A

# Dependency Analysis for Traffic Anomaly Detection

HAO ZHANG, Virginia Tech
DANFENG (DAPHNE) YAO, Virginia Tech
NAREN RAMAKRISHNAN, Virginia Tech
ZHIBIN ZHANG, Chinese Academy of Sciences

This paper describes an approach to enforce dependencies between network traffic and user activities for anomaly detection. We present a framework and algorithms that analyze user actions and network events on a host according to their dependencies. Discovering these relations is useful in identifying anomalous events on a host that are caused by software flaws or malicious code. To demonstrate the feasibility of user intention-based traffic dependence analysis, we implement a prototype called *CR-Miner* and perform extensive experimental evaluation of the accuracy, security, and efficiency of our algorithm. The results show that our algorithm can identify user intention-based traffic dependence with high accuracy (average 99.6% for 20 users) and low false alarm rates. Our prototype can successfully detect several pieces of HTTP-based real-world spyware. We show that user intention-based traffic dependency is a safety property along with the corresponding security automaton as defined in Schneider's EM-enforceable security framework. This formalization demonstrates that dependency enforcement has useful characteristics for practical and scalable deployment.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General; D.4.6 [**Operating Systems**]: Security and Protection

General Terms: Design, Algorithms, Security

Additional Key Words and Phrases: anomaly detection, traffic dependency, user intention, enforceable policy

## 1. INTRODUCTION

Conventional intrusion detection approaches focus on detecting specific intrusive and malicious patterns. They work well if the attack signatures or behaviors are known or can be modeled *a priori*. Anomaly detection – a field pioneered by Denning [Denning 1987] – defines, specifies, and enforces normal traffic and interaction patterns in a network or on a host. Anomalies or outliers refer to any activities that do not conform to regular behaviors. Statistical techniques modeled under specific domain knowledge have been proposed for anomaly detection [Denning 1987; Heberlein et al. 1990; Shieh and Gligor 1997; Snapp et al. 1991]. For example, dynamic Bayesian networks can be used to detect abnormal data access patterns by malicious insiders to a sensitive database [An et al. 2006]. However, realizing general anomaly detection is challenging, especially for complex and diverse behaviors involving activities spanning users, hosts,

This work has been supported in part by NSF grant CAREER CNS-0953638 and ARO grant STIR-450080. A preliminary version of this work appeared in *Proceedings of Workshop on Semantics and Security (WSCS)*, in conjunction with the IEEE Symposium on Security and Privacy. San Francisco, CA. May 2012.
Authors' addresses: H. Zhang, D. Yao, and N. Ramakrishnan, Department of Computer Science, Virginia Tech, Blacksburg, VA 24060, {haozhang, danfeng, naren}@cs.vt.edu. Z. Zhang, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, 100190, zhangzhibin@ict.ac.cn. Yao is the corresponding author.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

and networks. When the definitions for normal behaviors are restrictive, false positives (i.e., false alarms) may be high; whereas broad definitions for normal behaviors may result in high false negatives.

We describe a novel user intention-based anomaly detection approach that can be used for detecting anomalous traffic on a host. Our solution aims at capturing *dependencies* between a user's input activities (e.g., clicking on a hyperlink of a webpage) and system/network events (e.g., HTTP GET requests). We explore direct and indirect dependencies in how a user interacts with applications and how applications respond to the user's requests following the specifications of the applications. By enforcing an application's correct responses to user activities, we are able to identify *vagabond events*. Vagabond events refer to outbound network events that are not generated by any user actions and may hence be due to anomalies. We do not require any knowledge or assumption on the regularities of user behavior patterns.

Our work aims to demonstrate the feasibility of user intention-based dependence analysis for detecting suspicious network connections of a host in a concrete web browser setting. The traffic dependence analysis is a powerful technique for identifying malware activities. We enforce correct system behaviors, as opposed to anomalous characteristics. Our dependence-based anomaly detection has advantages over conventional pattern-based solutions (such as [Chandola et al. 2009; Christodorescu et al. 2008; Denning 1987; Teng et al. 1990]), because it does not require *a priori* knowledge or assumptions about the normal data patterns. Our contributions are summarized as follows.

(1) We demonstrate the use of dependence analysis for detecting anomalous web traffic in our *CR-Miner* (Causal Relation Miner) framework. Specifically, we describe how to construct a concrete dependence analysis model for the web browser and use it for predicting and enforcing allowable web traffic by specific user actions. We address the underlying technical challenges by instrumenting the browser and operating system for monitoring, inferring dependency patterns, and designing efficient algorithms for real-time analyzing event hierarchies.
We describe a tree representation of dependencies existed in outbound traffic in a *traffic-dependency graph* (TDG). We design an efficient breadth-first search based algorithm for inferring dependencies of outbound requests, i.e., finding the event in the TDG that causes the newly-observed outbound request. The inference inspects the temporal, semantic, and process-related attributes of events and their dependencies.
(2) We implement a prototype of CR-Miner in Windows and extensively evaluate its performance in terms of its accuracy and feasibility in anomaly detection. We performed a user study with 20 participants and analyzed CR-Miner's false positive rates. We also evaluate the accuracy of our dependency inference algorithm in noisy traffic by combining the traffic of multiple users. Experimental results show that our algorithm substantially outperforms the temporal-only dependence analysis, which is mentioned in BINDER [Cui et al. 2005], in terms of the accuracy of dependence prediction. We further demonstrate the use of CR-Miner to detect several pieces of real-world and proof-of-concept spyware.
To prevent malware from spoofing legitimate traffic in order to circumvent our anomaly detection, we further provide a lightweight cryptographic mechanism in the Firefox browser to ensure the integrity of HTTP packet headers. Because fields of the header are used by our dependence analysis, our message authentication code improves the integrity of CR-Miner against stealthy malware's tampering.
(3) In order to demonstrate the generality of traffic dependency analysis, we provide theoretical modeling and analysis for CR-Miner. We give an abstract finite state

automaton model to represent the states, user-triggered transitions, and traffic responses in networked applications such as the browser. The FSAs allow us to further define the security automata for enforcing traffic dependency property and formally show that traffic dependency is an EM-enforceable security policy as defined in Execution Monitoring (EM) framework [Schneider 2000]. Specifically, traffic dependency has three properties: expressible as a predicate over executions, prefix closed, and decision made within a finite period. This work is useful for systematically understanding and formalizing user intention-based dependency analysis.

Our user intention-based traffic dependence analysis produces structures in network events. These structures across outbound requests enable improved context-aware security analysis. Dependence analysis on network flows builds a *traffic dependency graph* based on the observed network events and user actions. This approach of inferring and enforcing the logical dependencies among events is a general anomaly detection technique, which can also be applied to detect anomalies in file-system events.

Besides the traffic dependencies studied in this paper, the approach of user intention-based dependence analysis may be for anomaly detection in file-system events. For example, it is desirable to enforce the dependencies between user input events and file-system events on a host to detect unauthorized file-system activities, such as download, read, or write. DeWare [Xu et al. 2011], BLADE [Lu et al. 2010], and UIBAC [Shirley and Evans 2008] leverage user behaviors for certain file-access regulation. Yet, more systematic study is needed for general user-centric dependence analysis in file systems.

Our proposed traffic dependence solution cannot be realized by the conventional (stateful) firewall, because our inference of dependencies requires complex algorithmic computation on system events beyond simple rule-based filtering.

The rest of the paper is organized as follows. We give our traffic-dependency graph model and definitions in the next section. The building blocks and algorithms are presented in Section 3. We formalize a finite state machine model for the browser traffic dependency in Section 4 and show its connection with the enforceable mechanism framework (EM) of Schneider [Schneider 2000]. We analyze the security issues in Section 5.2. The prototype implementation of the CR-miner framework and experimental evaluations are presented in Section 6. The related works are discussed in Section 7. Conclusions and future work are described in Section 8.

## 2. TRAFFIC-DEPENDENCY GRAPH

Discovering user intention-based traffic dependencies is challenging, because modern applications such as web browsers often automatically fetch content and generate requests without explicit user actions. The dependencies of those legitimate requests should be properly identified without triggering false alarms. In the HTTP protocol, each object is retrieved in a separate outgoing 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. One needs to discover not only the dependencies among user actions and network events, but also the layered dependencies of those network events. This paper focuses on outbound HTTP packets by the web browser, which can be generalized to other types of applications and network-flow types.

Next, we give definitions used in our model and an example illustrating the traffic dependence among the events. Then, we describe the application of dependency finding in computer security.

### 2.1. Dependencies in Browser Traffic

We introduce the terminology used in the CR-Miner framework, including traffic-dependency graph, user and traffic events, subroot traffic, and the parent-child and sibling relations on the traffic-dependency graph.

*Definition* 2.1. A traffic-dependency graph (TDG) is a forest of trees of arbitrary depths with directed edges representing the dependencies among network events and user actions. The root of each tree is a user event, and the internal and leaf nodes of the trees are traffic events. A directed edge from event $a$ to $b$ represents that event $b$ is caused by $a$. The trees in the forest are chronologically ordered, so are the children of a node.

A TDG satisfying the above definition is *well formed*. The tree-based TDG enables us to apply breadth-first traversal when inferring dependencies, which is described in Section 3.1.

*User events* refer to the user's inputs to the application through input devices such as the keyboard or mouse, which have attributes such as timestamp and ID of the process notified by the event, event name, and content (e.g., the cursor's coordinate and the keystroke). A user event in TDG is legitimate if and only if it is not forged by any malicious software. We give several practical techniques of ensuring the authenticity and provenance of user events in Section 5.2. In the context of browser, we consider two main types of traffic-inducing user events: mouse clicks on hyperlinks and keyboard inputs to the textbox or address bar.

A *traffic event* refers to an outgoing HTTP request from the host, which includes attributes such as the timestamp, process ID, source and destination IP addresses, source and destination port numbers, and referrer header field in HTML. Traffic events are further categorized into different levels according to their relative dependencies. We use the phrases traffic event and network request interchangeably.

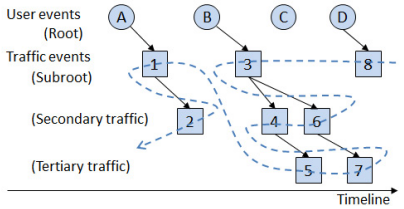
A *subroot* is a special type of traffic event. It refers to the traffic event that directly corresponds to the user's request, e.g., fetching `index.html` from web server `www.example.com` in response to a user's mouse click on link `www.example.com/index.html`. Each user event has *at most one* subroot on the traffic-dependency graph. For example in Figure 1, the traffic events 1, 3, and 8 are subroots, which are caused by the user events `A`, `B`, `D`, respectively.

The subroot traffic may cause the browser to fetch more objects by generating additional outgoing requests from the host, e.g., fetching the images or JavaScript referred to by a HTML page. We define that those requests are the *children* of the subroot events or *secondary traffic*, e.g., events 2, 4, and 6. Secondary traffic may cause the browser to issue further requests, e.g., as a result of running JavaScript code. Thus, *tertiary traffic* (e.g., events 5 and 7 in Figure 1) and lower-level traffic can be similarly categorized. The resulting hierarchies form a forest of trees of arbitrary depths with the user events being the *roots*.

*Parent-child relation* on TDG is between two traffic events that are at two adjacent levels and one of them directly triggers the other. For example, the pairs (1, 2), (3, 4), (3, 6), (4, 5), (6, 7) in Figure 1 have parent-child relations. *Sibling relation* describes the two traffic events that are at the same level and are generated by the same parent traffic. Events with the sibling relations share the same parent. Pair (4, 6) in Figure 1 has the sibling relation. However, events 5 and 7 are not siblings as they belong to different parents.

Our definition of security in the CR-Miner is given below.

*Definition* 2.2. (Definition of security) In the user intention-based traffic dependency model, a legitimate traffic event belongs to a tree in the traffic-dependency graph




| | Event Name | Value | URL |
|---|---|---|---|
| A | KeyDown | Return | http://www.engadget.com/ |
| B | MouseClick - Left | X=1069 Y=474 | http://www.cnet.com/ |
| C | Wheel | -120 | n/a |
| D | MouseClick - Left | X=240 Y=363 | http://www.nytimes.com/ |

| | http Request | Host | Referrer |
|---|---|---|---|
| 1 | / | www.engadget.com | http://www.engadget.com/ |
| 2 | /media/main.css | www.engadget.com | http://www.engadget.com/... |
| 3 | / | www.cnet.com | null |
| 4 | .../oreo.combined.js | i.i.com.com | www.cnet.com |
| 5 | .../site1headerBg.png | i.i.com.com | http://i.i.com.com/cnwk.1d/css/... |
| 6 | /en_US/all.js | connect.facebook.net | http://www.cnet.com/ |
| 7 | .../bnVbfysT6kL.css | b.static.ak.fbcdn.net | http://www.facebook.com/plugins/... |
| 8 | /opinion/index.html | www.nytimes.com | http://www.nytimes.com/ |

Fig. 1. An example of a traffic-dependency graph of events with their attributes. Solid arrows indicate dependency relations, and the dotted arrow indicates the breadth-first traversal in the dependency inference.

as defined in Definition 2.1 that is rooted at a legitimate user event. That is, the traffic event $p$ is either a subroot, i.e., the child node of a root user event, or $p$'s ancestor node (e.g., parent, grand-parent) is a subroot. Otherwise, the outbound request is a vagabond event and considered suspicious.

A TDG $T$ has the following properties.

—*Acyclic*: $T$ is a forest of trees in arbitrary height. For each individual tree in $T$, we name it *subtree*, which is rooted by *subroot*. The directed edge in each subtree connected between two nodes is unidirectional, which comes from the parent to its triggered node.
—*Expandable*: The degree of node in a subtree can be expanded as it grows. One legitimate network request $p^*$ is appended to $T$, then new TDG still meets the security definition.
Since $p^*$ is a legitimate packet, it can be categorized in one of the following cases: *(i)* $p^*$ is a *subroot* node, then $p^*$ is appended as a new subroot node in $T$; *(ii)* $p_i$ and $p^*$ have *parent-child* relation, where $p_i$ is an existing node in $T$, then $p^*$ is a dependent node of $p_i$ in $T$. So, we show that $p^*$ is either a subroot or a dependent node of an existing packet. Therefore, the security definition holds in new TDG.
—*Attribute-based*: Each node with multiple attributes, including network and kernel information of each request. Attributes describe properties of user and network events.
—*Partial well-formed*: We define *crown* of a tree in $T$ as any connected subtree that contains the root and the subroot of the tree. Partial well-formedness property states that if $T$ is well-formed, then any crown of $T$ is also well-formed.

### 2.2. Applications and Threat Model

The traffic dependence analysis can be used to detect anomalous activities on a host, which may include the detection of two specific types of threats: *i)* identifying the network activities of stealthy malware (e.g., spyware on a user's computer), and *ii)* identifying inadvertent software flaws or intentional software errors (e.g., software behaviors

that deviate from specifications). Our study in this paper is focused on the first type of anomalies.

— Stealthy malware that behaves as a user-level application on the host, certain instances of spyware and malicious bots perform data exfiltration, spamming, botnet command-and-control, or launch denial-of-service attacks. Specifically, we consider two cases of malware in this paper as follows.
*Case I*: malware is an extension or add-on component of an existing legitimate application, e.g., spyware as a malicious Firefox browser extension or parasitic malware [Srivastava and Giffin 2010]. Malware runs along with the host program and has the same process ID as the host program. A specific example of such a type of spyware is `FFsniff`, which secretly sends out victim's ID along with the password to the remote host.
*Case II*: malware is a stand-alone user-level application and runs with a unique process ID, such as the malware `Trojan.Brojack.A`, which we test in Section 6.6.
— Software, which comes from unknown or untrusted developers, may perform undesirable and unauthorized network activities that are not causally related to the user's inputs due to errors or flaws. Identifying stealthy unwanted traffic is important, as these packets may leak information of the user (e.g., [Jung et al. 2008]), consume bandwidths, and cause further security vulnerabilities. Legitimate automated traffic, such as system updates and RSS feeds, can be whitelisted (See also Section 6.2).

In this paper, we focus on analyzing the dependence in browser's HTTP traffic and experimentally demonstrate its effectiveness in detecting stealthy spyware activities. CR-Miner performs the dynamic analysis of dependencies in network traffic, which differs from the static dependence analysis, such as call graph construction in the programming language paradigm or the work on dependencies of services [Bursztein and Goubault-Larrecq 2007].

If malicious extensions can modify the existing DOM of a visited page loaded to a browser, a user may be tricked to click malicious links, which sends outbound requests for unintended objects. In order to detect this attack, one needs to utilize additional techniques besides what is described in this work, such as strong browser security policies, a parallel universe approach for predicting browser content [Xiong et al. 2009], or user discretion in installing extensions on the host.

Our analysis requires the knowledge of certain specification of an application, in particular how the application responds to a user's inputs and actions (e.g. by generating particular types of system and network events). Based on these specifications, we extract and enforce policies defining dependencies within the application. For example, in the user's browser case, our approach requires knowing how a browser responds to the user's requests.

Our solution described in the next section enforces the dependencies across user actions and network events without modifying either the application or the kernel. Yet, one may attempt to design an alternative approach through taint-tracking [Yin et al. 2007] based techniques through modifying and instrumenting the application in such a way that: *i)* the root of the traffic-dependency graph may be tainted; *ii)* the taint bits can be further propagated throughout the application according to dependency rules enforced; and *iii)* outbound traffic are required to have the valid taint proofs which should be hard to forge or replay. However, such an alternative would require substantial modification of the application (e.g., browser).

The work-flow in CR-Miner including data collection, dependency rule generation, and causal relation analysis is illustrated in Figure 2.

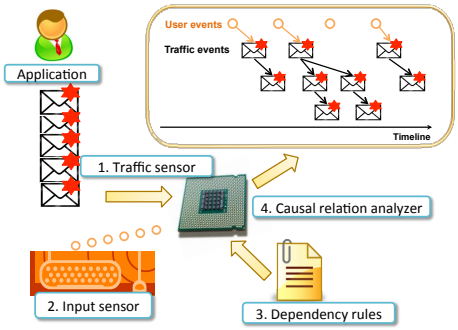



Fig. 2. An illustration of the system components and work-flow in CR-Miner.

## 3. CONSTRUCTION OF TDG

The goal of CR-Miner is to identify structured dependencies (or lack thereof) in network traffic. CR-Miner can be used as a tool for detecting anomalous events. We describe the dependency inference procedure and its building blocks for constructing the traffic-dependency graph (TDG).

A TDG may be constructed incrementally by inserting a new traffic event with unknown dependency to a well-formed TDG, which is suitable for real-time monitoring and is utilized by our CR-Miner. The construction of TDG relies on the attributes of events and dependency rules derived from the specific application.

### 3.1. Dependency Inference Procedure

This section describes our breadth-first search (BFS) based algorithm for the TDG construction. The algorithm utilizes the building blocks (namely *Is_Child*, *Is_Sibling*, and *Is_Subroot*), which are presented in the next section.

Given a new request, dependency inference (DI) algorithm aims to identify its dependence with respect to the known requests. We construct a forest structure to store the network requests and organize them according to the definition of TDG. The requests with known dependencies are chronologically organized into trees rooted by user events in the existing traffic dependency graph. The subroots, thus, are also chronologically ordered. Given a new network request, *Is_Subroot*() routine determines whether or not it is a subroot. If the request is of the type subroot, then there is no need to traverse the traffic-dependency graph. Otherwise, the algorithm traverses the forest of trees to identify the parent of this new request based on its attributes.

Our algorithm opts for a breadth-first traversal within a tree starting from the most recent subroot for the following two reasons. Once subroot events are identified, one needs to further identify the parent-child and sibling relations among observed network requests based on their attributes according to dependency rules. We observe from our experiments that *i)* the incoming new request is typically caused by recent requests, and *ii)* the traffic dependency trees that we manually construct are shallow and wide. Therefore, this BFS approach allows us to quickly identify the parent node of the new request. As an example, the sequence of traversal for the TDG in Figure 1 is 8, 3, 6, 4, 7, 5, 1, 2.

In order to find the parent of a new request, the traversal starts at the most recent subroot request (e.g., node 8) and runs *Is_Child* to test the parent-child relationship between this subroot and new request. If no dependence is found, then it compares the new request with the child nodes of this subroot starting from the most recent one, as well as their child nodes if needed. For each comparison, *Is_Child* and *Is_Sibling* tests are run. If the dependence is not found after all nodes on the tree are compared, then

the next subroot (e.g., node 3 in Figure 1) and its descendant nodes are compared. Intuitively, the process terminates if either a dependence is found or all existing requests have been compared. The worst-case complexity of such a basic dependency inference algorithm requires traversing the entire TDG, and is $O(n)$ where $n$ is the total number of traffic and user distinct nodes on the current TDG. At each node, comparisons of event attributes as shown in sub-procedures are fast.

We further optimize the algorithm by skipping unnecessary comparisons. We achieve the speedup by leveraging the underlying consistency among attributes (e.g., PID) of dependent nodes on the same tree in TDG. We use a simple data structure to avoid the unnecessary comparisons to obsolete nodes. Our dependency rule for parent-child relation requires the timestamps of the two events fall within a threshold $\tau$ (details in the next section). We consider the timestamp of the new request ($p^*$.*timestamp*) with respect to the most recent traffic event ($\hat{p}$.*timestamp*) on a tree. If $p^*$.*timestamp* - $\hat{p}$.*timestamp* $> \tau$, then it is not necessary to compare $p^*$ with other nodes with older timestamps in that tree or subtree. This speedup requires keeping track of the timestamps of the most recent nodes within subtrees. To realize this optimization, each internal node $p$ on a tree in TDG has an additional attribute that contains the timestamp of the most recent traffic event in the subtree rooted at $p$. This attribute needs to be updated when the tree expands. Similarly, the process ID of a subroot is the same as the PIDs of its descendants. Therefore, if a new request has a different PID than the subroot, then there is no need to compare other nodes on the same tree. We use an auxiliary queue to realize the breadth-first traversal. These optimizations improve the average-case complexity of the algorithm. The pseudocode of our dependency inference algorithm is shown in Algorithm 1. If the algorithm returns true on a new traffic event $p^*$, then $p^*$'s parent exists in the current TDG. Otherwise, $p^*$ may be vagabond and thus suspicious.

### 3.2. Details of Sub-Procedures

To instantiate the building blocks *Is_Child*(), *Is_Sibling*(), and *Is_Subroot*() used in Algorithm 1, we describe sets of rules and procedures to infer dependencies.

— *Is_Child*() determines whether or not there is a parent-child relation between two traffic events.
— *Is_Sibling*() determines whether or not there is a sibling relation between two traffic events.
— *Is_Subroot*() determines whether or not there is any dependence between a user event and its corresponding subroot traffic event.

Our rules are derived based on patterns of user interaction and attributes of HTTP traffic from the browser including their system properties in order to capture the various dependencies. The rules are summarized and categorized by analyzing browser behaviors together with our experimental observations. How to automatically extract traffic-dependency features with machine learning techniques is our ongoing work.

Inputs to *Is_Child*() are two traffic events $p_a$ and $p_b$, where $p_a$ is a node on TDG with known dependency and $p_b$'s dependency is unknown. Event $p_a$ may or may not be a subroot node. Traffic event $p_a$ is a parent of $p_b$, if and only if the following conditions are all satisfied.

(1) The interval between timestamps of $p_a$ and $p_b$ is within a threshold $\tau$ and event $p_a$ proceeds $p_b$.
(2) The two outbound network requests $p_a$ and $p_b$ share the same (non-null) process ID.
(3) The domain name in $p_b$'s referrer is identical to that of $p_a$. (Referrer is defined by the HTTP standard as the URL of the previous request that leads to this request.)

**Algorithm 1** Dependency Inference Procedure in CR-Miner.

**Require:** A newly-observed traffic event $p^*$;
a forest $F$ of chronologically ordered trees of events rooted by user events, which are parents of subroots $\{T_1, \ldots, T_m\}$ where subroot $T_m$ is the most recent one; and a threshold $\tau$.
**Ensure:** True, if the parent node of request $p^*$ is found;
False, otherwise.

1: **if** Is_Subroot($p^*$) **then**
2: $T_{m+1} \leftarrow p^*$
3: Append $T_{m+1}$ to forest $F$ and update $T_{m+1}.newestTimestamp \leftarrow p^*.timestamp$
4: **return** True
5: **else**
6: **for** $i \leftarrow m$ to 1 **do**
7: create Queue $Q$ and enqueue the subroot $T_i$ onto $Q$
8: **while** $Q$ is not empty **do**
9: node $n \leftarrow$ dequeue($Q$)
10: **if** $n.pid \neq p^*.pid$ or $p^*.timestamp - n.newestTimestamp > \tau$ **then**
11: go to line 8
12: **else if** Is_Child($n$, $p^*$) **then**
13: $p^*.parent \leftarrow n$ and update the $newestTimestamp$ for nodes on the path from $p^*$ to its subroot node
14: **return** True
15: **else if** Is_Sibling($n, p^*$) and !Is_Subroot($n$) **then**
16: $p^*.parent \leftarrow n.parent$ and update the $newestTimestamp$ for nodes on the path from $p^*$ to its subroot node
17: **return** True
18: **else**
19: ▷ Breadth-first traversal by an auxiliary queue
20: **for** all children of node $n$ **do**
21: enqueue the child nodes onto $Q$
22: **end for**
23: **end if**
24: **end while**
25: **end for**
26: **end if**
27: **return** False

*Is_Sibling*() procedure is used for the nodes whose parent nodes cannot be directly determined; identifying the sibling relation of a request helps establish parent-child relation by the transitivity. We are given two outbound HTTP requests $p_a$ and $p_b$, where $p_a$'s parent node is known, $p_b$'s parent is unknown, and $p_a$ proceeds $p_b$. To determine whether $p_b$ is a sibling node of $p_a$, we define dependency rules as follows.

(1) The interval between timestamps of $p_a$ and $p_b$ is within a threshold $\tau$ and event $p_a$ proceeds $p_b$.
(2) The two outbound network requests $p_a$ and $p_b$ share the same (non-null) process ID.
(3) The referrers of both requests are non-null and identical.

Finding sibling relations is useful in identifying new parent-child relations in our traffic dependence analysis. *Transitivity* helps constitute the parent-child relation after finding the sibling relation, and it defines as follows. If requests $p_a$ and $p_b$ are siblings, and request $p_c$ is the parent of $p_a$, then $p_c$ is also the parent of $p_b$. *Is_Sibling*()

complements the dependency test in *Is_Child*(). Some requests are sent far later their parents, as observed in our experiment. *Is_Sibling* helps identify late-arriving child nodes whose intervals of timestamps with respect to their parent are larger than the specified threshold, yet whose intervals with respect to the (older) sibling are still within the threshold.

The transitivity analysis above provides the comprehensive coverage in dependency analysis. In Section 6.1, we show that the transitivity rule mitigates the problem of missing attributes.

*Is_Subroot* procedure is to identify the traffic events of the type subroot through analyzing the user events and the outgoing network requests by their attributes including process IDs, timestamps, and the content of events.

In the context of the browser, traffic-inducing user events may include typing into the address bar of the browser, clicking on a hyperlink or a bookmark, opening a new window or tab, and reloading a webpage. Given a user event (of either a mouse click or keyboard inputs), the corresponding subroot traffic event in CR-Miner is the first immediate outgoing network request that has the identical process ID and with correlating content. The content may be the URL of the hyperlink for a mouse click, which needs to match the URL in the subroot request. Some keyboard inputs entered to text fields (such as Google search box) are not associated with a specific URL. Therefore, accurately identifying the subroot is technically complex in CR-Miner, and requires comparing the timestamps and PIDs of user events collected at both the kernel level and the application level. Subroot requests may contain referrers, e.g., mouse clicks on hyperlinks of webpages. Some subroots may not contain referrers, e.g., entering a URL directly in the address bar of a browser. Thus, the referrer attribute is not used to identify subroots in the CR-Miner. More details on user-event processing are given in Section 6.1.

Algorithm 1 shows how the three sub-procedures (namely *Is_Child*, *Is_Sibling*, and *Is_Subroot*) are used in constructing the traffic-dependency graph.

We formalize the traffic-dependency model based on the finite-state automaton and its constraints in the next section. Such a formal dependency model allows one to derive fine-grained requirements of legitimate event sequences, and is a specialized Schneider's execution monitor [Schneider 2000].

## 4. SECURITY AUTOMATA AND ENFORCEABILITY

In this section, we analyze properties of enforcing user intention-based traffic dependency in a system, specifically on whether or not traffic dependency property is enforceable at real time according to requirements defined by the widely accepted execution monitoring (EM) model [Schneider 2000]. We show that traffic dependency property is a EM-enforceable security policy, and present the specific security automaton as the recognizer or classifier for identifying abnormal event sequences.

Execution Monitoring (EM) [Schneider 2000] is a formal language based security model [Schneider et al. 2001]. It has been widely used in many security applications, including mobile code [Sekar et al. 2001, 2003], program verification [Sun et al. 2008; Langar et al. 2011], operating systems [Shieh et al. 2005], and access control [Naldurg and Campbell 2003; Li et al. 2009; Irwin et al. 2008]. The model defines three requirements that a *safety property* needs to satisfy in order to be EM-enforceable [Schneider 2000]. Intuitively, EM-enforceability of a security policy means that the security policy can be applied to transactions observed as the system executes, specifically *i)* the policy can be expressed as a predicate (i.e., an expression evaluated to true or false), *ii)* the decision can be made based on observed transactions and does not depend on any future transactions, and *iii)* the decision can be made after a finite period. Ligatti *et al.* studied non-safety security policies in [Ligatti et al. 2005].

THEOREM 4.1. *User intention-based traffic dependency is a security policy that is execution-monitoring enforceable or EM-enforceable.*

*Proof.* We prove Theorem 4.1 below by showing that user intention-based traffic dependency is a safety property satisfying the aforementioned three requirements (formally in Equations 1, 2, and 3, respectively). Our notation used is shown in Table I.

— Our traffic dependency security policy can be expressed as a predicate as in Equation 1. Given the event $\sigma$ and the current TDG $G$, we define the predicate $\hat{P}$ to evaluate whether or not the parent event of $\sigma$ is in $G$. Specifically $\hat{P}(\sigma)$ can be expressed as whether the parent of event $\sigma$ can be found in the current TDG or not, i.e., $parent(\sigma) \in TDG$.

$$P(\Pi) : (\forall \sigma \in \Pi : \hat{P}(\sigma)) \tag{1}$$

— In our model, the dependency of each event is decided by its past events. The dependency properties of events is not affected by any future events, which is formally expressed in Equation 2. It guarantees that if the event $\tau$ does not have the well formed dependency structure (i.e., $\neg\hat{P}(\tau)$), then $\tau$ followed by event $\sigma$ does not either ($\neg\hat{P}(\tau\sigma)$). In our traffic dependency model, $\tau$ denotes a vagabond traffic event. Such an event is not included in the TDG being constructed.

$$\forall \tau \in \Psi^{-} : \neg\hat{P}(\tau) \Rightarrow (\forall \sigma \in \Psi : \neg\hat{P}(\tau\sigma)) \tag{2}$$

— For a given event, our model can enforce the traffic dependency security and make the decision within a finite period, by applying the predicate to the (finite) prefix of each event. More precisely, the decision can be made within in a time period linear to the number of observed events (within the temporal threshold), which is formally expressed in Equation 3.

$$\forall \sigma \in \Psi : \neg\hat{P}(\sigma) \Rightarrow (\exists i : \neg\hat{P}(\sigma[..i])) \tag{3}$$

Therefore, user intention-based traffic dependency is EM-enforceable.

□

Table I. Notation Table.

| Notation | Definition |
|---|---|
| $P$ | The traffic dependency security policy. |
| $\hat{P}$ | The predicate that is enforceable on events. |
| $\sigma$ | Event (user/traffic event). |
| $\sigma[..i]$ | The prefix of $\sigma$ involving its first $i$ steps. |
| $\tau\sigma$ | The event $\tau$ followed by event $\sigma$. |
| $\Pi$, $\Psi$ | A set of events. |
| $\Psi^{-}$ | The set of all finite prefixes of events in set $\Psi$. |

The security automata in EM model [Schneider 2000] are similar to ordinary nondeterministic finite-state automata, including state variables and transition functions. They serve as the recognizers for sets of executions that satisfy safety properties. They are the basis for enforcement mechanisms in EM. In this section, we describe the security automaton for enforcing traffic dependency in a system. Given some input symbol (e.g., a traffic event) and the current state (e.g., a well-formed TDG), the system is only allowed to perform transitions according to its transition function defined by the

security automaton. If the automaton cannot make a transition on an event, then the system reports that event as an anomaly.

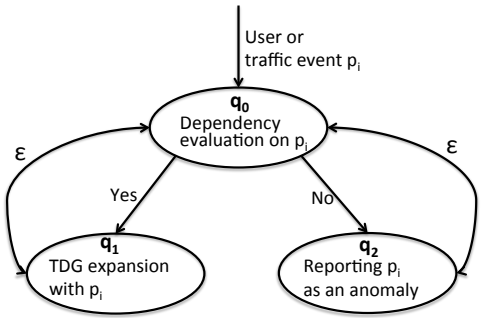



Fig. 3. The security automaton in our model.

We define the security automaton $M = (Q, Q_0, I, \delta)$ by defining its states $Q$, initial state $Q_0$, input symbols $I$, and transition function $\delta$. See also Figure 3. $M$ has three states $Q = \{q_0, q_1, q_2\}$. $q_0$ represents the state where user events and traffic events are consumed and their dependencies are analyzed. $q_0$ is the initial state $Q_0$ of the system. $q_1$ (normal state) is the state where the input has proper dependency. The corresponding data structures such as TDG are augmented in $q_1$. $q_2$ (abnormal state) is the state where the anomalies, specifically events that lack proper dependencies, are found. Anomalies are logged and reported in $q_2$.

The set of input symbols to state $q_0$ is defined as the set of user events and traffic events that can be observed on the system. The events are defined by their attributes, whose values form finite sets. Inputs to states $q_1$ and $q_2$ are boolean variables, which indicate whether or not the input to $q_0$ possesses proper dependency.

The transition function $\delta$ is defined as follows. It is also illustrated in Figure 3. If the input to state $q_0$ is a user event or a traffic event with proper dependency, then transit to state $q_1$, otherwise (i.e., the input is a vagabond traffic event), then transit to state $q_2$. Both $q_1$ and $q_2$ transit back to state $q_0$ with no input.

This formalization in EM framework on traffic dependency security policy is useful for generalizing dependency analysis as a powerful and versatile execution monitoring mechanism. Our work also demonstrates that dependency enforcement being a safety property is practical to deploy.

## 5. DISCUSSION

In this section, we discuss the usability of CR-Miner under complex web scenarios, such as requests to third-party hosts, redirection, AJAX calls, HTTPS traffic and automatic updates. We also thoroughly analyze its security.

### 5.1. CR-Miner Under Complex Web Scenarios

Web browser is the most important and widely used application, with high extensibility and supporting many dynamic features. A traffic dependence analysis needs to properly handle complex web scenarios without generating false alarms. We explain how CR-Miner addresses them.

— *Requests to third-parties* such as `doubleclick` or `facebook` are automatically issued by the browser. The dependencies of these records can be recognized in our experiments, because the referrer fields of the third-party traffic match the hosting domain.

Alternatively, parsing and analyzing the content of proceeding webpages have been used to predict future (legitimate) outbound requests in [Xiong et al. 2009]. We compare our approach with [Xiong et al. 2009] in Section 7.

— *Redirection* allows the browser to issues a HTTP GET request from the URL $A'$ different from the user's original request $A$. The dependency of redirected traffic can be identified by CR-Miner as the request for $A'$ contains the original domain in its referrer. Therefore, CR-Miner can successfully identify the causal relation between $A$ and $A'$.
— *AJAX architecture* allows users to retrieve information from the web server without interfering the display of the current page. Our experiments (with AJAX traces) confirm that CR-Miner handles the AJAX traffic, in terms of discovering the dependencies.
— *HTTPS traffic* contains encrypted HTTPS packet which can be sniffed by a known SSL proxying technique [ proxy]. The technique allows to generate a certificate for the server and signs it with its own root certificate, so that the client's outbound traffic is relied by the man-in-the-middle and can be sniffed inside the HTTPS packets.
— *Automatic updates* do not have explicit user actions that trigger them (e.g., system updates and RSS feeds). One mitigation is to recognize the periodic automatic update traffic with pre-defined or automatically learned whitelisting rules.
— *Cached objects* Browser caching does not affect CR-Miner. According to RFC2616 [ rfc2616], even when the html files are cached by the browser, it still needs to send a request to the server with the freshness of that file. The server may return the code 304 (Not Modified), or the new contents. Thus, CR-Miner still captures all the necessary outbound requests for the dependence analysis.
— *Social engineering attacks* may allow an attacker to fool a user to click on a malicious link and visit an attacker's website, as a result the traffic to attacker's website has the proper traffic dependence and are not deemed suspicious. Therefore, CR-Miner requires additional mechanisms to educate users about social engineering attacks.

### 5.2. Security Analysis

In this section, we answer the question *Can CR-Miner be tricked?* CR-Miner consists of *data collection* and *data analysis* components. Once the data is collected, the dependence analysis may be conducted on a separate trusted machine. Thus, the main security threats come during the data collection phase. Our threat model (in Section 2) considers application-level malware. Therefore, we analyze the security and defense of CR-Miner against two types of attacks: *i) forgery attack* where an adversary modifies attributes of his network activities to make them appear legitimate, and *ii) piggybacking attack* where an adversary strategically determines when to send outbound requests and exploits CR-Miner's temporal rules. We then summarize the effectiveness of CR-Miner in realizing our security goal of identifying anomalous network activities. Our dependence analysis relies on the integrity of the data collected and analyzed, specifically the outbound HTTP header and the user event information.

*5.2.1. Integrity of Traffic Information.* Malware may attempt to spoof the header fields in its outgoing request, e.g., forging its referrer field in the HTTP header so that it appears to be referred by a valid subroot. To prevent this problem, we equip the browser with a *signer*, which implements a lightweight message authentication code to ensure the integrity of the HTTP header created. Then, the signed headers are verified by a trusted program called *verifier* on the same host. The signer and the verifier share secret keys that are used for signing and verification. Traditional key distribution mechanisms, such as the Diffie-Hellman key exchange scheme, can be used to exchange and set up the shared secret key as the operating system starts. Our cryptography-based

verification method effectively prevents this type of forgery, because the headers are tamper-resistant once the browser creates them.

The signer resides in the browser and we implement it in Mozilla Firefox 4.0. We modify the Firefox browser to add a message authentication code (MAC) field to the HTTP header. The MAC prevents the header from being tampered by malware on the host.

For implementation, we define a new HTTP atom `MAC` in `nsHttpAtomList.h` for storing the keyed hash value of the HTTP header. `Init()` in `nsHttpTransaction.cpp` is used to create the whole header. After the HTTP header is generated in `Init()`, we calculate its keyed hash (MD5). Our keyed hash method takes two inputs: the original HTTP header and the symmetric key. The output of the hash function is a 32-digit hexadecimal value, which is stored in the MAC field of the header. Our experiment shows that the overhead for the keyed hash mechanism is negligible.

The verifier is implemented as a stand-alone program on the host outside the browser. HTTP packets that fail the integrity verification are logged. When collecting the outbound traffic packets, the verifier obtains the HTTP headers and peels off the MAC fields to recover the original headers. The verifier recomputes the keyed hash of the original header. If the computed MD5 value is identical to the MAC value found in the HTTP header, the verifier delivers the packet to the traffic module for further processing. Otherwise, the verifier regards the packets as suspicious.

Case I malware spoofing is prevented as spoofed or tampered packets can be detected due to missing valid MAC. Although Case II stand-alone malware is still capable of forging referrers, as it operates independently from the browser and is not subject to the cryptographic verification. Case II malware can be detected based on the rules in previous sections specifying the correlation between the process information.

Our cryptographic operations are orthogonal to those provided by SSL, which is for end-to-end security with a remote server, whereas the purpose of signer and verifier is to guarantee the network packets are not tampered with by malware before leaving the host. In our case, both the signer and verifier reside on the same host. Our above integrity verification solution may bear superficial similarity with the web referral architecture against DDoS [Wang and Reiter 2010]. However, our verification mechanism is specific for web browsers and is designed to protect against stealthy malware on the host.

*5.2.2. Integrity of System Data.* Because user input events are used for identifying subroot events, the integrity of user events is important. Our threat model considers application-level stealthy malware. Therefore, the kernel-level system data, including the process ID, keyboard and mouse events, is trusted. User events may be forged or deleted by user-space application leveraging well-known APIs. To ensure the integrity of system data, advanced keystroke-integrity solutions such as the provenance verification in [Gummadi et al. 2009; Hasan et al. 2009; Stefan et al. 2010] can be incorporated in CR-Miner to further improve system-data assurance, which is a useful fail-safe mechanism to guard against potential operational errors. These methods provide provable assurance to the origin of user events and effectively prevent event forgery (i.e., injection of fake user input events). Hasan *et al.* [Hasan et al. 2009] show how to provide strong integrity and confidentiality assurances for data provenance information at the kernel, file system, or application layer, which may be applied to prevent the forgery of user events in CR-Miner. Existing solutions for detecting clickjacking attacks (e.g., [Balduzzi et al. 2010]) also allow one to defeat fake mouse clicks.

Process information (such as PID) can be obtained through known APIs (Windows Hook API and IP Helper API). The user-space malware cannot forge process information without Administrator/Root privilege. Recent work on cryptographic identification

of natives applications in operating system prevents the forgery of process ID information [Almohri et al. 2012]. Robust host-based rootkit prevention remains an active research problem.

*5.2.3. Defense Against Piggybacking Attack.* In a piggybacking attack during the data collection, the adversary sends outbound network requests (to the attacker's server) immediately after a legitimate traffic event. Such an attack would be effective in a naive temporal-only analysis. However, our dependency rules inspect the context and property of traffic such as domain names, referrers, and PIDs. Therefore, piggybacking requests can be easily detected as vagabond events, as malware traffic lacks the required attributes. We compare our detection accuracy with the temporal-only analysis in Section 6.3. Similar piggybacking attacks are discussed by Xu *et al* in [Xu et al. 2011] in the context of detecting drive-by-download attacks.

*Summary* The goal that we set for our technique in Section 2 is to be able to identify vagabond outbound traffic events for anomaly detection. Stealthy malware typically causes vagabond outgoing traffic that has no dependence on user activities. Our Algorithm 1 provides an iterative mechanism to construct complete user intention-based TDG, through which vagabond events can be identified.

Robustness and accuracy are the two complementary security properties in our context. We have analyzed the robustness of CR-Miner above against several potential circumvention attacks. The accuracy of CR-Miner relies on the accuracy of inferring parent-child relations in Algorithm 1, which we extensively evaluate in our experiments next, among which we demonstrate the effectiveness of our traffic dependency based anomaly detection against stealthy malware activities.

## 6. IMPLEMENTATION AND EVALUATION

We describe the prototype implementation of CR-Miner in Section 6.1. The questions we seek to answer through our experiments are:

— How accurate is our dependency inference algorithm in identifying dependencies in outbound HTTP requests?
— How efficient is the BFS based dependency inference algorithm?
— How much better in accuracy is our algorithm compared to a temporal-only dependence analysis?
— Does the inference accuracy suffer in noisy traffic?
— Can we detect real-world stealthy malware traffic using the traffic-dependency graph?

### 6.1. Prototype Implementation

We develop a CR-Miner prototype in Windows 7 operating system. The detailed architecture of our prototype is shown in Figure 4. The CR-Miner prototype is easy to adopt and does not require any modification to the browser in order to taint or track the dependencies. We build CR-Miner (the darker parts in Figure 4) between the application and the kernel layers.

There are three sensors deployed to collect data on the host. The causal relation analyzer computes the dependencies based on the rules and algorithms in Section 3. The Windows APIs (namely hook API, IPHelper API, and libpcap API) are used in the implementation. Signer and verifier are a pair of tools in order to guarantee the integrity of the HTTP headers, as we discussed in Section 5.2.1. Our implementation details are described next, including process identification, i.e., identifying the process ID associated with an observed network flow, traffic monitoring, and user-action collection.

The traffic module implemented with the `SharpPcap` library filters the network packets to record outbound HTTP GET requests. We store the packet information in

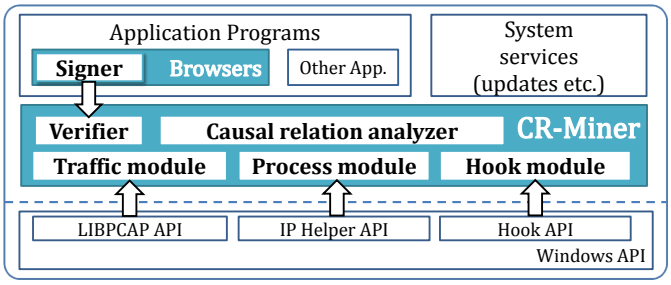



Fig. 4. Architecture of CR-Miner prototype.

the quadruple <source IP, source port, destination IP, destination port>. The process module obtains network and system (namely process) information about active connections. We obtain the IP table, a kernel data structure in Windows, by using `GetExtendedTcpTable()` in `IPHelperAPI.dll` and map TCP connections to corresponding process IDs.

The hook module sets up system hooks in order to collect kernel-level user events to the application. Our module, using the existing Windows Hook API, installs the hooks to log keyboard and mouse events (including mouse click, mouse double click, mouse wheel, and key press). Furthermore, we obtain the process ID of the current foreground window by using `GetWindowThreadProcessId()`, so that we find out the corresponding process for each user event. Repetitive user events that do not generate traffic such as mouse movements are ignored. `Key Down`, `Key Press`, and `Key Up` events are repetitive and only `Key Press` events are recorded.

We also record user events at the application level through the use of `Tlogger`. It is a Firefox extension for capturing the information of mouse clicks during web browsing [ tlogger], including the navigation and tab events. The information gathered by the Tlogger is complementary to the data recorded by the kernel hook module. Tlogger provides the URL information associated with mouse-click events, which is unavailable at the hook level. Attributes of user events collected at the application level and kernel level are both used for identifying the subroot nodes in the *Is_Subroot* procedure.

### 6.2. Accuracy of Dependency Inference

We conducted a user study with 20 participants to collect samples of HTTP traces. Each participant was asked to actively browse the Internet for 30 minutes on a laptop installed with CR-Miner. Participants were asked not to reveal any sensitive personal data such as passwords during the study. The means and standard deviations of the number of events that we collected are shown in Table II. The number of traffic-generating user events is much fewer than the total user events observed. Because our *Is_Subroot* analysis is based on traffic-generating user events, it is quite efficient. We use a SQL Server database to store the records. We measure the size of the database BAK file for each user.

We define *hit rate* as the ratio of the number of legitimate requests identified by CR-Miner to the total number of HTTP GET requests per user. A legitimate HTTP GET request needs to belong to a valid tree in the traffic-dependency graph rooted by a user event. The distribution of hit rates in our user studies is shown in the Table III. For 85% of the users, their hit rates are above 99.0%, which indicates the high accuracy of our prediction. The average hit rate of the user studies is 99.6%. We manually inspected the dependencies to ensure their correctness.

Table II. Mean and standard deviations (SD) of statistics of records collected in the user study for each user.

| | User Events | | Traffic Events | Process Record | DB File Size (KB) |
|---|---|---|---|---|---|
| | Traffic-generating | Total | | | |
| Mean | 61 | 2761 | 2357 | 1178 | 3221 |
| SD | 28 | 1768 | 1204 | 686 | 608 |

Table III. The distribution of hit rates across 20 user cases.

| Hit Rate | Frequency | Percentage |
|---|---|---|
| $0.98 \leq r < 0.985$ | 1 | 5% |
| $0.985 \leq r < 0.99$ | 2 | 10% |
| $0.99 \leq r < 0.995$ | 4 | 20% |
| $0.995 \leq r < 1.00$ | 10 | 50% |
| $r = 1.00$ | 3 | 15% |

We calculate the percentage of requests whose dependencies are inferred by *Is_Subroot*, *Is_Child*, and *Is_Sibling*, respectively. The result in Table IV shows that most of the dependent relations (87.4%) are inferred by the *Is_Subroot* procedure. We also utilize a whitelist in our experiment. The whitelist is constructed based on four categories: *software-update traffic*, *requests for traffic analytics*, *trustworthy web portals*, and *legitimate advertisement websites*. Details are not shown due to space limit. The construction of the whitelist reduces false alarms, as it allows the dependence identification of outbound requests that may have incomplete attributes. However, our algorithm cannot be replaced by a pure whitelist approach because of the diversity nature of the Internet traffic.

Table IV. Percentages of requests inferred by different subroutines for 20 user cases.

| Category | | Percentage |
|---|---|---|
| Inferred Dependency | Is_Subroot | 1.9% |
| | Is_Child | 87.4% |
| | Is_Sibling | 8.6% |
| | Whitelisting | 1.7% |
| | **Total** | **99.6%** |
| Missing Dependency | | 0.4% |

We further investigate the outbound requests with missing dependencies, which account for 0.4% of the total traffic as shown in Table IV. The main reason of having these vagabond requests is missing referrer, which may be due to either the use of dereferrer by a website for privacy purpose or an HTTP connection being referred by an HTTPS site [H 1999]. Some of the vagabonds are legitimate requests, i.e., false positives. Whereas, others are requests to known blacklisted websites [ blocked], e.g., `atwola.com`, `adadvisor.net`, and `pixel.quantserve.com`.

### 6.3. Accuracy Comparison With Temporal-Only Analysis

We compare CR-Miner with a temporal-only dependence analysis that infers dependencies based solely on the intervals and PIDs of requests. Such a temporal-only dependence analysis is used in BINDER [Cui et al. 2005]. We filter non-subroot requests by an interval threshold $\tau$, instead of using *Is_Child* and *Is_Sibling*. We define *precision* as the ratio of the number of legitimate requests identified in the temporal-only algorithm to that of CR-Miner. That is, it is the percentage of dependent relations that

can be successfully identified by the temporal-only method, out of the ones found by CR-Miner. Our comparison results are shown in Figure 5. The longer the threshold $\tau$ is, the larger the chance that a request can find its parent node, as observed in our experiments. A low precision value means the low accuracy of temporal-only method compared to CR-Miner.

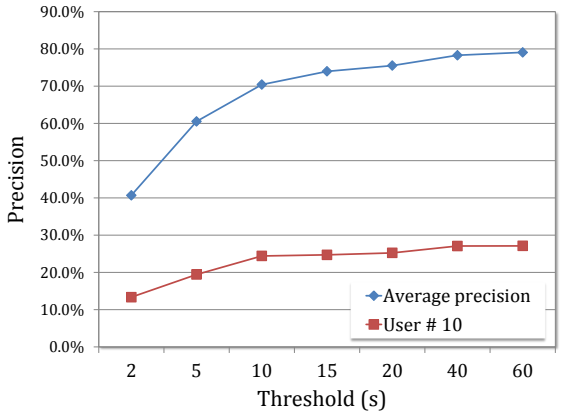



Fig. 5. (Poor) dependency inference accuracy by the temporal-only analysis compared to our BFS based algorithm, for a specific user (#10) and the averaged value from 19 other users.

When the actual delay between a request and its subroot is longer than the threshold, which may be due to the AJAX technique, the temporal-only prediction suffers. For example, a participant (user # 10) frequently visited Google map. That case yields the lowest precision value as shown in Figure 5. Figure 5 also shows the averaged precisions of the other 19 users. The results show that the temporal-only analysis is limited in predicting traffic dependency, especially when the interval threshold is small. When the actual delay between a request and its subroot is longer than the threshold – which may be due to the AJAX technique – the temporal-only prediction suffers. CR-Miner with a threshold of 15 seconds achieves the average hit rate as high as 99.6%. Hence, our algorithm substantially outperforms temporal-only algorithm in terms of the accuracy of identifying traffic dependencies.

### 6.4. Prediction Accuracy Under Multi-user Condition

Cross-user validation is to measure the accuracy of our analysis under the noisy traffic. We arbitrarily introduce noises by merging two users' records, and to infer traffic dependencies in merged datasets.

We choose five independent user datasets, and create 10 cross-user data sets by choosing independent records and merging them ($5C_2 = 10$). Rather than shuffling two user studies and combining them, we merge the two user studies without losing their internal orders. The merging algorithm is similar to *Merge Sort*. We add a component to merge two lists without changing the chronological ordering of events in each list. We then run the BFS based DI algorithm on the mixed data. We define the *error rate* as the percentage of traffic events whose parent nodes in the cross-user study are different from those found in the regular analysis. To compare the consistency of dependencies in the two analyses, we recursively locate the subroot for each request. We run ten cross-user tests which are composed from five independent user studies by

BFS based DI algorithms. The average error rate is 0.8%. The cross-user validation shows a high prediction accuracy of CR-Miner even under multi-user condition.

### 6.5. Time Efficiency Comparison

In order to compare the run-time efficiency of our BFS based dependency inference (DI) algorithm, we implement an alternative sequential traversal DI algorithm. The sequential algorithm stores the outbound requests with known dependencies as a list, and infers the new request's dependency by scanning the list. This sequential traversal algorithm serves as a baseline in our efficiency comparison.

We evaluate the BFS based and sequential traversal algorithms on a machine equipped with Intel Core 2 Quad 2.40 GHz and 2GB system memory. Both algorithms are used to analyze the traffic of 20 users. The running-time for each algorithm is presented in Figure 6. The sequential algorithm takes about 3 times as long as the BFS based one. Thus, our BFS based DI algorithm runs significantly faster. The advantage in run-time efficiency of our BFS based algorithm comes from its optimization for the data structure and the order of comparison. We also confirm that for each user case both algorithms yield the exact same hit rates.

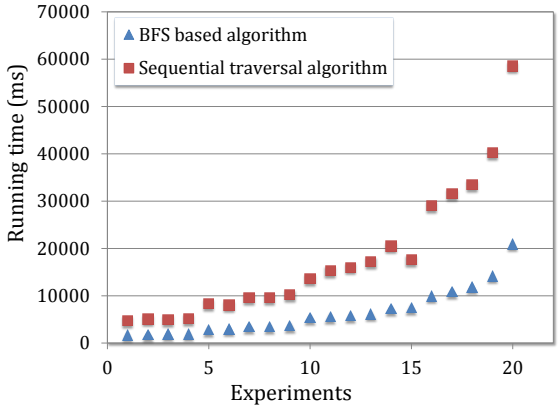



Fig. 6. The run time of BFS based and sequential dependency inference algorithms for each of the 20 user cases (X-axis).

### 6.6. Real-World Spyware Detection

We use our CR-Miner to detect two pieces of real-world malware, namely `Infostealer.Maximus` and `Trojan.Brojack.A`, and a proof-of-concept malicious Firefox extension. `Infostealer.Maximus` sends out two requests at the same time to one host (`www.scieki.com.pl`) to retrieve two executable files (`/css/k2pac.exe` and `/css/w2pac.exe`) when it is active. The requested files are trojan downloaders, which can be downloaded and installed without the user's knowledge. Therefore, `Infostealer.Maximus` is case I malware as defined in Section 2.

`Trojan.Brojack.A` [ brojack] not only modifies the registry entries and captures all links that are browsed by the user, but also sends out outbound traffic to a host (`watson.microsoft.com`). The malware runs with an independent PID and sends out outbound HTTP requests to a malicious host. `Trojan.Brojack.A` is case II malware. Neither piece of malware carries appropriate referrers. CR-Miner successfully detects both

malware. It flags the malware traffic as vagabond, as these requests are not rooted by any user event in TDG.

We also wrote and evaluated a proof-of-concept malicious Firefox extension, which is a piece of password-stealing spyware. When a user clicks on the `Submit` button of any web form in the browser, the extension finds the non-blank password filled in the form and sends out an outbound HTTP request with the password as a parameter to the attacker's server. Our spyware is similar to the known spyware such as `FormSpy`, `FireSpyFox`, and `FFsniff`. CR-Miner successfully identifies the traffic sent by this malicious extension. Dependencies of events found by CR-Miner are partly shown in Table V. The experiment procedure is described next.

Table V. Spyware traffic (with ID 23) is identified as it has -1 in its Parent ID field, indicating no dependence with existing traffic or user events.

| ID | Timestamp | PID | Parent ID | Host | Referrer | Http Request |
|---|---|---|---|---|---|---|
| ... | | | | | | |
| 15 | 0:0:51.447 | 5668 | 0 | `mail.yahoo.com` | | / |
| ... | | | | | | |
| 21 | 0:0:52.843 | 5668 | 15 | `view.atdmt.com` | `http://www.yahoo.com/` | `/M0N/view/307963403/direct...` |
| 22 | 0:0:54.843 | 5668 | 15 | `mail.yahoo.com` | `http://www.yahoo.com/` | `/;_ylt=AsDjYtx1zxEscRUUSbl...` |
| **23** | 0:1:03.313 | 5668 | **-1** | `www.attacker.net` | | `/query?id=user&ps=password` |
| ... | | | | | | |

The user types `mail.yahoo.com` (corresponding to record # 15) into the browser's address bar and the browser issues the request. The Parent ID, which is 0, indicates it as a subroot. The requests for other objects from Yahoo and other providers (e.g., # 21 and # 22), which are legitimate requests issued by browser, have 15 in Parent ID fields. The user then intends to log in the Yahoo Mail account by entering the user ID and password. Due to the spyware, upon the user clicking on the submit button, a single outbound HTTP GET request with the stolen login credential is sent to the attacker's server, `www.attacker.net` in our example. The CR-Miner detects the spyware activity and identifies it (# 23) as vagabond, which has -1 in its Parent ID field. The spyware activity is not qualified to be a subroot as its domain name does not match any valid user event, and its domain name is not referred by any proceeding requests. By finding dependencies in traffic, we show that our solution renders spyware and keyloggers useless, as their outbound communication channels are blocked.

## 7. RELATED WORK

The research on the interplay between human behaviors and system properties has not been extensively studied in the context of anomaly detection, with a few exceptions [Cui et al. 2005; Gummadi et al. 2009; Lu et al. 2010; Shirley and Evans 2008; Xu et al. 2011]. Our work was inspired by BINDER [Cui et al. 2005]. It is a host-based solution that detects break-ins on personal computers through analyzing the dependency of traffic events based on temporal and process information. In comparison, our CR-Miner framework describes a more powerful user intention-based approach that supports application-specific dependence analysis with a much finer granularity. The advantage of our solution is its accuracy. For example, BINDER treats traffic events equally without inspecting its content, consequently it does not defend against piggybacking attacks. Our analysis can semantically distinguish legitimate HTTP requests from suspicious ones and can successfully detect piggybacking traffic.

BLADE [Lu et al. 2010], UIBAC [Shirley and Evans 2008], and DeWare [Xu et al. 2011] leverage user behaviors for certain file-access regulation. In UIBAC (user intention-based access control) Shirley and Evans proposed to analyze the correlation between user actions and subsequent program events to extract rules, and enforce these rules for controlling the access of system resources. BLADE prevents drive-by download by enforcing the dependence between mouse clicks and the file creation of a browser. DeWare describes a more general framework for enforcing the host-wide file-system dependency, which can be used to detect the onset of infection. Our work in CR-Miner is for analyzing traffic events, as opposed to file-system events in above three solutions. Therefore, we address new technical challenges. Not-A-Bot (NAB) is a system for authenticating traffic-generating user inputs such as mouse clicks on hyperlinks [Gummadi et al. 2009]. It can be used for defeating DDoS attacks as well as click fraud. NAB does not analyze the dependencies among network packets for anomaly detection as CR-Miner does. As explained in Section 5.2, CR-Miner can use NAB and similar techniques (e.g., [Xu et al. 2012]) to ensure the integrity of user inputs collected.

Despite these recent advances toward user behavior inspired security, many important problems have not been studied, including general challenges, requirements, models, theory, practical applications, and limitations for user intention based traffic anomaly detection. Our work in this paper systematically and thoroughly addresses these aspects, and our solutions will have high impact beyond the specific web anomaly problem. We present not only a practical tool CR-Miner for building accurate and fine-grained dependencies in HTTP traffic on a host, but also a powerful FSA-based abstraction for representing and enforcing dependency rules for networked applications such as the browser. We further describe the enforceability property of traffic dependency following Schneider's powerful and well-known EM (execution monitoring) framework. Our work aims at fully understanding and demonstrating the capabilities of user intention-based dependency analysis as a general methodology for monitoring and anomaly detection.

Below we survey recent related work on identifying malicious or anomalous network traffic patterns including both host-based and network-based analysis through graph analysis, statistics analysis, or system engineering approach. King *et al.* analyzed logs of intrusion detection systems (IDS) such as SNORT across multiple hosts to track traffic dependence [King et al. 2005]. The dependence graphs are formed by correlating IP addresses in IDS entries. Zhou *et al.* [Zhou et al. 2010] built a general-purpose event correlation mining system and proposed failure correlation graphs to inspect clustered systems. Our work differs from [King et al. 2005; Zhou et al. 2010] in that our traffic dependence analysis is user-related and application-specific. Although these solutions involve representing dependencies in directed graphs, our dependency analysis is application-specific and fine-grained.

WebTap, developed by Borders and Prakash [Borders and Prakash 2004], is a tool to anomalous patterns in outbound HTTP traffic. WebTap identifies anomalies in outbound HTTP traffic by monitoring the metrics such as request regularity, bandwidth usage, inter-request delay time, and transaction size. The authors further improved the detection accuracy by pruning repetitive information (e.g., header fields) in [Borders and Prakash 2009]. Their anomaly detection approaches aim to identify changes and deviations in *aggregated* flow patterns in terms of usages with statistical metrics. The main difference between our user intention-based anomaly detection approach and theirs is that we do not require any knowledge of behavior patterns of any user groups, and our rules are derived from the properties of applications.

Moshchuk *et al.* proposed the execution-based web content analysis in SpyProxy [Moshchuk et al. 2007] that renders and observes active web content in a

disposable virtual machine before it is allowed to reach a user's browser. Li *et al.* [Li et al. 2011] designed WebShield, a middlebox framework hosted on a proxy of the user for guarding against malicious JavaScripts. The above solutions are for detecting server-side malicious web content, whereas our solution is focused on client-side anomalies. CR-Miner only collects outbound requests and does not collect or analyze incoming responses.

The anomaly detection approach in [Xiong et al. 2009] aims at predicting anticipated outbound HTTP requests by parsing retrieved web pages that a user requests and prefetching the objects. Their work aims at creating a tool that serves as a parallel universe to the browser, so that one can predict the legitimacy of outgoing HTTP requests. The tool is for detecting corrupted browser or extensions. The discrepancies in traffic events from the parallel universe and the browser are reported as anomalies. Because the predictor needs to independently fetch webpages, it doubles the bandwidth overhead. In comparison, our work does not parse or analyze web content and thus can handle any types of websites. Our rule-based dependence-finding is more efficient in network bandwidth usage.

Srivastava and Giffin [Srivastava and Giffin 2010] presented a technique for discovering the origin of parasitic malware on a host through sophisticated OS-level diagnostic. It instruments sensors for collecting and reasoning about various types of system data, and works with existing network IDS for identifying the malicious activities. Their solution can be used to pinpoint the origin of the malware, after CR-Miner has identified its stealthy traffic. Malware detector [Christodorescu et al. 2008] is an automatic technique to mine the malicious behavior from the known malware and use the knowledge to detect the variants. This type of code-analysis solutions nicely complements to the proposed network-based analysis method.

## 8. CONCLUSIONS AND FUTURE WORK

Analyzing the dependencies between network traffic and user activities has not been systematically investigated as a general approach for anomaly detection. Our traffic-dependency graph captures the causal relations of user actions and network events for improving host integrity. We performed extensive experimental evaluation on CR-Miner. Our results indicate the feasibility of enforcing HTTP traffic dependencies.

Currently, we are investigating how to generalize our approach to include the analysis of outbound DNS traffic in the TDG. DNS traffic is involved in almost all networked applications. Our current analysis is focused on outbound HTTP requests. One needs to construct TDGs of requests of different types across multiple network layers and applications. Another thrust of our research is on how to automatically extract dependency rules using data-mining techniques, including basic techniques such as association rule mining [Lee and Stolfo 1998] and frequent episode mining [Patnaik et al. 2010]. We also plan to explore redescription mining [Zaki and Ramakrishnan 2005] and storytelling [Kumar et al. 2008] techniques for semantic modeling of application characteristics.

## REFERENCES

ALMOHRI, H. M. J., YAO, D. D., AND KAFURA, D. G. 2012. Identifying native applications with high assurance. In *ACM Conference on Data and Application Security and Privacy (CODASPY)*. ACM, 275–282.

AN, X., JUTLA, D. N., AND CERCONE, N. 2006. Privacy intrusion detection using dynamic Bayesian networks. In *International Conference on Electronic Commerce (ICEC)*. 208–215.

BALDUZZI, M., EGELE, M., KIRDA, E., BALZAROTTI, D., AND KRUEGEL, C. 2010. A solution for the automated detection of clickjacking attacks. In *ASIACCS*. 135–144.

blocked. Top 10 Blocked URLs and Domains. http://us.trendmicro.com/us/trendwatch/current-threat-activity/malicious-url-info/top10-blocked-urls-domains/.

BORDERS, K. AND PRAKASH, A. 2004. Web Tap: Detecting covert web traffic. In *Proceedings of the 11th ACM Conference on Computer and Communication Security*. 110–120.

BORDERS, K. AND PRAKASH, A. 2009. Quantifying information leaks in outbound web traffic. In *Proceedings of the IEEE Symposium on Security and Privacy*.

brojack. Technical Details about Trojan.Brojack. https://www.symantec.com/en/uk/security_response/writeup.jsp?docid=2008-070310-5229-99.

BURSZTEIN, E. AND GOUBAULT-LARRECQ, J. 2007. A logical framework for evaluating network resilience against faults and attacks. In *Computer and Network Security, 12th Asian Computing Science Conference (ASIAN)*. 212–227.

CHANDOLA, V., BANERJEE, A., AND KUMAR, V. 2009. Anomaly detection: A survey. *ACM Comput. Surv. 41*, 15:1–15:58.

CHRISTODORESCU, M., JHA, S., AND KRUEGEL, C. 2008. Mining specifications of malicious behavior. In *ISEC*, G. Shroff, P. Jalote, and S. K. Rajamani, Eds. ACM, 5–14.

CUI, W., KATZ, Y. H., AND TIAN TAN, W. 2005. Binder: An extrusion-based break-in detector for personal computers. In *In Proceedings: USENIX Annual Technical Conference*. 4.

DENNING, D. E. 1987. An intrusion-detection model. *IEEE Transactions on Software Engineering SE-13,* 2.

GUMMADI, R., BALAKRISHNAN, H., MANIATIS, P., AND RATNASAMY, S. 2009. Not-a-Bot: Improving service availability in the face of botnet attacks. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NDSI)*.

H 1999. Hypertext transfer protocol – http/1.1. http://tools.ietf.org/html/rfc2616#section-15.1.3.

HASAN, R., SION, R., AND WINSLETT, M. 2009. Preventing history forgery with secure provenance. *TOS 5,* 4.

HEBERLEIN, L. T., DIAS, G. V., LEVITT, K. N., MUKHERJEE, B., WOOD, J., AND WOLBER, D. 1990. A network security monitor. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*. 296–304.

IRWIN, K., YU, T., AND WINSBOROUGH, W. H. 2008. Enforcing security properties in task-based systems. In *Proceedings of the 13th ACM symposium on Access control models and technologies*. SACMAT '08. ACM, New York, NY, USA, 41–50.

JUNG, J., SHETH, A., GREENSTEIN, B., WETHERALL, D., MAGANIS, G., AND KOHNO, T. 2008. Privacy oracle: a system for finding application leaks with black box differential testing. In *Proceedings of Computer and Communications Security (CCS)*.

KING, S. T., MAO, Z. M., LUCCHETTI, D. G., AND CHEN, P. M. 2005. Enriching intrusion alerts through multi-host causality. In *Proceedings of Network and Distributed System Security (NDSS)*.

KUMAR, D., RAMAKRISHNAN, N., HELM, R. F., AND POTTS, M. 2008. Algorithms for storytelling. *IEEE Trans. Knowl. Data Eng. 20,* 6, 736–751.

LANGAR, M., MEJRI, M., AND ADI, K. 2011. Formal enforcement of security policies on concurrent systems. *J. Symb. Comput. 46,* 9, 997–1016.

LEE, W. AND STOLFO, S. J. 1998. Data mining approaches for intrusion detection. In *Proceedings of the 7th conference on USENIX Security Symposium - Volume 7*. SSYM'98. USENIX Association, Berkeley, CA, USA, 6–6.

LI, N., WANG, Q., QARDAJI, W., BERTINO, E., RAO, P., LOBO, J., AND LIN, D. 2009. Access control policy combining: theory meets practice. In *Proceedings of the 14th ACM symposium on Access control models and technologies*. SACMAT '09. ACM, New York, NY, USA, 135–144.

LI, Z., TANG, Y., CAO, Y., RASTOGI, V., CHEN, Y., LIU, B., AND SBISA, C. 2011. Webshield: Enabling various web defense techniques without client side modifications. In *NDSS*. The Internet Society.

LIGATTI, J., BAUER, L., AND WALKER, D. 2005. Enforcing non-safety security policies with program monitors. In *Proceedings of the 10th European conference on Research in Computer Security*. ESORICS'05. Springer-Verlag, Berlin, Heidelberg, 355–373.

LU, L., YEGNESWARAN, V., PORRAS, P., AND LEE, W. 2010. BLADE: An attack-agnostic approach for preventing drive-by malware infections. In *Proceedings of 17th ACM Conference on Computer and Communications Security*.

MOSHCHUK, A., BRAGIN, T., DEVILLE, D., GRIBBLE, S. D., AND LEVY, H. M. 2007. Spyproxy: Execution-based detection of malicious web content. In *Proceedings of the 16th Annual USENIX Security Symposium (USENIX Security 2007)*. Boston, MA.

NALDURG, P. AND CAMPBELL, R. H. 2003. Dynamic access control: preserving safety and trust for network defense operations. In *Proceedings of the eighth ACM symposium on Access control models and technologies*. SACMAT '03. ACM, New York, NY, USA, 231–237.

PATNAIK, D., LAXMAN, S., AND RAMAKRISHNAN, N. 2010. Discovering excitatory relationships using dynamic Bayesian networks. *Knowledge and Information Systems*, 1–31. 10.1007/s10115-010-0344-6.

proxy. An SSL proxying technique to sniff HTTPs packets. `http://www.charlesproxy.com/documentation/proxying/ssl-proxying/`.

rfc2616. RFC 2616. Caching in HTTP. `http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.1.1`.

SCHNEIDER, F. B. 2000. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC) 3,* 1, 30–50.

SCHNEIDER, F. B., MORRISETT, J. G., AND HARPER, R. 2001. A language-based approach to security. In *Informatics - 10 Years Back. 10 Years Ahead.* Springer-Verlag, London, UK, UK, 86–101.

SEKAR, R., RAMAKRISHNAN, C. R., RAMAKRISHNAN, I. V., AND SMOLKA, S. A. 2001. Model-carrying code (mcc): a new paradigm for mobile-code security. In *Proceedings of the 2001 workshop on New security paradigms*. NSPW '01. ACM, New York, NY, USA, 23–30.

SEKAR, R., VENKATAKRISHNAN, V., BASU, S., BHATKAR, S., AND DUVARNEY, D. C. 2003. Model-carrying code: a practical approach for safe execution of untrusted applications. *SIGOPS Oper. Syst. Rev. 37,* 5, 15–28.

SHIEH, A., WILLIAMS, D., SIRER, E. G., AND SCHNEIDER, F. B. 2005. Nexus: a new operating system for trustworthy computing. In *Proceedings of the twentieth ACM symposium on Operating systems principles*. SOSP '05. ACM, New York, NY, USA, 1–9.

SHIEH, S.-P. AND GLIGOR, V. D. 1997. On a pattern-oriented model for intrusion detection. *IEEE Transactions on Knowledge and Data Engineering 9,* 4.

SHIRLEY, J. AND EVANS, D. 2008. The user is not the enemy: Fighting malware by tracking user intentions. In *NSPW '08: Proceedings of the 2008 workshop on New security paradigms*. 33–45.

SNAPP, S. R., BRENTANO, J., DIAS, G. V., GOAN, T. L., GRANCE, T., HEBERLEIN, L. T., HO, C.-L., LEVITT, K. N., MUKHERJEE, B., MANSUR, D. L., PON, K. L., AND SMAHA, S. E. 1991. A system for distributed intrusion detection. *COMPCOM Spring '91 Digest of Papers*, 170–176.

SRIVASTAVA, A. AND GIFFIN, J. T. 2010. Automatic discovery of parasitic malware. In *Recent Advances in Intrusion Detection (RAID)*. 97–117.

STEFAN, D., WU, C., YAO, D., AND XU, G. 2010. Cryptographic provenance verification for the integrity of keystrokes and outbound network traffic. In *Proceedings of the 8th International Conference on Applied Cryptography and Network Security (ACNS)*.

SUN, W., SEKAR, R., LIANG, Z., AND VENKATAKRISHNAN, V. N. 2008. Expanding malware defense by securing software installations. In *Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. DIMVA '08. Springer-Verlag, Berlin, Heidelberg, 164–185.

TENG, H. S., CHEN, K., AND LU, S. C.-Y. 1990. Adaptive real-time anomaly detection using inductively generated sequential patterns. *Security and Privacy, IEEE Symposium on 0*, 278.

tlogger. TLogger – An Firefox Extension. `http://dubroy.com/tlogger/`.

WANG, X. AND REITER, M. K. 2010. Using web-referral architectures to mitigate denial-of-service threats. *IEEE Trans. Dependable Sec. Comput. 7,* 2, 203–216.

XIONG, H., MALHOTRA, P., STEFAN, D., WU, C., AND YAO, D. 2009. User-assisted host-based detection of outbound malware traffic. In *Proceedings of International Conference on Information and Communications Security (ICICS)*.

XU, K., XIONG, H., WU, C., STEFAN, D., AND YAO, D. 2012. Data-provenance verification for secure hosts. *IEEE Transactions on Dependable and Secure Computing 9*, 173–183.

XU, K., YAO, D., MA, Q., AND CROWELL, A. 2011. Detecting infection onset with behavior-based policies. In *Proceedings of the Fifth International Conference on Network and System Security (NSS)*.

YIN, H., SONG, D., EGELE, M., KRUEGEL, C., AND KIRDA, E. 2007. Panorama: Capturing system-wide information flow for malware detection and analysis. In *In Proceedings of the 14th ACM Conferences on Computer and Communication Security (CCS)*.

ZAKI, M. J. AND RAMAKRISHNAN, N. 2005. Reasoning about sets using redescription mining. In *KDD*, R. Grossman, R. J. Bayardo, and K. P. Bennett, Eds. ACM, 364–373.

ZHOU, W., ZHAN, J., MENG, D., XU, D., AND ZHANG, Z. 2010. Logmaster: Mining event correlations in logs of large scale cluster systems. *CoRR abs/1003.0951*.