

VIII. Conclusion

In this paper, we developed a machine learning-based methodology for identifying malicious HTTP redirections by examining a set of features associated with them, and showed that by utilizing these features, it is possible to accurately classify HTTP redirections into malicious and benign. In addition, we showed novel findings on new threats not identified by a commercial IDS as well as a new stealthy method that attackers employ to infect vulnerable clients, namely, the delivery of unsolicited content through traffic aggregators used in porn sites. The evaluation results show that our scheme can achieve both recall and precision of over 90% and up to 98%. Currently, we are working on a Firefox add-on that implements our algorithm for online detection of malicious redirections.

Acknowledgment

We are thankful to Marco Mellia for providing the datasets that made this work possible.

References

[1] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose, "All your iFRAMEs Point to Us," in *USENIX Security*, 2008.

[2] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang, "Knowing Your Enemy: Understanding and Detecting Malicious Web Advertising," in *CCS*, 2012.

[3] M. Rajab, L. Ballard, P. Mavrommatis, N. Provos, and X. Zhao, "The Nocebo Effect on the Web: An Analysis of Fake Anti-Virus Distribution," in *LEET*, 2010.

[4] J. Ma, L. Saul, S. Savage, and G. Voelker, "Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs," in *SIGKDD*, 2009.

[5] B. Feinstein and D. Peck, "Caffeine Monkey: Automated Collection, Detection and Analysis of Malicious JavaScript," in *Black Hat*, 2007.

[6] P. Likarish, E. Jung, and I. Jo, "Obfuscated Malicious JavaScript Detection using Classification Techniques," in *Malware*, 2009.

[7] C. Seifert, I. Welch, and P. Komisarczuk, "Identification of Malicious Web Pages with Static Heuristics," in *ATNAC*, 2008.

[8] J. Nazario, "PhoneyC: A Virtual Client Honeypot," in *LEET*, 2009.

[9] Y. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King, "Automated Web Patrol with Strider Honeymonkeys," in *NDSS*, 2006.

[10] A. Moshchuk, T. Bragin, S. Gribble, and H. Levy, "A Crawler-based Study of Spyware on the Web," in *NDSS*, 2006.

[11] K. Chellapilla and A. Maykov, "A Taxonomy of JavaScript Redirection Spam," in *AIRWeb*, 2007.

[12] D. Wang, S. Savage, and G. Voelker, "Cloak and Dagger: Dynamics of Web Search Cloaking," in *CCS*, 2011.

[13] L. Lu, R. Perdisci, and W. Lee, "Surf: detecting and measuring search poisoning," in *CCS*, 2011.

[14] D. Canali, M. Cova, G. Vigna, and C. Kruegel, "Prophiler: A Fast Filter for the Large-scale Detection of Malicious Web Pages," in *WWW*, 2011.

[15] L. Lu, R. Perdisci, and W. Lee, "WarningBird: Detecting Suspicious URLs in Twitter Stream," in *NDSS*, 2012.

[16] Alexa, http://www.alexa.com, Nov 2012.

[17] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

[18] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, *The WEKA Data Mining Software*. SIGKDD Explorations, 2009.

[19] M. Domains, "http://www.malwaredomainlist.com/," Sep 2012.

[20] PhishTank, "http://www.phishtank.com/," Sep 2012.

[21] ZeuS Tracker, "https://zeustracker.abuse.ch/," Sep 2012.

[22] Web of Trust, WOT, http://www.mywot.com/, Sep 2012.

[23] Automated Threat Analysis, http://www.threatexpert.com, Nov 2012.

[24] VirusTotal URL Scanner, https://www.virustotal.com/, Nov 2012.

[25] J. H. Friedman and B. E. Popescu, "Predictive Learning via Rule Ensembles," *The Annals of Applied Statistics*, pp. 916–954, 2008.