User-Assisted Host-Based Detection of Outbound Malware Traffic [⋆]

Huijun Xiong[1], Prateek Malhotra[1], Deian Stefan[2], Chehai Wu[3], and Danfeng Yao[1]

[1] Department of Computer Science, ,Rutgers University Piscataway, NJ 08854, USA
[2] Department of Electrical Engineering, The Cooper Union, New York, NY 10003, USA
[3] AppFolio, Inc. 55 Castilian Dr. Goleta, CA 93117, USA
huijun@cs.rutgers.edu, someone1@eden.rutgers.edu,
stefan@cooper.edu, wuchehai@gmail.com, danfeng@cs.rutgers.edu

Abstract. Conventional network security solutions are performed on network-layer packets using statistical measures. These types of traffic analysis may not catch stealthy attacks carried out by today's malware. We aim to develop a host-based security tool that identifies suspicious outbound network connections through analyzing the user's surfing activities. Specifically, our solution for Web applications predicts user's network connections by analyzing Web content; unpredicted traffic is further investigated with the user's help. We describe our method and implementation as well as the experimental results in evaluating its efficiency and effectiveness. We describe how our studies can be applied to detecting bot infection. In order to assess the workload of our host-based traffic-analysis tool, we also perform a large-scale characterization study on 500 university-users' wireless network traces for 4-month period. We study both the statistical and temporal patterns of individuals' web usage behaviors from collected wireless network traces. Users are classified into different profiles based on their web usage patterns. Our results show that users have regularities in their Web activities and the expected workload of our traffic-analysis solution is low.

1 Introduction

Several studies estimate that millions of computers worldwide are infected by malware and have become bots that are controlled by cyber criminals [9, 10, 14]. The infected computers are coordinated and used by the attackers to launch diverse malicious and illegal network activities, including perpetrating identity theft, sending spam (estimated 100 billion spam messages every day [25]), launching denial of service (DoS) attacks, and committing click fraud. Malicious bots are stealthy and difficult to detect using conventional anti-virus software. Botnet communications including command and control (C & C) and attacks disturb the usual and routine traffic patterns of a user. For a good description of the botnet structures, we refer readers to the paper by Dagon, Gu, Lee, and Lee [6]. Malicious bot is a special type of malware, which is an umbrella term for all malicious software such as virus, worm, rootkit, and spyware.

[⋆] This work has been supported in part by NSF grant CCF-0728937, CNS-0831186, and the Rutgers University Computing Coordination Council Pervasive Computing Initiative Grant.

Many network-wide intrusion detection and protection systems, both commercial products or research prototypes, have been developed for monitoring network traffics and report alerts if observing known suspicious attack patterns [24, 26]. Similarly, anomaly detection systems aim to detect deviations or abnormal events from historic usage patterns [22]. However, the existing network analysis and security tools are inadequate in two main aspects: *individualized analysis* and *personalized security*. It is reported that on average 3-5 percent of organizational assets are compromised by bots and malware – even when the best and most up-to-date security software is applied [7]. Most current network trace analysis focuses on the aggregated traffic flow of the entire network, e.g., network-side traffic volume, busiest hosts on the network, and bursty periods of the organization. These types of network-wide traffic analysis do not give insights to the usage patterns of individuals on the network.

In this paper, we describe a novel host-based anomaly detection approach based on both traffic prediction and user participation. We call it a *personalized security* approach. Specifically, we implement a traffic monitoring framework that is capable of *predicting* legitimate outbound network connections. Our framework intercepts network traffic from the host. The connections that are observed but *not* predicted by the framework may be due to malware activities on the host.

In order to further classify the unpredicted outbound connections, our approach is to leverage the user's personal knowledge about his or her Web activities, for example, by prompting a window asking the user whether she initiated a connection to a Web address. Studies found that users demonstrated regularities in their surfing patterns [12]. Our characterization results presented in this paper also indicate that users have highly repetitive network-connection patterns. Utilizing these regularities in realizing user-centric security solutions as we demonstrate in this work opens a new direction for battling against today's pervasive malware attacks. Many botnets successfully evade the IRC (Internet Relay Chat protocol)-based detection by switching to HTTP-based command and control [13, 30], as HTTP traffic is usually allowed through firewalls and not blocked.

Therefore, our study focuses on identifying HTTP traffic of malware. Our approach can be generalized to other application protocols. With a personalized security approach, we monitor and examine host-based traffic patterns to detect abnormal network requests caused by malware. This type of investigations represents a personalized analytical approach that can also be applied to managing the security of large organizations. We implement our traffic-monitoring framework in Python and evaluate its efficiency and prediction effectiveness. The technical challenges involved in predicting Web-related traffic are the diversity and flexibility of hypertexts including scripts. We focus on parsing and analyzing static Web pages, embedded iframes and cascaded style sheets, as well as redirected pages. *Our results indicate that most traffic can be effectively predicted using our code for static Web content*. Legitimate connections that our prediction misses are mainly due to JavaScript code. In an effort to reduce the number of questions asking to the user, we also utilize a whitelisting approach.

Another contribution of this paper is a large-scale characterization study on 500-users' wireless network traces for four-month period. We collected wireless network traces from a university. Our characterization work aims towards discovering the patterns and properties of individuals' network behaviors, in particular, we study both the statistical and temporal patterns of individual host's Web activities. (A host is uniquely identified by its MAC address which remains consistent throughout the dataset.) Our investigation is different from the conventional network-wide aggregated traffic analysis, as we focus on the micro-scale pattern of an individual user. Our characterization results suggest that users have low diversity in terms of daily Web sites visited – people tend to visit a small number of Web sites regularly. This repetitiveness can be leveraged to construct effective host-based malware detection solutions because malware-caused deviations from the regular patterns may be identified. It also indicates that the expected workload of our traffic-monitoring tool is low.

The rest of the paper is organized as follows. Our host-based traffic-monitoring framework is described in Section 2. Our wireless network trace analysis is given in Section 3. The related work is described in Section 4. Conclusions and future work are in Section 5.

2 Outbound Malware-Traffic Detection with User Participation

In this section, we describe a traffic monitoring framework that aims to identify malware traffic by carefully analyzing user's Web requests and content as well as involving the user in the process of classifying traffic. Although detecting anomaly traffic with user's help may appear to be straightforward, the challenge here is how to encourage user's participation and avoid intrusiveness to user. Thus we need a precise and efficient prediction mechanism. Our solution requires the minimal participation from the user and causes no undesirable delays to the user's surfing experience. Our study is focused on Web traffic because HTTP based malware activities such as Spyware or botnet command and control are notorious hard to detect – most firewalls allow HTTP traffic on port 80. Our implementation is realized in Python in Linux, but the architecture can be realized in other programming languages and platforms as well.

Our malware attack model and security assumptions are as follows. We consider stealthy malware that is secretly sending outbound HTTP traffic. The malware may corrupt the browser, e.g., through malicious extensions [16]. Thus, the browser is *not* assumed to be trusted. The malware may be at the application-level or kernel-level such as rootkits which actively hide their presence from the host's operating system. However, for kernel-level malware, we assume that components in our detection framework along with its files are not corrupted by the malware. This last assumption is reasonable, as the integrity of our framework can be ensured using trusted computing infrastructure such as Trusted Platform Module (TPM) [28, 29] that are available on most commodity PCs through a standard attestation procedure [20]. The integration of TPM into our framework is not described in this paper. Our study addresses client-side security, and complements any server-side security solutions.

2.1 System Architecture and Algorithm for Traffic Monitoring

In order to identify unauthorized HTTP connections possibly due to malware, our approach is to monitor and analyze outbound network requests. Blocking outbound malware packets can effectively render malware useless. Thus, we do not need to examine all incoming traffic, which makes our solution all the more efficient. Our solution can effectively eliminate a wide spectrum of harmful malware activities, e.g., identity theft, spam, DDoS attacks, click fraud, or botnet command & control messages. Malware is unable to deliver stolen personal data to the outside. Our framework has the following three main components:

- *Sniffer*: interception and filtering all outbound HTTP requests. Pending outbound HTTP requests are put on a list waiting for approval to execute. Sniffing outbound HTTP requests can be realized using existing network libraries such as `libpcap` library in Python.
- *Predictor*: the prediction of legitimate outbound HTTP requests based on the user's activities and parsing of Web content retrieved *out of band* (Section 2.2). Observed-yet-unpredicted connections will be prompted to the user for further classification. An important requirement to the predictor is to reduce the number of questions for the user while maintaining the prediction accuracy.
- *User interface*: pop-up windows where a user can indicate whether or not observed network attempts are initialized by her (Section 2.3). The interface needs to be easy to use by nontechnical-savvy users. A screenshot of our user interface is shown in Figure 3.

In the next few sections, we will describe the technical details involved in realizing our predictor as well as our experimental evaluation on the framework.

2.2 Analysis on Web Contents

In HTTP protocols, each object is retrieved in a separate HTTP request. For example, if a Web page has 10 images, then the browser issues 11 separate HTTP requests sequentially to the Web server. For persistent HTTP connections, all 11 HTTP requests may be sent in one TCP connection between the server and the client, whereas for nonpersistent HTTP connection, each HTTP request requires a separate TCP connection. Being persistent or not does not affect the deployment of our solution.

To predict legitimate Web traffic, a straightforward solution is that each time an outbound HTTP request is observed, we ask the user whether she is responsible for that connection. However, this simple approach may create many questions and be quite intrusive to users due to the pervasive third-party content on the Web – advertisements or (multimedia) content hosted by content delivery providers instead of the main web server. Third-party content (e.g., from `amakai.com` or `yimg.com`) is retrieved from URLs that may seem arbitrary to the user, i.e., bearing no similarity to the main website URL (e.g., `yahoo.com`), impacting user's classification decisions. This problem is solved by us with out-of-band retrieval and analysis of Web content (explained below). The workflow for identifying suspicious outbound traffic in our solution is as follows.

1. The predictor fetches the requested Web page independent of the browser (e.g., using `wget`), which we call *out-of-band retrieval*. It parses the retrieved content to whitelist the outbound HTTP requests for fetching referenced objects (e.g., images). The predictor repeats this step until there are no referenced objects. The whitelist is stored in memory, and is domain-based to improve our prediction efficiency.

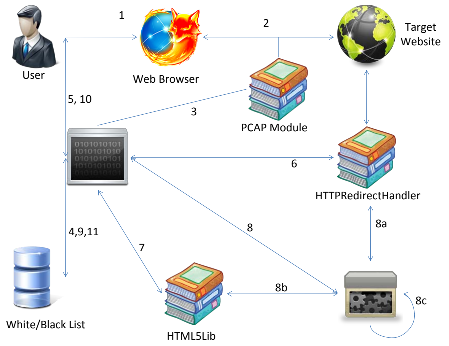


Fig. 1. Workflow in our traffic-monitoring framework.

2. The sniffer intercepts all attempted outbound HTTP connections from the host (including those from applications other than the browser), which are put into a waiting list. The HTTP requests that appear on the predictor's whitelist are permitted. For example, a HTTP GET request to fetch object `yimg.com/images/tree.jpg` is allowed if `yimg.com` is on the whitelist.
3. For the pending connections that cannot be predicted, we prompt a small window to the user asking whether she has initialized that request. The connection is allowed if the user enters *Yes*, and denied otherwise.

A schematic drawing of the detailed workflow regarding our traffic prediction is shown in Figure 1 and explained as follows. Figure 2 gives an example of the objects/connections predicted as a result of a user visiting `www.cs.rutgers.edu`.

1. The user visits a target website W in Step 1 and 2. This initial request to URL W is intercepted by `libpcap` library in Step 3.
2. In Step 4, our predictor checks to see if the domain of W is whitelisted or not. If the domain is blacklisted, then the user is given a warning. If it is whitelisted, the request is allowed. Otherwise, we prompt the user to confirm URL W as shown in Figure 3 in Step 5. User-permitted domains are put onto the whitelist for future reference.
3. In Step 6, the object `HTTPRedirectHandler` is for keeping track of redirected requests by putting a listener to each executed outbound HTTP request. Therefore, our predictor is capable of tracking redirections of any arbitrary depth. We note that our analysis including the sending and processing of HTTP requests is outside and independent of the browser, which we call *out-of-band* analysis.
4. The core parsing and prediction steps are Step 7 and 8 (8a, 8b, and 8c). Content retrieved by the out-of-band request is parsed by the `HTML5Lib` Python library to predict additional HTTP requests for objects therein. For example, in Figure 2 `.CSS` file may contain additional image objects that need to be requested. This process loops (indicated by Step 8c) until there is no more object to retrieve.

5. Unique domain names of predicted connections are put on the whitelist in Step 9. Observed actual connections that do not appear on the whitelist are prompted to the user for classification in Step 10, the results of which are used to update the white/black lists in Step 11.

In our current implementation, our analysis is in parallel with the browser and takes place in a *postmortem* fashion where we aim to identify suspicious URL connections and alert to the user. Therefore, it causes no delay in the user's actual surfing experience. An alternative solution is to suspend all outbound HTTP requests until they are either predicted by our tool or explicitly approved by the user. This approach may cause delays in users' Web usage and is not adopted by us.

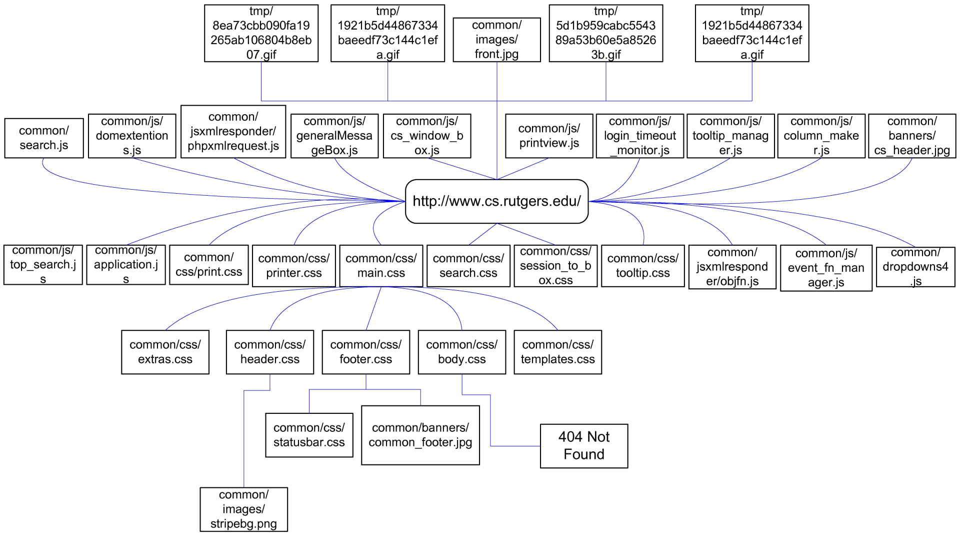
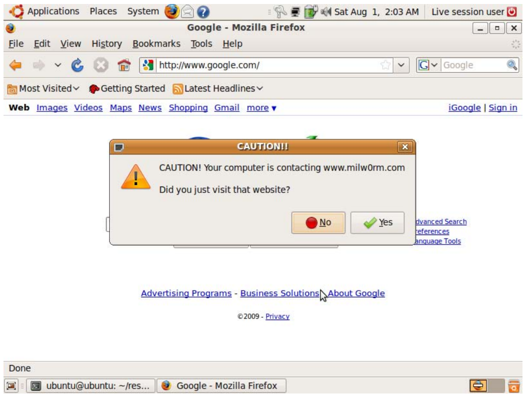

tmp/ 8ea73cbb090fa19 265ab106804b8eb 07.gif
tmp/ 1921b5d44867334 baeedf73c144c1ef a.gif
common/ images/ front.jpg
tmp/ 5d1b959cabc5543 89a53b60e5a8526 3b.gif
tmp/ 1921b5d44867334 baeedf73c144c1ef a.gif
common/ search.js
common/js/ domextention s.js
common/ jsxmlresponder/ phpxmlrequest.js
common/js/ generalMessa geBox.js
common/js/ cs_window_b ox.js
common/js/ printview.js
common/js/ login_timeout _monitor.js
common/js/ tooltip_manag er.js
common/js/ column_make r.js
common/ banners/ cs_header.jpg
http://www.cs.rutgers.edu/
common/js/ top_search.j s
common/js/ application.j s
common/ css/print.css
common/css/ printer.css
common/css/ main.css
common/css/ search.css
common/css/ session_to_b ox.css
common/css/ tooltip.css
common/ jsxmlrespond er/objfn.js
common/js/ event_fn_man ager.js
common/ dropdowns4 .js
common/css/ extras.css
common/css/ header.css
common/css/ footer.css
common/css/ body.css
common/css/ templates.css
common/css/ statusbar.css
common/banners/ common_footer.jpg
404 Not Found
common/ images/ stripebg.png

Fig. 2. The illustration of a tree capturing the hierarchical invoking sequences among (automatic) outbound HTTP requests as a result of visiting www.cs.rutgers.edu.

Because our analysis is completely independent of the browser, it is robust against corrupted browser. For example, a browser with malicious extensions on the client may secretly exporting users' personal data to the attacker (e.g., spyware); our solution can detect the stealthy traffic. Our parser utilizes the `HTML5LIB` Python module. This module provides a robust ability to parse through HTML/XML and XHTML code including those with malformed markup code. The module can automatically fix bad markup and return the parsed data in several formats including a tree format. After we download and parse through an HTML file, our predictor goes through every node in the tree using a built-in tree-traversal mechanism from the HTML5lib module to identify tags with the attribute `SRC`. HTML tag with the attribute `SRC` initiates a network request to fetch the contents of the source destination. The tags include `IMG`, `SCRIPT`, `IFRAME`,



CAUTION!!

CAUTION! Your computer is contacting www.milw0rm.com

Did you just visit that website?

Fig. 3. A screenshot of our user interface.

FRAME, and INPUT, as well as tags AUDIO, EMBED, VIDEO in HTML5. The content found within style tags are parsed through for any URL includes. This procedure predicts @import requests and image includes for backgrounds or behavior scripts. Last but not the least, attribute HREF from link tags is parsed, as it usually contains the include for CSS files. In essence, we recursively identify objects referred in retrieved WebPages to estimate the (separate) HTTP connections required to fetch all of them. We note that our solution does not require crawling hyperlinks and thus is quite efficient. Advanced Web contents such as Applets or AJAX (asynchronous JavaScript and XML) requests through XMLHttpRequest typically concern objects residing on the same domain as their parent page, and thus are safely disregarded by us.

2.3 Experimental Evaluation

In this section, we describe experimental evaluation on our solution. All the experiments were executed on a HP Pavilion dv9500t laptop computer that has 4GB memory, an Intel Core 2 Duo 2.2GHz CPU with ArchLinux X86_64.

In Table 1, we evaluate our program on several websites to assess its ability to predict legitimate outbound HTTP connections. For most websites studied, our program is able to predict most of the actual requests. For websites with the heavy use of JavaScript code such as digg.com, the prediction percentage is relatively low. Improving our prediction on JavaScript-generated requests requires interpreting JavaScript code along with the DOM object *out of the browser*. This task is subject to our future study.

We perform extensive experiments to evaluate the efficiency of the *predictor* mechanism in our implementation. The prediction is performed in parallel with the actual Web requests by the browser, and thus its execution has little impact on the browser's responsiveness to the user. Nevertheless, fast prediction is desirable because of the early detection of suspicious outbound HTTP requests. We evaluate four websites with distinct characteristics and our results shown in Table 2 blogs.zdnet.com is a very dynamic and rich website, which usually has new connections on every

Table 1. Evaluation on the prediction ability of outbound HTTP requests. Req. stands for requests. Dom. stands for domains.

URL	Actual Req.	Predicted Req.	% Predicted	Actual Dom.	Predicted Dom.
yahoo.com	37	67	92%	7	4
eset.com	51	67	94%	5	2
google.com	7	3	43%	3	2
cs.rutgers.edu	34	39	100%	1	1
digg.com	111	47	42%	21	7
codeigniter.com	29	177	100%	1	6

Table 2. Evaluation of prediction efficiency on four websites with full (F.) or lite (L.) predictors. Results are averaged from three runs. The time is shown in seconds.

	blogs.zdnet.com	www.cs.rutgers.edu	yahoo.com	google.com
Actual Req.	228	67	40	3
Network time (F.)	23.50	3.54	1.86	0.22
Parsing time (F.)	2.19	0.28	0.09	0.03
Total (F.)	30.99	4.035	2.87	0.28
Network time (L.)	3.75	0.22	0.56	0.19
Parsing time (L.)	0.83	0.27	0.09	0.03
Total (L.)	12.55	0.68	1.39	0.24

page load. `yahoo.com` provides less dynamic content than `blogs.zdnet.com` – this Web page stays consistent over a short period of time (e.g., a couple of days). `www.cs.rutgers.edu` is a static and medium-sized website. `google.com` is a very light and mostly static website. It only uses JavaScript code for the pull-down menu at the top.

We test two versions of our predictor implementation: a *full* predictor and a *lite* predictor. The full predictor is as described in Section 2.2, where each requested object is analyzed in the same fashion. In the lite predictor, images and JavaScript objects are not requested and processed, i.e., Steps 6 through 8c are skipped if the request is to fetech an image or JavaScript object. The lite version is effective for most websites and may significantly improve our prediction efficiency. For `yahoo.com` and `www.cs.rutgers.edu`, the lite predictor is faster than the full predictor, in particular for the more dynamic pages. For simple static page like `google.com`, the two versions do not differ much as expected. For `blogs.zdnet.com`, prediction time goes down with the lite predictor. However, some unique domains are not discovered in the process: certain objects are not found (404) or being redirected (300); these cases are not pursued by the lite predictor. Our experimental results indicate that both the full and lite versions of predictors perform reasonably well for typical websites on a personal computer.

3 Analysis on University Wireless Network Traces

Our solution provides a real-time suspicious out-bound traffic discovery mechanism. In detection phrase, we need user's participation to classify suspicious traffic into malicious or good traffic. In order to assess user's workload of our host-based traffic-analysis tool, we carry out a characterization study on 500 university-users' wireless network traces for 4-month period. We study both statistical and temporal patterns of *individuals*' Web usage behaviors from collected wireless network traces. Unlike pattern recognition based detection mechanism, we use these patterns to gain an insight in user's network activity, which implies the user's workload in using our system.

We run several filters on the original data. First, we remove the users whose total traffic volume is lower than 1 MB, which effectively remove the users with failed login and temporary users. Second, we only keep the outgoing and incoming TCP traffic with destination or source port 80, in order to filter out non-HTTP connections and exclude data from peer-to-peer software. Many HTTP-based P2P applications run on high port numbers. We find that there are many invalid MAC addresses in our data. We wrote a program to automatically verify the validity of a MAC address by comparing it with the published prefixes of authorized network interface card manufactures. Unmatched MAC addresses are notified and their corresponding traffic is removed.

In what follows, we use the words *host* and *user* interchangeably, as the hosts that connect to the wireless network are virtually all personal laptops.

Volume of Distinct IP Addresses Our overall analysis methodology is to explore and characterize network activities belonging to individual hosts. We compute and categorize each host's daily web traffic volume. We choose the top 500 users represented by distinct local MAC addresses that have the highest number active days. In an inactive day, the host has zero HTTP traffic with port 80. In Figure 4 (left), hosts are categorized by their daily numbers of IP addresses visited. Y-axis denotes the number of users who are in a category represented by the values in the square bracket. The majorities of users out of the 500 studied only visited a small number of servers and have low diversity in their daily Web traffic. On the other hand, a few users are extremely active and visit a large number of distinct IP addresses every day. On average, 278 hosts contacted less than 50 distinct IP addresses daily. Note that duplicate HTTP connections are counted only once, thus automatic refresh or reload operations by Web servers do not artificially increase the count of IP addresses.

Temporal Analysis Of Individual Web Usage We analyze how many *new* IP addresses and *old* IP addresses that a host visits each day. If an IP address visited by a host exists in the previous surfing history, then it is labeled as an *old IP*, otherwise, a *new IP* of that day. The surfing history of a user is initialized to be empty on day one, i.e., $H_1 = \emptyset$. Thus, all traffic on that day is new. At the beginning of an active day i, the surfing history is concatenated with day $i - 1$'s new IP set new_{i-1}, i.e., $H_i = H_{i-1} \cup new_{i-1}$. Thus, an IP address at day i is new only with respect to a user's surfing history up to day i. We observe that most hosts visit fewer numbers of new IP addresses than old IP addresses each day, indicating that most nodes are consistent with their surfing history. In Figure 4 (right), X-axis is the index of each user, Y-axis is daily number of IP addresses that visited by a user, the solid blue line is daily number of old IP addresses, and the dotted red one is the daily number of new IP addresses. We select

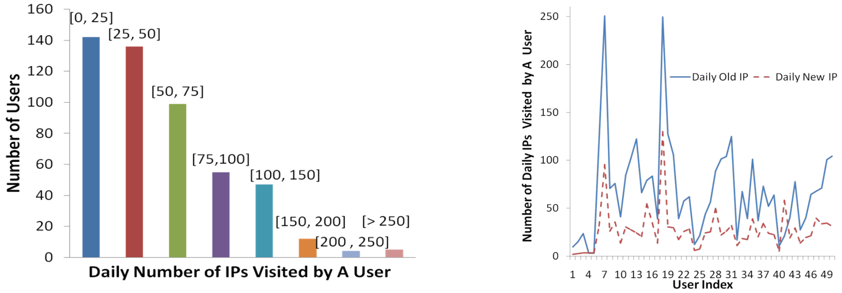


Fig. 4. Left: The number of hosts (in Y-axis) based on their volume of daily distinct IP addresses visited. Values in square brackets are the ranges of daily number of IP addresses visited. Right: The numbers of daily old and new IP addresses visited by 50 hosts, respectively.

the most active 50 hosts to show in this analysis. For most users in Figure 4, the number of new IP addresses grows with the number of old IP addresses. An overwhelming majority of hosts visited many more old IP addresses than new IP addresses on active days.

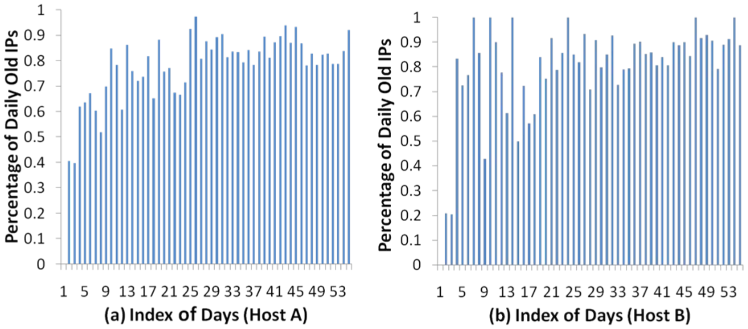


Fig. 5. Percentage of daily old IP addresses visited by two hosts, respectively. Both hosts have 55 active days.

We show the analysis results on two specific hosts in Figure 5, where Y-axis denotes the percentage of daily old IP addresses visited by a host and X-axis denotes the index of active days. For each day, the percentage is computed as the number of old IP addresses divided by the number of all IP addresses visited by the host. Inactive days are not included in the analysis. Both hosts visited significantly more old IP addresses than new ones each day. Both users demonstrate repetitive patterns in their surfing history. As the volume of new IP addresses is significantly lower than that of old IP addresses, the

Table 3. Numbers of days during which two hosts visit a certain percentage of old IP addresses, respectively. Total numbers of active days are 122 and 118, respectively.

Percentage of Old IP addresses Visited	Host 1	Host 2
100%	59 days	57 days
$\geq$ 90%	80 days	79 days
$\geq$ 80%	100 days	95 days

workload for analysis and monitoring would be low. For our host-based bot detection approach described in Section 2, we aim to focus on analyzing new IP addresses or URLs visited by a host.

We also identify the two active hosts in our dataset and find that both hosts have a high degree of repetitiveness in the IP addresses visited. In Table 3, we count the numbers of days during which two hosts visit a certain percentage of old IP addresses, respectively. For each active day of a host, the percentage is computed as the number of old IP addresses visited divided by the total IP addresses visited during that day.

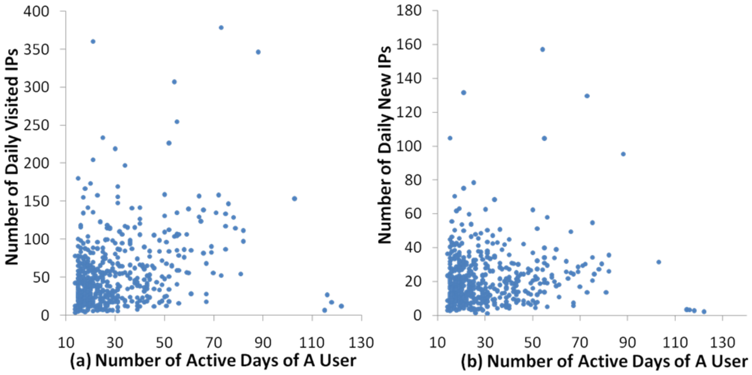


Fig. 6. X-axis is the number of active days of a host. Y-axis is the average number of the total IP addresses in (a) and new IP addresses in (b), respectively. 500 hosts are plotted. Each dot represents a user.

Profiling Visit Patterns Of Individual Hosts The degree of activity of a host can be represented by the number of total and new IP addresses visited daily or the number of active days during the 4-month long period. Intuitively, a user who surfs on Internet regularly tends to be more proficient in using the web and visit a diverse and large number of IP addresses. In an effort to investigating the correlation between the two metrics, we plot the number of daily visited IP addresses against the number of active days of a user in Figure 6. We study the 500 most active hosts whose numbers of active days range from 10 to 130 days. In Figure 6, X-axis in each graph is the number of active days; Y-axis is the daily number of the total IP addresses in (a) and new IP addresses in (b), respectively. We find that for some hosts the number of IP addresses visited grows with the number of active days, which is consistent with our intuition. The area close to the origin is dense with points in Figure 6 indicating that the majority of users studied have limited HTTP-based network activities both in terms of active days and the volume of distinct remote IP addresses visited. We further draw a 3-D plot with 500 hosts according to their (1) number of active days (2) daily IP addresses visited and (3) daily new IP addresses visited. The details can be found in technical report [32].

Summary on wireless network traces The number of distinct IP addresses visited by a host is most likely to be higher than the actual number of websites visited. A website refers to the top-level domain name, e.g., `google.com`, `yahoo.com`, `amazon.com`. There are several reasons. (1) Many popular servers have multiple IP addresses for load-balancing and fault-tolerance purposes. For example, 066.102.001.166 and 074.125.067.118 are among the several `google.com` servers. They are counted as two different IP addresses in our analysis. (2) Many websites heavily use third-party content providers for multimedia contents or advertisements. For example, when `www.cnn.com` is loaded, several content providers are contacted. (3) Dynamic IP addresses due to DHCP cause a single (remote) host to have distinct IP addresses at different time. More fine-grained data collection and analysis on the URLs and websites visited by the users are subject to our future study. We note that our analysis is based on distinct IP addresses visited as opposed to URLs, due to the limitation of our dataset. As a Web server typically hosts many Web pages, the actual number of URLs visited by the user may differ from the number of IP addresses.

4 Related Work

Our proposed personalized security approach is different from existing anomaly detection techniques [3, 22]. Personalized security aims at exploring individuals' and personal usage patterns for the detection purpose, whereas conventional anomaly detection methods construct generic solutions for users. In addition, we focus on detecting *internal* malware abuse and threats, as opposed to preventing break-ins coming from outside in the conventional settings. Security-oriented traffic analysis has caught much attention from both the network and security communities, including malware or botnet characterization [8, 21, 23, 31] and privacy-preserving routing and packet trace anonymization [17–19]. Our work differs from them in that i) we analyze the statistical and temporal patterns of *individuals'* application-layer Web usage (as opposed to low-level network packets); and ii) our analysis is user-centric by leveraging user's personal knowledge about her own surfing activities. To that end, our solution aims to address the usability, in particular nonintrusiveness, of the host-based malware detection solutions. Our tool is complementary to the existing network-level or program-level malware identification solutions.

Analyzing and characterizing organizational wireless network traces have traditionally been studied for maintaining the stability and availability of network resources [4]. Several studies have been performed on university campus wireless network traces [1, 11, 15, 27]. Researchers in Stanford University [27] studied a 12-week trace of their local-area wireless network in Computer Science Department building with attempt to find out the peak throughput rates and the cause of the peaks. Kotz and Essien [15] carried out a similar study with a significantly larger and broader population. They found out that network backup and file-sharing traffic contributed an unexpectedly large amount to the overall traffic, which was also found in [11].

Authors in the paper [1, 2] paid more attention on user behavior in the wireless network, but these behavior studies served as parameters for network performance optimization. For example, authors in [1] pointed out the load of each access point was determined mostly by individual user behaviors. In [2], it was found that the data-transfer rates of users follow a power law distribution. Power-law distributions were also found in certain characteristics of WWW, such as the distributions of document sizes and user requests for documents [5]. In comparison, our study on user behaviors in local wireless network differs from aforementioned studies. As opposed to optimizing network or server performance, we focus on analyzing users' individual and temporal behavior patterns in their wireless Web traffic, which is motivated by the need for personalized security. Policies in firewalls are typically based on port numbers and IP addresses. In contrast, we provide much more fine-grained inspection as we examine each HTTP connection intercepted on the network interface.

5 Conclusions and Future Work

In this paper, we proposed a novel host-based security tool that identifies suspicious outbound network requests with user's participation. Specifically, we described a personalized security approach and a simple-yet-effective host-based network security solution that identifies abnormal outbound HTTP requests based on *out-of-band* (i.e., browser-independent) prediction and user-assisted classification. We also described our results on analyzing a large-scale wireless network dataset that involves more than 500 users over 4-month period. We analyzed the *individual* usage patterns of users in an organization in order to assess the workload of our host-based malware detection solution. Our characterization analysis on individuals' surfing patterns is useful beyond the specific malware-detection problem studied, as it provides insights to how individual surfing-behavior patterns may be leveraged for improved web services.

For future work, we plan to carry out user studies to evaluate humans' traffic recognition abilities. The hypothesis that we aim to evaluate in the user study is that a user knows the websites she is currently visiting and thus can recognize malware-related traffic to unfamiliar URLs. In a user study, each participant will be asked to freely surf online for 10 to 20 minutes, during which we will randomly access a list of arbitrary (bot) servers, i.e., inject malware traffic to test whether a user can recognize it. For unpredicted outbound HTTP requests including the injected ones, we will prompt a window (as in Figure 3) asking whether or not the user just visited the URL. From users' responses, we will compute false positive and false negative rates of their performance. Here, a false positive result will indicate that the user misclassified legitimate user-initiated traffic as malware HTTP requests. A false negative result, conversely, will indicate that the participant has misclassified bot URLs for their own traffic. With our comprehensive traffic prediction mechanism described in this paper, we expect this security tool to be nonintrusive to users.

In addition, we will investigate techniques to include personalized semantic analysis of surfing records and novel clustering methods for identifying outliers and suspicious traffic. This study will first extract *surfing tastes* of individuals and then detect suspicious Web requests whose content is *inconsistent* with the user's previous surfing history. We also plan to construct more advanced pattern-recognition techniques on individuals' usages.

References

1. A. Balachandran, G. M. Voelker, P. Bahl, and P. V. Rangan. Characterizing User Behavior and Network Performance in A Public Wireless LAN. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 195–205, New York, NY, USA, 2002. ACM.
2. M. Balazinska and P. Castro. Characterizing Mobility and Network Usage in A Corporate Wireless Local-Area Network. In *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 303–316, New York, NY, USA, 2003. ACM.
3. K. Borders and A. Prakash. Web Tap: Detecting Covert Web Traffic. In V. Atluri, B. Pfitzmann, and P. D. McDaniel, editors, *ACM Conference on Computer and Communications Security*, pages 110–120. ACM, 2004.
4. A Community Resource for Archiving Wireless Data At Dartmouth (CRAWDAD) `http://crawdad.cs.dartmouth.edu/`.
5. C. Cunha, A. Bestavros, and M. Crovella. Characteristics of WWW Client-based Traces. Technical report, Boston, MA, USA, 1995.
6. D. Dagon, G. Gu, C. P. Lee, and W. Lee. A Taxonomy of Botnet Structures. In *ACSAC*, pages 325–339, 2007.
7. Emerging Cyber Threats Report for 2009, October 2008. Georgia Tech Information Security Center.
8. S. Gianvecchio, M. Xie, Z. Wu, and H. Wang. Measurement and Classification of Humans and Bots in Internet Chat. In *Proceedings of USENIX Security Symposium*, 2008.
9. G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proceedings of the 17th USENIX Security Symposium*, 2008.
10. G. Gu, J. Zhang, and W. Lee. Botsniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS)*, 2008.
11. T. Henderson, D. Kotz, and I. Abyzov. The Changing Usage of A Mature Campus-wide Wireless Network. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 187–201, New York, NY, USA, 2004. ACM.
12. B. A. Huberman, P. L. Pirolli, J. E. Pitkow, and R. M. Lukose. Strong Regularities in World Wide Web Surfing. *Science*, 280(95), 1998.
13. N. Ianelli and A. Hackworth. Botnets as A Vehicle for Online Crime, 2005. `http://www.cert.org/archive/pdf/Botnets.pdf`.
14. A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-Scale Botnet Detection and Characterization. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 7–7, Berkeley, CA, USA, 2007. USENIX Association.
15. D. Kotz and K. Essien. Analysis of A Campus-wide Wireless Network. In *Proceedings of ACM Mobicom*, pages 107–118. ACM Press, 2002.

16. M. T. Louw, J. S. Lim, and V. N. Venkatakrishnan. Enhancing Web Browser Security Against Malware Extensions. *Journal in Computer Virology*, 4(3):179–195, 2008.
17. J. C. Mogul and M. Arlitt. SC2D: an Alternative to Trace Anonymization. In *MineNet '06: Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 323–328, New York, NY, USA, 2006. ACM.
18. R. Pang, M. Allman, V. Paxson, and J. Lee. The Devil and Packet Trace Anonymization. *SIGCOMM Comput. Commun. Rev.*, 36(1):29–38, 2006.
19. R. Pang and V. Paxson. A High-level Programming Environment for Packet Trace Anonymization and Transformation. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies,architectures, and protocols for computer communications*, pages 339–351, New York, NY, USA, 2003. ACM.
20. R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *USENIX Security Symposium*, pages 223–238. USENIX, 2004.
21. S. Saroiu, S. D. Gribble, and H. M. Levy. Measurement and Analysis of Spyware in a University Environment. pages 141–153.
22. S.-P. Shieh and V. D. Gligor. On a Pattern-Oriented Model for Intrusion Detection. *IEEE Transactions on Knowledge and Data Engineering*, 9(4), July/August 1997.
23. S. Singh, C. Estan, G. Varghese, and S. Savage. Automated Worm Fingerprinting. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design Implementation*, Berkeley, CA, USA, 2004. USENIX Association.
24. SNORT, an open source network intrusion prevention and detection system. `http://www.snort.org/`.
25. J. Stewart. Top Spam Botnets Exposed, April 2008. `http://www.secureworks.com/research/threats/topbotnets`.
26. Symantec. `http://www.symantec.com/index.jsp`.
27. D. Tang and M. Baker. Analysis of A Local-Area Wireless Network. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 1–10, New York, NY, USA, 2000. ACM.
28. October 2003. Trusted Computing Group. Trusted platform module main specification, Part 1: Design principles, Part 2: TPM structures, Part 3: Commands. Version 1.2, Revision 62.
29. TCG PC Client Specific TPM Interface Specification (TIS), Version 1.2. Trusted Computing Group. `http://www.trustedcomputinggroup.org/groups/pc_client/`.
30. S. Webb, J. Caverlee, and C. Pu. Predicting Web Spam with HTTP Session Information. In *Proceedings of the Seventeenth Conference on Information and Knowledge Management (CIKM 2008)*, October 2008.
31. Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming Botnets: Signatures and Characteristics. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 171–182, New York, NY, USA, 2008. ACM.
32. H. Xiong, P. Malhotra, D. Stefan, C. Wu, and D. Yao. User-Assisted Host-Based Detection of Outbound Malware Traffic. Technical report, Rutgers University, October 2009.