THE UNIVERSITY OF TEXAS AT SAN ANTONIO, COLLEGE OF BUSINESS

Working Paper SERIES

WP # 0003MSS-432-2013

Date February 21, 2013

Cross-Layer Detection of Malicious Websites

Li Xu
Dept. of Computer Science
UNIVERSITY OF TEXAS AT SAN ANTONIO

Zhenxin Zhan
Dept. of Computer Science
UNIVERSITY OF TEXAS AT SAN ANTONIO

Shouhuai Xu
Dept. of Computer Science
UNIVERSITY OF TEXAS AT SAN ANTONIO

Keying Ye
Dept. of Statistics
UNIVERSITY OF TEXAS AT SAN ANTONIO

Keesook Han
Information Directorate
Air Force Research Laboratory

Frank Born
Information Directorate
Air Force Research Laboratory

Cross-Layer Detection of Malicious Websites*

Li Xu
Dept. of Computer Science
UT San Antonio
lxu@cs.utsa.edu

Zhenxin Zhan
Dept. of Computer Science
UT San Antonio
zzhan@cs.utsa.edu

Shouhuai Xu
Dept. of Computer Science
UT San Antonio
shxu@cs.utsa.edu

Keying Ye
Dept. of Statistics
UT San Antonio
keying.ye@utsa.edu

Keesook Han
Information Directorate
Air Force Research Laboratory
Keesook.Han@rl.af.mil

Frank Born
Information Directorate
Air Force Research Laboratory
Frank.Born@rl.af.mil

ABSTRACT

Malicious websites have become a major attack tool of the adversary. There are two main approaches to detect malicious websites: *static* and *dynamic*. The static approach is centered on the static analysis of website contents and can scale up to a large number of websites in cyberspace. However, this approach has limited success in dealing with sophisticated attacks that include obfuscation. The dynamic approach is centered on the analysis of website contents via their run-time behaviors, and can cope with these sophisticated attacks. However, this approach is often expensive and cannot scale up to the magnitude of the number of websites in cyberspace. This research aims to achieve the best performance of two malicious website detection approaches simultaneously. In this paper, we propose an analysis of the corresponding network-layer traffic between the browser and the web server by incorporating the static analysis of website contents, which is conducted at the application layer. The insight of this approach is that the network-layer may expose useful information about malicious websites from a different perspective. Evaluation based on the data collected during 37 days shows that certain cross-layer detection methods can be almost as effective as the dynamic approach. Performance experiments show that, when both approaches are deployed as a service, the cross-layer detection approach is about 50 times faster than the dynamic approach.

Categories and Subject Descriptors

K.6.5 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Security and Protection—*Invasive software*

General Terms

Security

*Approved for Public Release; Distribution Unlimited

CODASPY'13, February 18–20, 2013, San Antonio, Texas, USA.

Keywords

Malicious URL, Cross-layer detection, static analysis, dynamic analysis, hybrid analysis

1. INTRODUCTION

Malicious websites have become a severe cyber threat because they can cause the automatic download and execution of malware in browsers, and thus compromise vulnerable computers [32]. The phenomenon of malicious websites will persevere in the future because we cannot prevent websites from being compromised or abused. For example, Sophos Corporation has identified the percentage of malicious code that is hosted on hacked sites as 90% [6]. Often the malicious code is implanted using SQL injection methods and shows up in the form of an embedded file. In addition, stolen ftp credentials allow hackers to have direct access to files, where they can implant malicious code directly into the body of a web page or again as an embedded file reference. Yet another powerful adversarial technique is obfuscation [36], which is very difficult to cope with. These attacks are attractive to hackers because the hackers can exploit them to better hide the malicious nature of these embedded links from the defenders.

Existing approaches to detect malicious websites can be classified into two categories:

- The *static* approach aims to detect malicious websites by analyzing their URLs [25, 26] or their contents [39]. This approach is very efficient and can scale up to deal with the huge population of websites in cyberspace. This approach however has limited success in coping with the aforesaid sophisticated attacks, and can cause high false-negative rates by classifying malicious websites as benign ones.

- The *dynamic* approach aims to detect malicious websites by analyzing their run-time behaviors using Client Honeypots or their like [38, 28, 40, 4, 3]. This approach is very effective. However, it is resource-consuming because it runs or emulates the browser and possibly the operating system [9], and thus cannot scale up to deal with the large number of websites in cyberspace.

How can we achieve the best of the static and dynamic approaches simultaneously? A simple solution is to run a front-end static analysis tool that aims to rapidly detect suspicious websites, which are then examined by a back-end dynamic analysis tool. However, the effectiveness of this approach is fundamentally limited by the assumption that the front-end static analysis tool has a very low false-negative rate; otherwise, many malicious websites will not be

examined by the back-end dynamic analysis tool. Unfortunately, static analysis tools often incur high false-negative rates, especially when malicious websites are equipped with the aforesaid sophisticated techniques. In this paper, we propose a novel technique by which we can simultaneously achieve almost the same effectiveness of the dynamic approach and the efficiency of the static approach. The core idea is to exploit the network-layer or cross-layer information that somehow exposes the nature of malicious websites from a different perspective.

1.1 Our Contributions

We propose an analysis of the corresponding network-layer traffic between the browser and the web server by incorporating the static analysis of website contents. The insight of this approach is that the network-layer may expose useful information about malicious websites from a different perspective. The cross-layer detection is further coupled with the trick of statically tracing redirects, which are embedded into the websites to hide the actual websites that disseminate malwares. That is, the redirection URLs are not obtained via dynamic analysis, but obtained by slightly extending the static analysis method. This allows us to consider not only redirection related features of the present website, but also the redirection website contents.

Evaluation of our approach is based on real data that is collected during the span of 37 days. We find that cross-layer detection can be almost as effective as the dynamic approach and almost as efficient as the static approach, where effectiveness is measured via the vector of (*detection accuracy*, *false-negative rate*, *false-positive rate*). For example, using the dynamic approach as effectiveness base, our data-aggregation cross-layer classifier achieves (99.178%, 2.284%, 0.422%), while the application-layer classifier only achieves (96.394%, 6.096%, 2.933%). Moreover, the XOR-aggregation cross-layer classifier can achieve (99.986%, 0.054%, 0.003%), while resorting only 0.014% of the websites to the dynamic approach. We also discuss the deployment issues of the cross-layer detection approach. Since performance experiments in Section 4.4 show that the cross-layer detection can be 50 times faster than the dynamic approach when processing a batch of URLs, the cross-layer detection is very suitable for deployment as a service. Moreover, the cross-layer detection incurs no more than 4.9 seconds for processing an individual URL, whereas the dynamic approach takes 20 seconds to process a URL on average. This means that the cross-layer detection would be acceptable for real-time detection.

1.2 Related Work

Both the industry and academia are actively seeking effective solutions to the problem of malicious websites. The industry has mainly offered their proprietary blacklists of malicious websites, such as Google's Safe Browsing [1] and Mcafee's SiteAdvisor [5]. Effectiveness of the blacklist approach is fundamentally limited by the frequency the blacklists are updated and disseminated. This justifies why we advocate pursuing light-weight real-time detection, which is the goal of the present paper.

Researchers have used logistic regression to study phishing URLs [18], which does not consider the issue of redirection. On the other hand, redirection has been used as an indicator of web spams [8, 29, 41, 33]. Kurt et al. [39] presented a system for scalably detecting spam contents. Ma et al. [25, 26] studied how to detect phishing and spams based on URLs themselves.

In terms of detecting malicious websites that may host malwares, Choi et al. [11] investigated the detection of malicious URLs, and Canali et al. [9] presented the design and implementation of a static detection tool called Prophiler. However, all these studies did not consider the usefulness of cross-layer detection. On the other hand, the back-end system for deeper analysis is also an active research topic [13, 10, 42, 27], because attackers have been attempting to circumvent dynamic analysis [23, 35].

The rest of the paper is organized as follows. Section 2 describes our cross-layer data collection and analysis methodology. Section 3 investigates two single-layer detection systems. Section 4 presents our cross-layer detection systems. Section 5 explores the deployment of cross-layer detection systems. Section 6 discusses the limitation of the present study and future research directions. Section 7 concludes the present paper.

2. METHODOLOGY

We now describe the methodology underlying our study, including data collection, data pre-processing, evaluation metrics and data analysis methods. The methodology is general enough to accommodate single-layer analyses, but will be extended slightly to accommodate extra ideas that are specific to cross-layer analyses.

2.1 Data Collection

In order to facilitate cross-layer analysis and detection, we need an automated system to collect both the application-layer website contents and the corresponding network-layer traffic. The architecture of our automated data collection system is depicted in Figure 1. At a high level, the data collection system is centered on a crawler. The crawler takes a list of URLs as input, automatically fetches the website contents by launching HTTP/HTTPS requests and tracks the redirects that are identified from the website contents (elaborated below). The crawler also uses the URLs, including the input URL and the detected redirection URLs, to query the DNS, Whois, and Geographic services. This collects information about the registration dates of websites and the geographic locations of the URL owners/registrants. The application-layer website contents and the *corresponding* network-layer IP packets are recorded separately (where the IP packets are caused by the application-layer activities), but are indexed by the input URLs to facilitate cross-layer analysis.

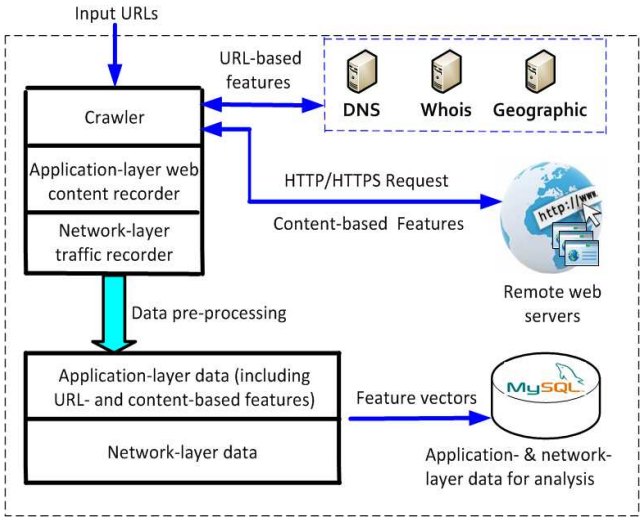


Figure 1: Data collection system architecture.

As mentioned above, the data collection system proactively tracks redirects by analyzing the website contents in a static fashion. Specif-

ically, it considers the following four types of redirects. The first type is the server side redirects, which are initiated either by server rules (i.e., `.htaccess` file) or by server side page code such as `PHP`. These redirects often utilize HTTP 300 level status codes. The second type is JavaScript-based redirects. The third type is the refresh Meta tag and the HTTP refresh header, which allow to specify the URLs of the redirection pages. The fourth type is the embedded file redirects. Some examples of this type are the following: `<script src='badsite.php'> </script>`, `<iframe src='badsite.php'/>`, and `<img src='badsite.php'/>`.

The input URLs may consist of malicious and benign websites. A URL is malicious if the corresponding website content is malicious or any of its redirects leads to a URL that corresponds to malicious content; otherwise, it is benign. In this paper, the terms *malicious URLs* and *malicious websites* are used interchangeably. In our experimental system for training and testing detection models, malicious URLs are initially obtained from the following blacklists: `compuweb.com/url-domain-bl.txt`, `malware.com.br`, `malwaredomainlist.com`, `zeustracker.abuse.ch` and `spyeyetracker.abuse.ch`. Since some of the blacklisted URLs are not accessible or malicious any more, we use the high-interactive client honeypot called Capture-HPC version 3.0 [38] to identify the subset of URLs that are still accessible and malicious. To be concrete, our experiments were based on Capture-HPC, which is assumed to offer the ground truth. This is a practical choice because we cannot manually analyze the large number of websites. Even if we can, manual analysis might be still error-prone. Note that any dynamic analysis system (e.g., another client honeypot system) can be used instead in a plug-and-play fashion. Pursuing a client honeypot that truly offers the ground truth is an orthogonal research problem. The benign URLs are obtained from `alexa.com`, which lists the top 2,088 websites that are supposed to be well protected. The data was collected for a period of 37 days between 12/07/2011 and 01/12/2012, with the input URLs updated daily.

2.2 Data Pre-Processing

Each input URL has an associated application-layer raw feature vector. The features record information such as HTTP header fields, information returned by DNS, Whois and Geographic services, information about JavaScript functions that are called in the JavaScript code embedded into the website content, information about redirects (e.g., redirection method, whether or not a redirect points to a different domain, and the number of redirection hops). Since different URLs may lead to different numbers of redirection hops, the raw feature vectors may not have the same number of features. In order to facilitate analysis, we use a pre-processing step to aggregate multiple-hop information into some *artificial* single-hop information. Specifically, for numerical data, we aggregate them by using their average instead; for boolean data, we aggregate them by taking the `OR` operation; for nominal data, we only consider the final destination URL of the redirection chain. For example, suppose the features of interest are: (`Content-Length`, "Does JavaScript function `eval()` exist in the code?", `Country`). Suppose an input URL is redirected twice to reach the final destination URL, and the raw feature vectors corresponding to the input, first redirect, and second redirect URLs are (100, `FALSE`, US), (200, `FALSE`, UK), and (300, `TRUE`, RUSSIA), respectively. We aggregate the three raw features into a single feature (200, `TRUE`, RUSSIA). After the pre-processing step, the application-layer data have 105 features, some of which will be elaborated below.

Each input URL has an associated network-layer raw feature vector. The features are extracted from the corresponding PCAP (Packet CAPture) files that are recorded when the crawler accesses the URLs. There are 19 network-layer features that are derived from the IP, UDP/TCP or flow level, where a flow is uniquely identified by a tuple (source IP, source port number, destination IP, destination port number, protocol).

Each URL is also associated with a cross-layer feature vector, which is simply the concatenation of its associated application-layer and network-layer feature vectors.

2.3 Data Description

The resulting data has 105 application-layer features of 4 sub-classes and 19 network-layer features of 3 sub-classes. Throughout the paper, "average" means the average over the 37-day data.

2.3.1 Application-Layer Features

Feature based on the URL lexical information.

We defined 15 features based on the URL lexical information, 3 of which are elaborated below.

(A1): `URL_Length`. URL consist of serval parts: protocol, domain name or plain IP address, optional port, directory file, and when using HTTP Get to request information from a server, a question mark that is followed by a list of "$key = value$" pairs. In order to make malicious URLs hard to blacklist, malicious URLs often include automatically and dynamically generated long random character strings. Our data showed that the average length of benign URLs is 18.23 characters, whereas the average length of malicious URLs is 25.11 characters.

(A2): `Number_of_special_characters_in_URL`. This is the number of special characters (e.g., ?, -, _, =, %) that appear in a URL. Our data showed that benign URLs used on average 2.93 special characters, whereas malicious URLs used on average 3.36 special characters.

(A3): `Presence_of_IP_address_in_URL`. This feature indicates whether an IP address is presented as the domain name in a URL. Some websites use IP addresses instead of domain names in URL often because the IP addresses represent the compromised computers that actually do not have registered domain names. This explains why this feature may be indicative of malicious URLs. This feature has been used in [9].

Features based on the HTTP header information.

We defined 15 features based on the HTTP header information, 4 of which are elaborated below.

(A4): `Charset`. This is the encoding charset of URL in question (e.g., iso-8859-1). It hints the language a website used and the ethnicity of the targeted users of the website. It is also indicative of the nationality of the webpage.

(A5): `HTTPHeader_server`. This is the server field in the http response head. It gives the software information at the server side, such as the webserver type/name and its version. Our data showed that the Top 3 webservers that were abused to host malicious websites are Apache, Microsoft IIS, nginx, which respectively correspond to 322, 97, 44 malicious websites on average. On the other hand, Apache, Microsoft IIS, nginx were abused to host 879, 253, 357 benign websites on average.

(A6): `HTTPHeader_cacheControl`. Four cache control strategies are identified in the websites of our data: no-cache, private, public, and cache with max-age. The average numbers of benign websites that use these strategies are respectively 444, 276, 67, and 397, whereas the average numbers of malicious websites that use these strategies are respectively 99, 46, 0.5, and 23.

(A7): `HTTPHeader_content_length`. This feature indicates the content-length field of a HTTP header in question. For malicious URLs, the value of this field may be manipulated so that it does not match the actual length of the content.

Features based on the host information (include DNS, Whois data).

We defined 7 features based on the host information, 5 of which are elaborated below.

(A8-A9): `Whois_regDate` and `Updated_date`. These two features are closely related to each other. They indicate the dates the webserver was registered and updated with the Whois service, respectively. Our data showed that on average, malicious websites were registered in 2004, whereas benign websites were registered in 2002. We also observed that on average, malicious websites were updated in 2009, about one year earlier than the update dates of 2010 for benign websites .

(A10-A11): `Whois_country` and `Whois_stateProv`. These two features respectively indicate the counter and the location where the website was registered. These two features, together with the aforementioned `charset` feature, can be indicative of the locations of websites. Our data showed that the average numbers of benign websites registered in US, NL, and AU are respectively 618, 523, and 302; whereas the average numbers of malicious websites registered in US, NL, and AU are respectively 152, 177, and 98.

(A12): `Within_domain`. This feature indicates whether or not the destination URL and the original URL are in the same domain. Redirection has been widely used by both benign and malicious websites. From our data, we found that malicious websites are more often redirected to exploit servers that reside in different domains. Specifically, we found that 21.7% malicious websites redirect to different domains, whereas 16.1% benign websites redirect to different domains.

Features based on web content information (includes HTML and Script source code).

We defined 68 content-based features, 7 of which are described as follows.

(A13): `Number_of_Redirect`. This is the total number of redirects embedded into an input URL. It is indicative of malicious URLs because our data showed that on average, malicious URLs have 0.67 redirects whereas benign URLs have 0.43 redirects. Note that this feature is unique at the application layer because it cannot be precisely obtained at the network layer, which cannot tell a redirect from a normal link.

(A14): `Number_of_embedded_external_URLs`. This feature counts the number of URLs that are embedded into the input URL and use external resources (e.g., image, voice and video). This feature can be indicative of malicious URLs because external URLs are often abused by attackers to import malicious content to hacked URLs.

(A15): `Content_length_valid`. This feature checks the consistency between the `HTTPHeader_content_Length` feature value (i.e., the value of the content length field in HTTP header) and the actual length of web content. It is relevant because the content length field could be a negative number, which may cause buffer overflow attacks. This feature has been used in [11].

(A16): `Number_of_long_strings`. This feature counts the number of long strings used in the JavaScript code that is embedded into the input URL. A string is considered long if its length is greater than 50. Because attackers try to encode some shell code into a string and then use heap-overflow to execute that shell code, this feature can be indicative of malicious URLs as suggested in [9]. Our data showed that the average `Number_of_long_strings` is 0.88 for malicious URLs and 0.43 for benign URLs.

(A17-A18): `Number_of_iframe` and `number_of_small_size_iframe`. These two features respectively count how many iframe and small size iframes are present in a webpage. If any iframe contains malicious code, the URL is malicious. Small size iframe is even more harmful because it imports malicious content that is invisible to the users.

(A19): `Number_of_suspicious_JavaScript_functions`. This feature indicates whether or not the JavaScript code is obfuscated. We check suspicious JavaScript functions in both the script block and the imported JavaScript files such as `eval()`, `escape()`, and `unescape()`. These JavaScript functions are often used by attackers to obfuscate their code and bypass statical analysis. For example, `eval()` can be used to dynamically execute a long string at runtime, where the string can be the concatenation of many dynamic pieces of obfuscated substrings at runtime. This will make them hardly detected by statical analysis. This feature has been used in [21]

2.3.2 Network-Layer Features

Features based on remote server attributes.

(N1): `Tcp_conversation_exchange`. This is the total number of TCP packets sent to the remote server by the crawler. Malicious websites often use rich web resources that may cause multiple HTTP requests sent to webserver. Our data showed the average `Tcp_conversation_exchange` is 73.72 for malicious websites and 693.38 for benign websites.

(N2): `Dist_remote_TCP_port`. This is the total number of distinct TCP ports that the remote webserver used during the conversation with the crawler. Our data showed that benign websites often use the standard http port 80, whereas malicious websites often use some of the other ports. Our data showed the average `Dist_remote_TCP_port` is 1.98 for malicious websites and 1.99 for benign websites.

(N3): `Remote_ips`. This is the number of distinct remote IP addresses connected by the crawler, not including the DNS server IP addresses. Multiple remote IP addresses can be caused by redirection, internal and external resources that are embedded into the webpage corresponding to the input URL. Our data showed the average `Remote_ips` is 2.15 for malicious websites and 2.40 for benign websites.

Features based on crawler-server communication.

(N4): `App_bytes`. This is the number of Bytes of the application-layer data sent by the crawler to the remote webserver, not including the data sent to the DNS servers. Malicious URLs often cause the crawler to initiate multiple requests to remote servers, such as multiple redirections, iframes, external links to other domain names. Our data showed the average `App_bytes` is 36818 bytes for malicious websites and 53959 bytes for benign websites.

(N5): `UDP_packets`. This is the number of UDP packets generated during the entire lifecycle when the crawler visits an URL, not including the DNS packets. Benign websites with online streaming application (such as video, audio and internet phone) will generate lots of UDP packets, whereas malicious websites often incur lots of TCP packets. Our data showed the average `UDP_packets` for both benign and malicious URLs are 0 because the crawler does not download any video/audio stream from the sever.

(N6): `TCP_urg_packets`. This is the number of urgent TCP packets with the URG (urgent) flag set. Some attacks abuse this flag to bypass the IDS or firewall systems that are not properly set up. If

a packet has the URGENT POINTER field set, but the URG flag is not set, this constitutes a protocol anomaly and usually indicates a malicious activity that involves transmission of malformed TCP/IP datagrams. Our data showed the average is 0.0003 for malicious websites and 0.001 for benign websites.

(N7): `Source_app_packets`. This the number of packets send by the crawler to remote servers. Our data showed the average `source_app_packets` is 130.65 for malicious websites and 35.44 for benign websites.

(N8): `Remote_app_packets`. This the number of packets send by the the remote webserver(s) to the crawler. This feature is unique to the network layer. Our data showed the average value of this feature is 100.47 for malicious websites and 38.28 for benign websites.

(N9): `Source_app_bytes`. This is the volume (bytes) of the crawler-to-webserver communications. Our data showed that the average application payload volumes of benign websites and malicious websites are about 146 Bytes and 269 Bytes, respectively.

(N10): `Remote_app_bytes`. This is the volume (bytes) of data from the webserver(s) to the crawler, which is similar to feature `Source_app_byte`. Our data showed the average value of this feature is 36527 bytes for malicious websites and 49761 bytes for benign websites.

(N11): `Duration`. This is the the duration of time, starting from the point the crawler was fed with an input URL to the point the webpage was successfully obtained by the crawler or an error returned by the webserver. This feature is indicative of malicious websites because visiting malicious URLs may cause the crawler to send multiple DNS queries and multiple connections to multiple web servers, which could lead to a high volume of communications. Our data showed that visiting benign websites causes 0.793 seconds duration time on average, whereas visiting malicious websites causes 2.05 seconds duration time on average.

(N12): `Avg_local_pkt_rate`. This is the average rate of IP packets (packets per second) that are sent from the crawler to the remote webserver(s) with respect to an input URL, which equals to `source_app_packets/duration`. This feature measures the packet sending speed of the crawler, which is related to the richness of webpage resources. Webpages containing rich resources often cause the crawler to send large volume of data to the server. Our data showed the average `Avg_local_pkt_rate` is 63.73 for malicious websites and 44.69 for benign websites.

(N13): `Avg_remote_pkt_rate`. This is the average IP packets rate (packets per second) sent from the remote server to the crawler. When multiple remote IP addresses are involved (e.g., because of redirection or because of the webpage using external links), we amortize the number of packets to them, despite that some remote IP addresses may send more packets than others back to the crawler. Websites containing malicious code or contents can cause large volume communications between the remote server(s) and the crawler. Our data showed the average `Avg_remote_pkt_rate` rate is 63.73 for malicious websites and is 48.27 for benign websites.

(N14): `App_packets`. This is the total number of IP packets generated for obtaining the content corresponding to an input URL, including redirects and DNS queries. It measures the data exchange volume between the crawler and the remote webserver(s). Our data showed the average value of this feature is 63.73 for malicious websites and 48.27 for benign websites.

Features based on crawler-DNS flows.

(N15): `DNS_query_times`. This is the number of DNS queries sent by the crawler. Because of redirection, visiting malicious URLs often causes the crawler to send multiple DNS queries and to connect multiple remote webservers. Our data showed the average value of this feature is 13.30 for malicious websites and 7.36 for benign websites.

(N16): `DNS_response_time`. This is the response time of DNS servers. Benign URLs often have longer life-times and their domain names are more likely cached at local DNS servers. As a result, the average value of this feature of benign URLs may be shorter. Our data showed the average value of this feature is 13.29 ms for malicious websites and are 7.36 ms for benign websites.

Features based on aggregated values.

(N17): `Iat_flow`. This is the accumulated inter-arrival time between consecutive flows. Given two consecutive flows, the inter-arrival time is the difference between the timestamp of the first packet in each flow. Our data showed the average `Iat_flow` is 1358.4 for malicious websites and 512.99 for benign websites.

(N18): `Flow_number`. This is the number of flows generated during the entire lifecycle for the crawler to download the web content corresponding to an input URL, including the recursive queries to DNS and recursive access to redirects. It includes both TCP flows and UDP flows, and is a more general way to measure the communications between the crawler and the remote webservers. Each resource in the webpage may generate a new flow. This feature is also unique to the network layer. Our data showed the average `Flow_number` is 19.48 for malicious websites and 4.91 for benign websites.

(N19): `Flow_duration`. This is the accumulated duration of each basic flow. Different from feature `Duration`, this feature indicates the linear process time of visiting an URL. Our data showed the average `Flow_duration` is 22285.43 for malicious websites and 13191 for benign websites.

2.4 Effectiveness Metrics

In order to compare different detection models (or methods, algorithms), we consider three effectiveness metrics: *detection accuracy*, *false-negative rate*, and *false-positive rate*. Suppose we are given a detection model (e.g., J48 classifier or decision tree), which may be learned from the training data. Suppose we are given a test data that consists of d_1 malicious URLs and d_2 benign URLs. Suppose further that the detection model correctly detects d_1' out of the d_1 malicious URLs and d_2' out of the d_2 benign URLs. The detection accuracy is defined as $\frac{d_1'+d_2'}{d_1+d_2}$. The false-negative rate is defined as $\frac{d_1-d_1'}{d_1}$. The false-positive rate is defined as $\frac{d_2-d_2'}{d_2}$. A good detection model achieves high effectiveness (i.e., high detection accuracy, low false-positive and false-negative rate).

2.5 Data Analysis Methods

In order to identify the better detection model, we consider four popular machine learning algorithms: Naive Bayes, Logistic regression, Support Vector Machine (SVM) and J48. Naive Bayes classifier is a probabilistic classifier based on Bayes' rule [22]. Logistic regression classifier [24] is one kind of linear classification, where the domain of the target variable is $0, 1$. SVM classifier aims to find an maximum-margin hyperplane for separating different classes in the training data [12]. We use the SMO (Sequential Minimal-Optimization) algorithm in our experiment with polynomial kernel function [31]. J48 classifier is an implementation of C4.5 decision trees [34] for binary classification. These algorithms have been implemented in the Weka toolbox [19], which also resolves issues such as missing feature data and conversion of strings to numbers.

In order to know whether using a few features is as powerful as using all features and which features are more indicative of ma-

licious websites, we consider the following three feature selection methods. The first method is Principle Component Analysis (PCA), which transforms a set of feature vectors to a set of shorter feature vectors [19]. The second feature selection method is called "CfsSubsetEval with best-first search method" in the Weka toolbox [19], or Subset for short. It essentially computes the features' prediction power according to their contributions [20]. It outputs a subset of features, which are substantially correlated with the class but have low inter-feature correlations. The third feature selection method is called "InfoGainAttributeEval with ranker search method" in the Weka toolbox [19], or InfoGain for short. Its evaluation algorithm essentially computes the information gain ratio (or more intuitively the importance of each feature) with respect to the class. Its selection algorithm ranks features based on their information gains [14]. It outputs the ranks of all features in the order of decreasing importance.

3. SINGLE-LAYER DETECTION OF MALICIOUS WEBSITES

In this section, we investigate two kinds of single-layer detection systems. One uses the application-layer information only, and corresponds to the traditional static approach. The other uses the network-layer information only, which is newly introduced in the present paper. The latter was motivated by our insight that the network layer may expose useful information about malicious websites from a different perspective. At each layer, we report the results obtained by using the methodology described in Section 2.

The application-layer and network-layer effectiveness results averaged over the 37 days are described in Table 1. For application-layer detection, we make two observations.

- J48 classifier is significantly more effective than the other three detection models, whether feature selection is used or not. However, J48 classifiers may incur somewhat high false-negative rates.
- Feature selection will significantly hurt detection effectiveness, which is true even for J48 classifiers. This means that conducting feature selection at the application layer does not appear to be a good choice.

For network-layer detection, we observe the following:

- J48 classifier is significantly more effective than the other three detection models, whether feature selection is used or not. Note that although Naive Bayes incurs a lower false-negative rate, it has a very low detection accuracy. Similar to what we observed at the application layer, J48 classifier also incurs pretty high false-negative rates, meaning that network-layer alone is not competent.
- Overall, feature selection hurts detection effectiveness. This also means that conducting feature selection at the network layer is not a good idea.

By comparing the application layer and the network layer, we observed two interesting phenomena. First, each single-layer detection method has some inherent limitation. Specifically, since we were somewhat surprised by the high false-negative and false-positive rates of the single-layer detection methods, we want to know whether they are caused by some outliers (extremely high rates for some days), or are persistent over the 37 days. By looking into the data in detail, we found that the false-negative and false-positive rates are reasonably persistent. This means that single-layer detection has some inherent weakness.

Second, we observe that network-layer detection is only slightly less effective than application-layer detection. This confirms our original insight that the network-layer traffic data can expose useful information about malicious websites. Although network-layer detection alone is not good enough, this paved the way for exploring the utility of cross-layer detection of malicious websites, which is explored in Section 4.

4. CROSS-LAYER DETECTION OF MALICIOUS WEBSITE

Having showed that network-layer traffic information can give approximately the same detection effectiveness of the application layer, now we show how cross-layer detection can achieve much better detection effectiveness. Given the pre-processed feature vectors at the application and network layers, we extend the preceding methodology slightly to accommodate extra ideas that are specific to cross-layer detection.

- Data-aggregation cross-layer detection: For a given URL, we obtain its cross-layer feature vector by concatenating its application-layer feature vector and its network-layer feature vector. The resultant feature vectors are then treated as the pre-processed data in the methodology described in Section 2 for further analysis.
- OR-aggregation cross-layer detection: For a given URL, if either the application-layer detection model or the network-layer detection model says the URL is malicious, then the cross-layer detection model says the URL is malicious; otherwise, the cross-layer detection model says the URL is benign. This explains why we call it OR-aggregation.
- AND-aggregation cross-layer detection: For a given URL, if both the application-layer detection model and the network-layer detection model say the URL is malicious, then the cross-layer detection model says the URL is malicious; otherwise, the cross-layer detection model says the URL is benign. This explains why we call it AND-aggregation.
- XOR-aggregation cross-layer detection: For a given URL, if both the application-layer detection model and the network-layer detection model say the URL is malicious, then the cross-layer detection model says the URL is malicious; if both the application-layer detection model and the network-layer detection model say the URL is benign, then the cross-layer detection model says the URL is benign. Otherwise, the cross-layer detection model resorts to the dynamic approach. That is, if the dynamic approach says the URL is malicious, then the cross-layer detection model says the URL is malicious; otherwise, the cross-layer detection model says the URL is benign. We call it XOR-aggregation because it is in the spirit of the XOR operation.

We stress that the XOR-aggregation cross-layer detection model resides in between the above three cross-layer detection models and the dynamic approach because it partly relies on the dynamic approach. XOR-aggregation cross-layer detection is practical *only when* it rarely invokes the dynamic approach.

4.1 Overall Effectiveness of Cross-Layer Detection

The effectiveness of cross-layer detection models, averaged over the 37 days, is described in Table 2, from which we observe the

Feature selection?	Naive Bayes			Logistic			SVM			J48		
	Acc (%)	FN (%)	FP (%)	Acc (%)	FN (%)	FP (%)	Acc (%)	FN (%)	FP (%)	Acc (%)	FN (%)	FP (%)
application-layer average detection effectiveness												
none	51.260	11.029	59.275	90.551	22.990	5.692	85.659	55.504	3.068	96.394	6.096	2.933
PCA	67.757	9.998	38.477	91.495	20.526	5.166	89.460	30.031	5.189	95.668	9.537	2.896
Subset	77.962	35.311	18.162	86.864	37.895	6.283	84.688	51.671	5.279	93.581	15.075	3.999
InfoGain	71.702	19.675	30.664	84.895	43.857	7.097	83.733	52.071	6.363	94.737	12.148	3.390
network-layer average detection effectiveness												
none	51.767	0.796	61.645	90.126	21.531	6.630	86.919	24.449	9.986	95.161	9.127	3.676
PCA	67.766	4.017	40.278	87.454	30.651	7.520	85.851	32.957	9.346	89.907	22.587	6.604
Subset	70.188	0.625	38.035	88.141	25.629	8.061	86.534	25.397	10.188	92.415	14.580	5.658
InfoGain	55.533	0.824	56.801	86.756	29.783	8.647	82.822	40.875	10.560	92.853	15.442	4.852

Table 1: Single-layer average effectiveness (Acc: detection accuracy; FN: false negative rate; FP: false positive rate)

following. First, data-aggregation cross-layer J48 classifier without using feature selection achieves (99.178%, 2.284%, 0.422%)-effectiveness, which is significantly better than the application-layer J48 classifier that achieves (96.394%, 6.096%, 2.933%)-effectiveness, and is significantly better than the network-layer J48 classifier that achieves (95.161%, 9.127%, 3.676%)-effectiveness. In other words, cross-layer detection can achieve significantly higher effectiveness than the single-layer detection models. This further confirms our motivational insight that network-layer can expose useful information about malicious websites from a different perspective. This phenomenon can be explained by the low correlation between the application-layer feature vectors and the network-layer feature vectors of the respective URLs. We plot the correlation coefficients in Figure 2, which shows the absence of any correlation because the correlation coefficients fall into the interval of $(-0.4, 0.16]$. This implies that the application layer and the network layer expose different kinds of perspectives of malicious websites, and can be exploited to construct more effective detection models.

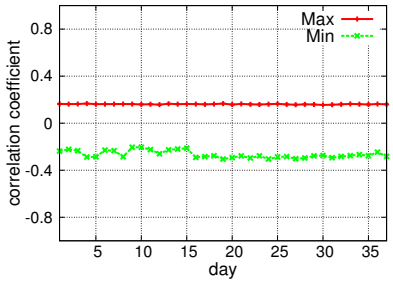


Figure 2: The $\max$ and $\min$ correlation coefficients between application-layer and network-layer feature vectors.

Second, J48 classifier is significantly better than the other three classifiers, with or without feature selection alike. Since the above comparison is based on the average over 37 days, we want to know whether or not J48 classifier is consistently more effective than the other three classifiers. For this purpose, we looked into the data and found that J48 classifier is almost always more effective than the other three classifiers. Therefore, we recommend to use J48 classifier and will focus on J48 classifier in the rest of the paper.

Third, because the preceding discussion is based on the average of the 37 days, it is interesting to know whether the effect of feature selection is persistent over the 37 days. For this purpose, we considered both feature selection algorithms and found that they exhibit similar phenomenon. Specifically, we looked into the day-by-day effectiveness of cross-layer detection models with respect to the InfoGain feature selection algorithm. We found that the effect of feature selection is persistent over the 37 days, especially for the XOR-aggregation cross-layer detection model. This further confirms that feature selection can be adopted in practice.

Fourth, the OR-aggregation cross-layer J48 classifier can achieve significantly lower false-negative rate than the data-aggregation cross-layer J48 classifier, at the price of a lower detection accuracy and a higher false-positive rate; whereas, the AND-aggregation cross-layer J48 classifier can achieve a significantly lower false-negative rate than the data-aggregation cross-layer J48 classifier, at the price of a lower detection accuracy and a higher false-negative rate. This phenomenon can be explained by using the definitions of the effectiveness metrics as follows. For fixed population of d_1 malicious URLs and d_2 benign URLs, a lower false-negative rate $\frac{d_1-d_1'}{d_1}$ implies a higher d_1'. Since the detection accuracy $\frac{d_1'+d_2'}{d_1+d_2}$ slightly decreases when compared with the data-aggregation cross-layer detection, d_2' must decrease. This means that the false-positive rate $\frac{d_2-d_2'}{d_2}$ increases. In a similar fashion, we can deduce that an increase in false-positive rate can lead to a decrease in the false-negative rate. The above phenomenon has a useful implication: cross-layer classifiers offer a spectrum of deployment possibilities, depending on the security needs (e.g., preferring lower false-negative rate or lower false-positive rate). In Section 5, we will explore the deployment issues of the cross-layer detection models.

Fifth, feature selection still hurts the cross-layer detection effectiveness, but at a much lesser degree. Moreover, the data-aggregation cross-layer J48 classifier with feature selection is still significantly better than the single-layer J48 classifiers without using feature selection. Indeed, the data-aggregation cross-layer J48 classifier with feature selection offers very high detection accuracy and very low false-positive rate, the OR-aggregation cross-layer J48 classifier with feature selection offers reasonably high detection accuracy and reasonably low false-negative rate, and the AND-aggregation cross-layer J48 classifier with feature selection offers reasonably high detection accuracy and extremely low false-positive rate. When compared with the data-aggregation cross-layer detection, the OR-aggregation cross-layer detection has a lower false-negative rate, but a lower detection accuracy and a higher false-positive rate. This can be explained as before.

Sixth, the XOR-aggregation cross-layer detection can achieve almost the same effectiveness as the dynamic approach. For example, it achieves (99.986%, 0.054%, 0.003%) effectiveness without using feature selection, while only losing 0.014%(1-99.086%) accuracy to the dynamic approach. This means that J48 classifier is extremely appropriate for XOR-aggregation, which can be deployed in real-life whenever possible. Note that the false-negative rate of the XOR-aggregation J48 classifier equals the false-negative rate of the OR-aggregation J48 classifier. This is because all of the malicious websites which are mistakenly classified as benign by the OR-aggregation J48 classifier are necessarily mistakenly classified as benign by the XOR-aggregation J48 classifier. For a similar rea-

Layer	Feature selection?	Naive Bayes			Logistic			SVM			J48		
		Acc (%)	FN (%)	FP (%)	Acc (%)	FN (%)	FP (%)	Acc (%)	FN (%)	FP (%)	Acc (%)	FN (%)	FP (%)
Cross-layer (data-aggregation)	none	55.245	7.961	55.104	96.861	7.945	1.781	94.568	21.227	1.112	99.178	2.284	0.422
	PCA	72.084	4.124	34.659	97.582	5.740	1.481	96.014	9.330	2.492	98.807	3.007	0.692
	Subset	80.396	1.402	24.729	94.568	13.662	3.129	93.296	15.575	4.244	98.335	4.245	0.945
	InfoGain	73.146	1.342	34.069	90.703	22.267	5.693	88.297	26.562	7.571	97.365	6.052	1.685
Cross-layer (OR-aggregation)	none	40.286	0.162	76.437	91.565	6.116	9.104	88.517	7.858	12.542	97.101	0.054	3.708
	PCA	41.582	0.212	74.707	90.039	7.992	10.529	88.342	19.301	9.919	94.251	1.279	7.010
	Subset	57.666	0.065	54.162	88.493	11.460	11.554	86.958	14.154	12.770	94.263	2.615	6.622
	InfoGain	45.276	0.150	70.051	87.342	12.075	12.851	85.266	18.144	13.802	95.129	1.621	5.794
Cross-layer (AND-aggregation)	none	79.097	8.262	24.502	92.528	33.536	0.202	90.335	44.216	0.142	97.888	9.781	0.003
	PCA	79.918	12.428	22.355	90.437	43.244	0.192	85.642	66.755	0.005	94.524	24.998	0.037
	Subset	88.188	17.355	10.246	88.984	49.660	0.300	86.738	60.510	0.205	95.448	20.508	0.111
	InfoGain	83.719	14.269	16.888	87.625	55.774	0.293	84.313	71.175	0.265	95.496	20.685	0.023
Cross-layer (XOR-aggregation)	none	80.861	0.162	24.502	98.510	6.116	0.202	98.186	7.858	0.142	99.986	0.054	0.003
	PCA	82.552	0.212	22.355	98.103	7.992	0.192	96.052	19.301	0.005	99.693	1.279	0.037
	Subset	91.990	0.065	10.246	97.275	11.460	0.300	96.754	14.154	0.205	99.346	2.615	0.111
	InfoGain	86.803	0.150	16.888	97.140	12.075	0.293	95.822	18.144	0.265	99.630	1.621	0.023

Table 2: Cross-layer average effectiveness (Acc: detection accuracy; FN: false-negative rate; FP: false-positive rate). In the XOR-aggregation cross-layer detection, the portions of websites were queried to the dynamic approach (i.e., the websites for which the application-layer and cross-layer detection models have different opinions) with respect to the four machine learning algorithms are respectively: without using feature selection: (19.139%, 1.49%, 1.814%, 0.014%); using PCA feature selection: (17.448%, 1.897%, 3.948%, 0.307%); using Subset feature selection: (8.01%, 2.725%, 3.246%, 0.654%); using InfoGain feature section: (13.197%, 2.86%, 4.178%, 0.37%). Therefore, J48 classifier is extremely appropriate for XOR-aggregation.

son, we see why the false-positive rate of the XOR-aggregation J48 classifier equals the false-positive rate of the AND-aggregation J48 classifier.

4.2 Which Features Are Indicative?

Identifying the features that are most indicative of malicious websites is important because it can deepen our understanding of malicious websites. Principal Components Analysis (PCA) has been widely applied to obtain unspervised feature selections by using linear dimensionality reduction technique. However, PCA-based feature selection method is not appropriate to discover indicative of malicious websites. Therefore, this research has focused on Subset and InfoGain.

The Subset feature selection algorithm.

This algorithm selects a subset of features with low correlation while achieving high detection accuracy. Over the 37 days, this algorithm selected 15 to 16 (median: 16) features for the data-aggregation cross-layer detection, and 15 to 21 (median: 18) features for both the OR-aggregation and the AND-aggregation. Since this algorithm selects at least 15 features daily, space limitation does not allow us to discuss the features in detail. Nevertheless, we will identify the few features that are also most commonly selected by the InfoGain algorithm.

The InfoGain feature selection algorithm.

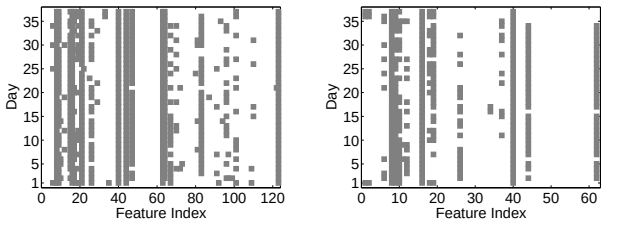
This algorithm ranks the contributions of individual features. For each of the three specific cross-layer J48 classifiers and for each of the 37 days, we used this algorithm to select the 5 most contributive application-layer features and the 4 most contributive network-layer features, which together led to the detection effectiveness described in Table 2. The five most contributive application-layer features are (in descedent order): (**A1**): URL_Length; (**A5**): HTTPHead_server; (**A8**): Whois_regDate; (**A6**): HTTPHead_cacheControl; (**A11**): Whois_stateProv. The four most contributive network-layer features are (also in descendent order): (**N11**): Duration; (**N9**): Source_app_byte; (**N13**): Avg_remote_pkt_rate; (**N2**): Dist_remote_TCP_port.

Intuitively, these features are indicative of malicious websites because during the compromise of browsers, extra communications may be incurred for connecting to the redirection websites while involving more remote TCP ports. We observed that most of the HTTP connections with large (**N11**): Duration time are caused by slow HTTP responses. This is seemingly because malicious websites usually employ dynamic DNS and Fast-Flush service network techniques to better hide from detection. This would also explain why malicious websites often lead to larger values of (**N2**): Dist_remote_TCP_port. We also observed that malicious websites often have longer DNS query time (1.33 seconds on average) than benign websites (0.28 seconds on average). This can be because the DNS information of benign websites are often cached in local DNS servers, meaning there is no need to launch recursive or iterative DNS queries. Moreover, we observe that malicious websites often incur smaller (**N13**): Avg_remote_pkt_rate because the average volume of malicious website contents is often smaller than the average volume of benign website contents. Our datasets show that the average volume of malicious website contents is about 36.6% of the length of benign website contents.

The most commonly selected features.

Now we discuss the features that are most commonly selected by both feature selection algorithms. On each of the 37 days, the Subset feature selection algorithm selected the aforesaid 15-21 features of the 124 features. Overall, many more features are selected by this algorithm over the 37 days. However, only 5 features were constantly selected everyday, where 4 features are from the application layer and 1 feature is from the network layer. Specifically, they are: (**A1**): URL_Length; (**A5**): HTTPHead_server; (**A2**): Number_of_special_characters_in_URL; (**A13**): Number_of_redirects; (**N1**): Duration. These features are indicative of malicious websites because visiting malicious URLs may cause the crawler to send multiple DNS queries and connect to multiple web servers, which could lead to a high volume of communications.

The InfoGain feature selection algorithm selected the aforesaid 15-16 features out of the 124 application-layer and network-layer features. Overall, only 17 out of the 124 features were ever selected, where 6 features are from the application layer and the other 11 features are from the network layer. Three of the afore-



(a) `Subset` feature selection (b) `InfoGain` feature selection

Figure 3: Selected features during the 37 days (features 1-19 correspond to the network-layer features, and features 20-124 correspond to the application-layer features).

said features were selected every day: (**A1**): `URL_Length`, (**N1**): `Duration`, (**N9**): `Source_app_byte`. As mentioned in the description of the `InfoGain` feature selection algorithm, (**N1**): `Duration` represents one important feature of malicious web page. As for (**N9**): `Source_app_byte` feature, intuitively, malicious web pages that contain rich content (usually phishing contents) can cause multiple HTTP requests.

Overall, the features most commonly selected by the two feature selection algorithms are the aforementioned (**A1**): `URL_Length`, (**A5**): `HTTPHead_server` and (**N1**): `Duration`. This further confirms the power of cross-layer detection. These features are indicative of malicious websites as explained before.

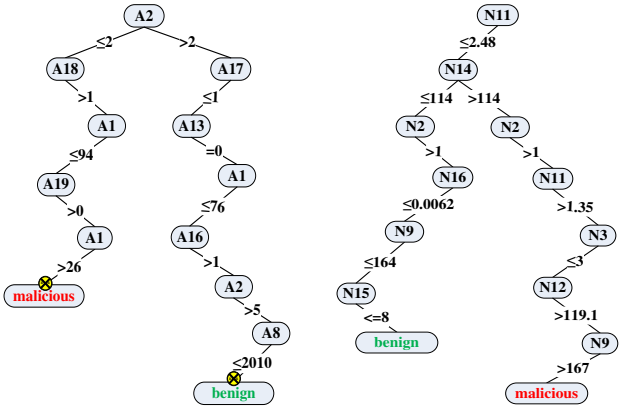
4.3 How Did the Network Layer Help Out?

In the above we observed the overall effectiveness of cross-layer detection, which at a high level can be attributed to the fact that the network-layer data has a low correlation with the application-layer data (i.e., the network-layer data does expose extra information about websites). Now we give a deeper characterization of the specific contributions of the network-layer information that leads to the correct classification of URLs.

Cross-layer aggregation method	Average correction of FN	Average correction of FP
Data-aggregation	79.59	13.91
`OR`-aggregation	126.16	N/A
`AND`-aggregation	N/A	16.23
`XOR`-aggregation	126.16	16.32

Table 3: Breakdown of the average mis-classifications that were corrected by the network-layer classifiers, where N/A means that the network-layer cannot help (see text for explanation).

Table 3 summarizes the average number of "corrections" made through the network-layer classifiers, where average is taken over the 37 days. The mis-classifications by the application-layer classifiers are either false-negative (i.e., the application-layer classifiers missed some malicious URLs) or false-positive (i.e., the application-layer classifiers wrongly accused some benign URLs). Note that for `OR`-aggregation, the network-layer classifiers cannot help correct the FP mistakes made by the application-layer classifiers because the benign URLs are always classified as malicious as long as one classifiers (in this case, the application-layer one) says they are malicious. Similarly, for `AND`-aggregation, the network-layer classifiers cannot help correct the FN mistakes made by the application-layer classifiers because (i) the malicious URLs are always classified as benign unless both kinds of classifiers think they are malicious and (2) the application-layer classifier already says they are benign. We observe that the contributions of the network-layer classifiers for `XOR`-aggregation in terms of correcting both FP and FN (126.16 and 16.32, respectively) are strictly more significant than the contributions of the network-layer information for `data`-aggregation (79.59 and 13.91, correspondingly). This explains why `XOR`-aggregation is more effective than `data`-aggregation.



(a) Two mis-classification examples by application-layer classifier

(b) Network-layer corrections of the two application-layer mis-classifications

Figure 4: Portions of the application-layer and network-layer classifiers corresponding to the two URLs.

In what follows we examine two example URLs that were mis-classified by the application-layer classifier but corrected through the network-layer classifier. The two examples are among the URLs on the first day data, where one example corresponds to the FP mistake (i.e., the application-layer classifier mis-classified a benign URL as malicious) and the other example corresponds to the FN mistake (i.e., the application-layer classifier mis-classified a malicious URL as benign). The portion of the application-layer classifier corresponding to the two example URLs are highlighted in Figure 4(a), which involves the following features (in the order of their appearances on the paths):

(**A2**)	`Number_of_special_char`
(**A18**)	`Number_of_small_size_iframe`
(**A1**)	`URL_length`
(**A19**)	`Number_of_suspicious_JavaScript_functions`
(**A17**)	`Number_iframe`
(**A13**)	`number_of_redirect`
(**A16**)	`Number_of_long_strings`
(**A8**)	`register_date`

The portions of the network-layer classified corresponding to the two URLs are highlighted in Figure 4(b), which involves the following features (in the order of their appearances on the paths):

(**N11**)	`Duration`
(**N14**)	`App_packets`
(**N2**)	`Dist_remote_TCP_port`
(**N16**)	`DNS_response_time`
(**N9**)	`Avg_local_pkt_rate`
(**N15**)	`DNS_query_times`
(**N3**)	`Remote_ips`
(**N12**)	`Source_app_bytes`

Note that some features can, and indeed often, appear multiple times on a single path.

For the FP mistake made by the application-layer classifier, the feature values are **A2**=0 (no special characters in URL), **A18**=2 (two small iframes), **A1**=61 (medium URL length) and **A19**=4 (four

suspicious JavaScript functions), which lead to the left-hand path in Figure 4(a). The application-layer mis-classification may be attributed to **A18**=2 and **A19**=4, while noting that benign websites also use the `eval()` function to dynamically generate code according to certain information about the browser/user and use obfuscation to hide/protect JavaScript source code. On the other hand, the relevant network-layer feature values are **N11**=0.89 seconds (close to 0.793 second, the average of benign URLs), **N14**=79 (close to 63.73, the average of malicious URLs), **N2**=5 (not indicative because it is almost equally close to the averages of both benign URLs and malicious URLs), **N16**=13.11ms (close to 13.29, the average of malicious URLs), **N9**=113 (close to 146, the average of benign URLs), **N15**=6 (close to 7.36, the average of benign URLs). We observe that the three network-layer features, namely **N11**, **N9** and **N15**, played a more important role in correctly classifying the URL.

For the FN mistake made by the application-layer classifier, **A2**=7 (close to 3.36, the average of malicious URLs), **A17**=0 (indicating benign URL because there are no iframes), **A13**=0 (indicating benign URL because there are no redirects), **A1**=22 (close to 18.23, the average of malicious URLs), **A16**=2 (close to 0.88, the average of malicious URLs), and **A8**=2007 (indicating benign URL because the domain name has been registered for multiple years). The above suggests that **A17**, **A13** and **A8** played a bigger role that caused the mis-classification. On the other hand, the relevant network feature values are **N11**=2.13 (close to 2.05, the average of malicious URLs), **N14**=342 (close to 63.73, the average of malicious URLs), **N2**=7 (not very indicative because the respective averages of benign URLs and malicious URLs are about the same), **N3**=3 (close to 2.40, the average of benign URLs), **N12**=289 bytes (relatively close to 63.73, the average of malicious URLs), and **N9**=423 (relatively close to 269, the average of malicious URLs). The above suggests that the network-layer classifier can correct the mistake made by the application-layer classified because of features **N11**, **N14**, **N12** and **N9**.

4.4 Performance Evaluation

As discussed in the Introduction, we aim to make our system as fast and scalable as the static approach while achieving as high of effectiveness as the dynamic approach. In the preceding, we have demonstrated that cross-layer J48 classifiers (indeed, all of the cross-layer detection models we investigated) are almost as effective as the dynamic approach. In what follows we report that the cross-layer J48 classifiers are much faster than the dynamic approach and almost as efficient as the static approach.

The time spent on running our system consists of three parts: the time spent for collecting application-layer and network-layer data, the time spent for training the cross-layer J48 classifiers, and the time spent for using the J48 classifiers to classify websites. Since the training of cross-layer J48 classifiers is conducted periodically (e.g., once a day in our experiments), this time is not a significant factor and can be omitted. Nevertheless, we report that the time spent for learning data-aggregation cross-layer J48 classifiers is typically less than 10 seconds on a modest computer when the training dataset has thousands of feature vectors. The training time spent for learning `OR`-aggregation, `AND`-aggregation, or `XOR`-aggregation cross-layer J48 classifiers is about the same. Therefore, we will focus on the time spent for collecting the application-layer and network-layer data corresponding to a given URL and the time spent for classifying the given URL. These two metrics are the most important because they ultimately determine whether the cross-layer J48 classifiers can be deployed for the purpose of real-time detection.

In the afore-reported effectiveness experiments, the cross-layer J48 classifiers and the Capture-HPC client honeypot (as example of the dynamic approach) ran on different computers with different hardware configurations. Therefore, we cannot simply measure and compare their respective time complexities. In order to have a fair comparison, we conducted extra experiments by using two computers with the same configuration. One computer ran our cross-layer J48 classifiers and the other computer ran the Capture-HPC client honeypot. The hardware of the two computers is Intel Xeon X3320 4 cores CPU and 8GB memory. We use Capture-HPC version 3.0.0 and VMWare Server version 1.0.6. The Host OS is Windows Server 2008 and the Guest OS is Windows XP sp3. Our crawler was written in JAVA 1.6 and ran on top of Debian 6.0. We used IPTABLES [2] and a modified version of TCPDUMP [7] to parallelize the data collection system. The application-layer features are directly obtained by each crawler instance, but the network-layer features are extracted from the network traffic that is collected by the TCPDUMP software on the local host. IPTABLES are configured to log network flow information with respect to different processes, which correspond to different crawler instances. Since our crawler is light-weight, we ran 50 instances concurrently in our experiments; whereas we ran 5 guest Operating Systems to parallelize the Capture-HPC. Experimental results indicated that more guest Operating Systems make the system unstable. Both computers use network cards with 100Mbps network cable.

Data-aggregation cross-layer J48 classifier	
Total data collection time	4 min
Total classification time	302 ms
Total time	≈ 4 min
Capture-HPC	
Total time	199min

Table 4: Measured performance comparison between the data-aggregation cross-layer J48 classifier and the dynamic approach (the Capture-HPC client honeypot) with 3,062 input URLs (1,562 malicious URLs + 1,500 Benign URLs)

Table 4 describes the performance of the cross-layer J48 classifier and of the Capture-HPC client honeypot. It took the data-aggregation cross-layer J48 classifier about 4 minutes to process the 3,062 input URLs, whereas it took the Capture-HPC 199 minutes to process the same 3,062 URLs. In order words, the cross-layer detection approach can be about 50 times faster than the dynamic approach, while achieving about the same detection effectiveness.

The preceding conclusion that the cross-layer detection approach can be about 50 times faster than the dynamic approach was based on the batch processing of 3,062 URLs. In order to get a clue on the performance comparison in terms of the processing time for individual URLs, we can approximately break down the measure performance as follows, where *approximation* is caused by the concurrent executions of the respective systems. Specifically, the time for the data-aggregation cross-layer J48 classifier to determine whether a given website is malicious or not may be calculated as: $240/(3062/50) \approx 3.92$ seconds because each crawler actually processed 3062/50 URLs on average. Among the 3.92 seconds, on average 2.73 seconds were actually spent for downloading the website content, which means that 1.19 seconds were spent for feature extractions etc. Similarly, the time for Capture-HPC to determine whether a given website malicious or not is $(199 \times 60)/(3062/5) = 19.5$ seconds because 5 Capture-HPC instances run concurrently. The reason why Capture-HPC is slow is because Capture-HPC spent much time on receiving all the diagnostic re-

sults caused by visiting URLs in virtual machine and reverting virtual machine back to clean snapshot whenever a URL is deemed as malicious. Moreover, the XOR-aggregation cross-layer J48 classifier without using feature selection would only incur the dynamic approach to analyze, on average, about $5.04\% \times 3062 \approx 154$ website. This means that even for XOR-aggregation, the processing time per URL is no more than $3.92 + 19.5 \times 154/3062 \approx 4.9$ seconds. Therefore, we conclude that even if the cross-layer detection system runs within each individual computer, rather than a third-party server, it is about 4 times faster than the dynamic approach. In any case, 4 seconds waiting time is arguably acceptable, especially, we can let the browser start displaying the portions of website content that have no security concerns. This is reasonable because the same idea has been used to give users the illusion that website contents are displayed almost instantly, but actually it takes a few seconds to display the entire website contents. On the other hand, waiting for 19.5 seconds for the dynamic approach to test whether a website is malicious or not is not usable, which perhaps explains why the dynamic approach, while powerful, is not used for real-time detection in practice.

5. DEPLOYMENT

Cross-layer detection offers a spectrum of deployment options. On one hand, it can be deployed as a stand-alone solution because it is highly effective as analyzed before. Moreover, it can be deployed as a light-weight front-end detection system of a bigger solution (see Figure 5), which aims at detecting as many malicious websites as possible while scaling up to a large population of websites. For this purpose, the data-aggregation and the OR-aggregation method would be competent. Moreover, the XOR-aggregation is particularly effective and should be deployed when it only incurs the back-end dynamic approach occasionally.

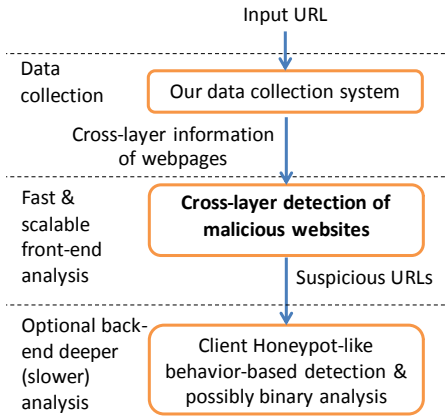


Figure 5: Example deployment of the cross-layer detection system as the front-end of a bigger solution because XOR-aggregation J48 classifiers achieve extremely high detection accuracy, extremely low false-negative and false-positive rates.

On the other hand, there are several ways to deploy the physical components of the cross-layer detection service. Recall that our system has three components: application-layer data collector (i.e., crawler), network-layer traffic recorder, and cross-layer data correlator. The crawler takes URLs as input, fetches the corresponding website contents, and conducts a light-weight analysis to identify the redirects that are embedded into the website contents. The traffic recorder collects the network traffic corresponding to the crawler's activities for fetching the website contents. The cross-layer data correlator relates the application-layer website contents to the corresponding network-layer traffic via the input URLs. These components may or may not be deployed on the same physical computer, as the following scenarios demonstrate.

First, we can deploy the stand-alone cross-layer detection system as a web browser plug-in. In this case, the detection system can test whether the website is malicious or not before the browser actually displays the website content. If it is malicious, the browser can take appropriate actions according to a pre-determined policy (e.g., warning the user that the website is malicious). The plug-in should collect the network-layer traffic corresponding to the application-layer website content of the given URL. The plug-in also may act as the network-layer traffic collector and the cross-layer correlator. Moreover, network-traffic could be collected at some routers or gateways, from which the plug-in can get the traffic corresponding to the application-layer website content.

Second, we can deploy the cross-layer detection system as an online service. This service may be accessed by web browsers via the proxy or gateway technique. Specifically, when a user browser points to a URL, the corresponding website will be analyzed by the cross-layer detection service, which will inform the outcome back to the browser. The browser can take appropriate actions based on its pre-determined policy (e.g., displaying the website or not).

Third, we can deploy the cross-layer detection system by the website hosting server itself. The website hosting service vendor might have the incentive for proactively examining whether the websites it hosts have been compromised, because this might enhance the reputation of the vendor. In this case, the vendor can deploy it as a front-end to a bigger detection system, or deploy it as a stand-alone system.

6. LIMITATION AND FUTURE WORK

First, a key limitation of the study is that the (back-end) dynamic approach itself may have its own non-zero false-negative and false-positive rates. This issue has been noticed by few studies except [23, 35], but more systematic studies are needed before making firm conclusions. While studying the dynamic approach is an orthogonal issue, we plan to study the impact of the false-negative and false-positive of the dynamic approach, with an emphasis on the Capture-HPC that is used in the present paper.

Second, it is interesting to know to what extent we can improve the effectiveness of cross-layer detection systems by incorporating new techniques such as those described in [13, 37, 30, 15].

Third, our cross-layer detection system provides some simple best-effort capability by statistically tracking the redirects that are embedded into the website contents. It is notoriously difficult to statistically detect obfuscated JavaScript-based redirects [17, 16]. Even though the effectiveness of our cross-layer detection system is almost as good as the dynamic approach, it is very interesting to know the impact of any progress made in the direction of detecting obfuscated JavaScript-based redirects. This is important because, although our collected data hints that JavaScript-based redirection is widely used by malicious websites, it appears that JavaScript obfuscation may not have been widely used because our system can effectively detect the malicious URLs (almost as effective as the dynamic approach which is capable of dealing with directs). However, this may not be true in the future because in the future such redirects may be exploited by the adversary much more widely. Fortunately, any progress in dealing with obfuscated directs can be adopted by our system in a plug-and-play fashion.

7. CONCLUSION

We presented a novel approach to detecting malicious websites based on the insight that network-layer traffic data may expose useful information about websites, which may be exploited to attain cross-layer detection of malicious websites. Experimental results showed that cross-layer detection can achieve almost the same detection effectiveness, but about 50 times faster than, the dynamic approach based on client honeypot systems. Moreover, the cross-layer detection systems can also be deployed to detect malicious website in real time because the average time for processing a website is approximately 4.9 seconds, which could be improved with some engineering optimization.

8. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments, and our shepherd, Ninghui Li, for his guidance.

9. REFERENCES

[1] Google safe browing, `http://code.google.com/apis/safebrowsing/developers_guide_v2.html`.

[2] iptables 1.4.12.1 `www.netfilter.org/projects/iptables`.

[3] Know your enemy: Behind the scenes of malicious web servers. `http://www.honeynet.org`.

[4] Know your enemy: Malicious web servers. `http://www.honeynet.org`.

[5] Mcafee siteadvisor. `http://www.siteadvisor.com`.

[6] Sophos corporation. security threat report update 07/2008. `http://sophos.com/sophos/docs/eng/papers/sophos-security-report-jul08-srna.pdf`.

[7] tcpdump 4.2.0 `www.tcpdump.org`.

[8] A. A. Benczur, K. Csalogany, T. Sarlos, and M. Uher. Spamrank - fully automatic link spam detection. In *AIRWeb'05*, 2005.

[9] D. Canali, M. Cova, G. Vigna, and C. Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. WWW'11, pages 197–206. ACM, 2011.

[10] K. Z. Chen, G. Gu, J. Nazario, X. Han, and J. Zhuge. WebPatrol: Automated collection and replay of web-based malware scenarios. In *ASIACCS'11*, 2011.

[11] H. Choi, B. B. Zhu, and H. Lee. Detecting malicious web links and identifying their attack types. In *WebApps'11*, pages 11–11, 2011.

[12] C. Cortes and V. Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.

[13] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *WWW'10*, 2010.

[14] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, NY, USA, 1991.

[15] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert. Zozzle: Fast and precise in-browser javascript malware detection. In *USENIX Security*, 2011.

[16] A. Dewald, T. Holz, and F. C. Freiling. Adsandbox: Sandboxing javascript to fight malicious websites. In *SAC'10*, pages 1859–1864, 2010.

[17] B. Feinstein and D. Peck. Caffeine monkey: Automated collection, detection and analysis of malicious javascript. In *Black Hat'07*, 2007.

[18] S. Garera, N. Provos, M. Chew, and A. D. Rubin. A framework for detection and measurement of phishing attacks. In *WORM'07*, pages 1–8, 2007.

[19] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, pages 10–18, 2009.

[20] M. A. Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.

[21] Y.-T. Hou, Y. Chang, T. Chen, C.-S. Laih, and C.-M. Chen. Malicious web content detection by machine learning. *Expert Syst. Appl.*, pages 55–60, 2010.

[22] G. H. John and P. Langley. Estimating continuous distributions in Bayesian classifiers. *UAI*, pages 338–345, 1995.

[23] A. Kapravelos, M. Cova, C. Kruegel, and G. Vigna. Escape from monkey island: Evading high-interaction honeyclients. In *DIMVA'11*, Amsterdam, The Netherlands, July 2011.

[24] S. le Cessie and J. van Houwelingen. Ridge estimators in logistic regression. *Applied Statistics*, pages 191–201, 1992.

[25] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond blacklists: Learning to detect malicious web sites from suspicious urls. In *KDD'09*, pages 1245–1254, 2009.

[26] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying suspicious urls: an application of large-scale online learning. In *ICML'09*, pages 681–688, 2009.

[27] T. F. Mario Heiderich and T. Holz. Iceshield: Detection and mitigation of malicious websites with a frozen dom. In *RAID'11*, 2011.

[28] J. Nazario. Phoneyc: A virtual client honeypot, 2009.

[29] Y. Niu, Y. min Wang, H. Chen, M. Ma, and F. Hsu. A quantitative study of forum spamming using contextbased analysis. In *NDSS'07*, 2007.

[30] W. Palant. Javascript deobfuscator 1.5.8. `https://addons.mozilla.org/en-US/firefox/addon/javascript-deobfuscator/`, 2010.

[31] J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods – Support Vector Learning*, pages 42–65. 1998.

[32] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iframes point to us. In *USENIX Security*, 2008.

[33] Z. Qian, Z. M. Mao, Y. Xie, and F. Yu. On network-level clusters for spam detection. In *NDSS'10*, 2010.

[34] R. Quinlan. *C4.5: Programs for Machine Learning*. San Mateo, CA, 1993.

[35] M. Rajab, L. Ballard, N. Jagpal, P. Mavrommatis, D. Nojiri, N. Provos, and L. Schmidt. Trends in circumventing web-malware detection. Technical report, Google, 2011.

[36] K. Rieck, T. Krueger, and A. Dewald. Cujo: efficient detection and prevention of drive-by-download attacks. In *ACSAC'10*, pages 31–39, 2010.

[37] C. S. The ultimate deobfuscator. `http://securitylabs.websense.com/content/Blogs/3198.aspx`.

[38] C. Seifert and R. Steenson. Capture - Honeypot Client (Capture-HPC). `https://projects.honeynet.org/capture-hpc`, 2006.

[39] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and evaluation of a real-time url spam filtering service. In *S&P'11*, 2011.

[40] Y.-M. Wang, D. Beck, X. Jiang, and R. Roussev. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *NDSS'06*, 2006.

[41] C. Whittaker, B. Ryner, and M. Nazif. Large-scale automatic classification of phishing pages. In *NDSS'10*, 2010.

[42] J. Zhang, C. Seifert, J. W. Stokes, and W. Lee. Arrow: Generating signatures to detect drive-by downloads. In *WWW'11*, pages 187–196, 2011.