User Intention-Based Traffic Dependence Analysis For Anomaly Detection

Hao Zhang, William Banick, Danfeng (Daphne) Yao, *Member, IEEE,* Naren Ramakrishnan, *Member, IEEE,*
Department of Computer Science, Virginia Tech

***Abstract*—This paper describes an approach for enforcing 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 alarms. Our prototype can successfully detect several pieces of HTTP-based real-world spyware. Our dependence analysis is fast with a minimal storage requirement. We give a thorough analysis on the security and robustness of the user intention-based traffic dependence approach.**

I. Introduction

Anomalies or outliers refer to any activities that do not conform to regular ones. Statistical techniques modeled under specific domain knowledge have been proposed for anomaly detection [8], [11], [17], [19]. For example, dynamic Bayesian networks can be used to detect abnormal data access patterns by malicious insiders to a sensitive database [1]. However, realizing general anomaly detection is challenging, especially for complex and diverse behaviors involving activities spanning users, hosts, and networks.

We describe a novel semantics-based approach 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. Our analysis requires the specifications of an application, based on which we extract and enforce policies defining dependencies within the application. 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. We enforce the correct system behaviors, as opposed to anomalous characteristics. Our dependence-based anomaly detection has advantages over conventional pattern-based anomaly detection solutions (such as [5], [6], [8], [22]), 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* 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 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.
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 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.
3) 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. The message authentication code we adopt in CR-Miner improves the integrity of CR-Miner against stealthy malware's tampering.

Our user intention-based traffic dependence analysis produces structures in network events. These structures across outbound requests enable the 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 semantic dependencies among events is a general anomaly detection technique, which can be also applied to detect anomaly in file-system events.

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.

II. Traffic-Dependency Graph

Discovering user intention-based traffic dependency 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. Next, we give definitions used in our model and an example illustrating the traffic dependence among the events.

A. Dependency 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 1: 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.

The tree-based TDG enables us to apply breadth-first traversal when inferring dependencies, which is described in Section III-A.

User event refers to the user's inputs to the application through the 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 IV. 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.

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. Traffic events are further categorized into different levels according to their relative dependencies. We use the phrases traffic event and network request interchangeably.

Subroot is a special type of traffic events. 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. In Figure 1, the traffic events 1, 3,

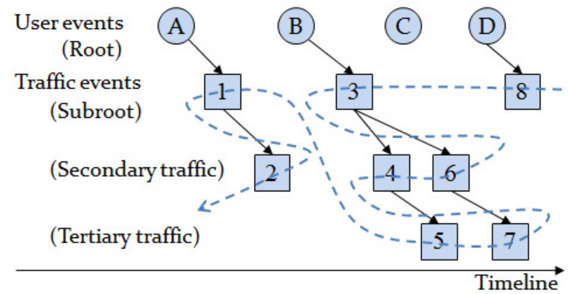

Fig. 1. An example of a traffic-dependency graph. Solid arrows indicate dependency relations, and the dotted arrow indicates the breadth-first traversal in the dependency inference.

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 . Thus, *tertiary traffic* (e.g., events 5 and 7 in Figure 1) and lower-level traffic can be similarly categorized.

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.

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

Definition 2: In the user intention-based traffic dependency model, a *legitimate* traffic event belongs to a tree in the traffic-dependency graph 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.

B. 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 [20]. 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 V-F.

- Software, which comes from unknown or untrusted developers, may perform undesirable network activities that are not causally related to the user's inputs due to errors or flaws. Identifying these stealthy unwanted traffic is important, as these packets may leak information of the user (e.g., [12]), consume bandwidths, and cause further security vulnerabilities. Legitimate automated traffic, such as system updates and RSS feeds, can be whitelisted (See also Section V-B).

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 by Bursztein and Goubault-Larrecq on dependencies of services [4].

III. Construction of TDG

The goal of CR-Miner is to identify structured dependencies (or lack thereof) in network traffic, which are used to detect anomalous events. A TDG is 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 semantics.

A. 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 at user events in the existing TDG. The subroots, thus, are also chronologically ordered.

Our algorithm opts for a breadth-first traversal within a tree starting from the most recent subroot for the following two reasons. 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.

We run 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_Sibling and Is_Child tests are run. If the dependence is not found after all nodes on the tree are compared, then the next subroot 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 this 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.

We further optimize the algorithm by skipping unnecessary comparisons. We achieve the speedup by leveraging the underlying consistency among attributes (e.g., PID) of nodes on the same tree in TDG. In the context of our dependence analysis, our dependency rule for parent-child relation requires the timestamps of the two events fall within a threshold τ. We consider the timestamp of the new request (p^*) with respect to the most recent traffic event ($\hat{p}$) on a tree. If the time difference between p^* and $\hat{p}$ is greater than τ, 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.

B. 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 in the following cases.

- The parent-child relation between two traffic events.
- The sibling relation between two traffic events.
- The 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

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 at user events, which are parents of subroots $\{T_1, \ldots, T_m\}$, where subroot T_m is the most recent one; and a threshold τ.
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: **for** 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: **for** all children of node n **do**
20: enqueue the child nodes onto Q
21: **end for**
22: **end if**
23: **end for**
24: **end for**
25: **end if**
26: **return** False

features with machine learning techniques is subject to our future work.

Is_Child is given 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 τ 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.

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 τ 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. 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. The Is_Sibling check is a necessary complement to the Is_Child check as *Is_Sibling* helps identify late-arriving child nodes whose intervals of timestamps with respect to the parent are larger than the specified threshold, yet whose intervals with respect to the (older) sibling are still within the threshold.

Is_Subroot procedure is to identify the traffic events of the type subroot through analyzing the user events and the outgoing network requests. 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, 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. More details on user-event processing are given in Section V-A.

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

CR-Miner is capable of handling complex web scenarios such as redirection, automatic refreshing and AJAX calls. Automatic updates do not have explicit user actions that request for them (e.g., RSS feeds). A mitigation is to recognize the periodic automatic update traffic with pre-defined or automatically learned whitelisting rules. *HTTPS traffic* contains encrypted HTTPS headers which can be investigated by the known SSL proxying technique [15]. 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. Compared to the traditional approach of analyzing network packets independently in isolation, our semantic-based approach provides more structural and contextual information for anomaly detection on network activities.

IV. Security Analysis

In this section, we answer the question "Can CR-Miner be tricked?". CR-Miner is consisted of *data collection* and *data analysis* phases. Once the data is collected, the dependence analysis may be conducted off on a separate trusted machine.

Thus, the main security threats come during the data collection phase. Our threat model (in Section II) 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, which we discuss in the next two sections respectively.

A. 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. 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.

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.

B. Integrity of System Data

Because user input events are used for deriving traffic dependencies in particular for identifying subroot traffic, the integrity of user events obtained is important. Our threat model considers user-space application-level stealthy malware. Therefore, the kernel-level system data – including the process ID, keyboard and mouse events – is trusted (provided the data is collected properly).

In practice, advanced keystroke-integrity solutions such as the provenance verification in [9], [10], [21] may be incorporated in CR-Miner to further improve system-data assurance, which can be a useful fail-safe mechanism to guard against potential operational errors.

C. 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 semantic 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 V-D. Similar piggybacking attacks are discussed by Xu *et al* in [24] in the context of detecting drive-by-download attacks.

V. Implementation and Evaluation

We describe the prototype implementation of CR-Miner in Section V-A. Several experiments are performed to extensively evaluate the accuracy of CR-Miner.

A. Prototype Implementation

We develop a CR-Miner prototype in Windows 7 operating system. The detailed architecture of our prototype is shown in Figure 2. 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 our CR-Miner (the darker parts) between the applications and the kernel level.

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 III. 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. 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 the quadruple $\langle$source IP, source port, destination IP, destination port$\rangle$. 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` method in IPHelperAPI.dll and associate the TCP connections with the corresponding process IDs. However, `GetExtendedTcpTable`, which is similar to `netstat` command, does not provide the real-time information about the process and its TCP table. To mitigate

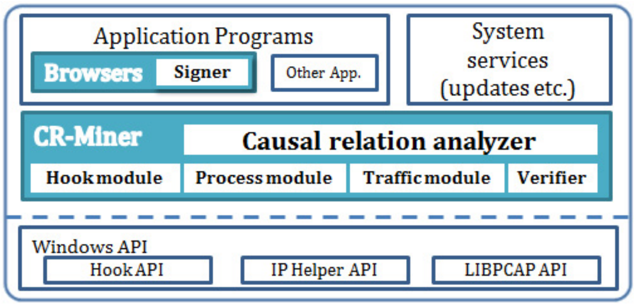

Fig. 2. Architecture of CR-Miner prototype.

this problem, we set up a timer to periodically retrieve the list of TCP connections and process information. Based on analyzing browser behaviors together with our experimental observations, we find that the request packets that are captured by `SharpPcap` may contain null attributes (e.g., PID or referrer). Therefore, we adjust the conditions of Is_Sibling in Section III-B so that the null attributes can be updated when needed. For two outbound HTTP request p_a and p_b, suppose p_a comes earlier than p_b (the time interval is less than the threshold τ) and p_b has null attribute(s). For the attributes of PID, host and referrer, if p_a and p_b have two identical attributes and p_b has the missing attribute on the third one, then we regard p_a and p_b has sibling relation and p_b's missing field can be revised by p_a's value. This update is important, as the fixed one may be used to infer the dependency of future requests.

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. Furthermore, we obtain the process ID of the current foreground window by using `GetWindowThreadProcessId()` method, 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.

We 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, including the navigation and tab events. The information gathered by the Tlogger is complementary to the data recorded by the kernel hook module.

B. Accuracy of Dependency Inference

We conducted a user study with 20 participants to collect samples of HTTP traces. All participants were graduate students in a university. Each participant was asked to actively browse the Internet for 30 minutes on a laptop pre-install with CR-Miner. Since the outbound HTTP traffic and user inputs can be collected, we asked the users not to reveal any sensitive personal data such as passwords. The means and standard deviations of the number of events that we collected are shown in Table I. We notice that the number of traffic-generating user events is far less than the total user events observed. Because our *Is_Subroot* analysis is based on traffic-generating user events, it is quite efficient. Since SQL Server database is employed in all experiments, we measure the size of the database BAK file for each user. The BAK file contains only data pages so that it is counted as the real disk space allocated for storing the records.

TABLE I
MEAN AND STANDARD DEVIATIONS (SD) OF STATISTICS OF DATA COLLECTED IN THE USER STUDY.

	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

We analyze the dependence in the data collected by our CR-Miner framework. A *legitimate* HTTP GET request needs to belong to a valid tree in the traffic-dependency graph rooted by a user event. 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. We have manually inspected the dependencies found to ensure they are correct. The distribution of hit rates in our user studies is shown in the Table II. 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%.

TABLE II
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 analyze the hit rate by calculating the percentages of how many requests are inferred by which one of the three subroutines *Is_Subroot*, *Is_Child*, and *Is_Sibling*. The result in Table III shows that most requests (about 87.4%) are inferred by *Is_Child* and a few can be inferred by *Is_Sibling*. The whitelist in our experiment 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.

We further investigate the outbound requests with missing dependencies, which account for 0.4% of the total traffic as shown in Table III. The major 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. Some of the vagabonds are legitimate requests, i.e., false positives. Whereas, others are requests to known malicious websites, i.e., true positives, e.g., atwola.com, adadvisor.net, and pixel.quantserve.com.

TABLE III
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%

To experimentally confirm the well-formed property of HTTP requests, we further evaluate the hit rates for top 20 websites from Alexa.com. We find that 0.28% of requests are missing their dependencies, suggesting that CR-Miner works well with legitimate websites.

C. Time Efficiency Comparison

In order to compare the run-time efficiency of our BFS based dependency inference (DI) algorithm, we implement a 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.

Through the observation of 20 user cases, the BFS based algorithm takes 2.7 second to process a thousand records, while the sequential scanning algorithm processes a thousand records in 7.6 seconds. Therefore, the result shows that the sequential algorithm takes about 2.8 times as long as the BFS based one in terms of the running time. 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.

D. Accuracy Comparison With Temporal-Only Analysis

We compare CR-Miner with a temporal-only dependence analysis algorithm that infers dependencies based solely on the intervals and PIDs of requests. Such a temporal-only dependence analysis is used in BINDER [7]. We filter non-subroot requests by an interval threshold τ, instead of using *Is_Child* and *Is_Sibling*. The ratio of the number of legitimate requests identified in the temporal-only algorithm to those of our CR-Miner is defined as *precision*.

A participant frequently visited Google maps and Google search pages; therefore the case yields a low *precision* (24.7%), which shows that the temporal-only analysis is limited in predicting traffic dependencies. Due to heavily using AJAX technique, the actual delay between a request and its subroot is longer than the threshold. Therefore, the temporal-only prediction suffers.

With considering the semantic information in network requests, we have the pre-set threshold in CR-Miner as 15 seconds and achieve the average hit rate as high as 99.6%. Hence, our algorithm substantially outperforms non-semantic analysis in terms of the accuracy of identifying traffic dependencies.

E. Prediction Accuracy Under Multi-user Data

Cross-user validation is to measure the accuracy of our analysis under the noisy traffic. We arbitrarily introduce noise 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 randomly choosing independent ones and merging them ($C_5^2 = 10$). Rather than shuffling two user studies and combining them, we merge the two user studies without losing their internal orders. The merging algorithm is along the same lines as *Merge Sort*, but we add a component to merge two list without breaking their orderings. We then run the BFS based DI algorithm on the mixed data. To evaluate the algorithm, 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.

In order to check the consistency of subroot in the user study and cross-user test, we have recursive functions to locate the subroot for each packet, as we find the root of arbitrary node in a tree. We run ten cross-user tests which are composed of five independent user studies by BFS based DI algorithms. The average error rate is 0.8%. Therefore, the cross-user validation shows a high prediction accuracy under multiuser dataset. These results indicate our DI algorithm is noise-resistant and robust to complex cases.

F. Real-World Spyware Detection

We use our CR-Miner to detect two pieces of real-world malwares. The malware 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 installed without user's full knowledge and consent once they are downloaded. Therefore, Infostealer.Maximus is a kind of case I malware as defined in Section II. Trojan.Brojack.A not only modifies the registry entries, captures all links that are browsed by the user, but also sends out outbound traffic to a host (watson.microsoft.com). According to the observation of the HTTP GET request, we can infer that the trojan tends to get a specific version of a piece of malware. Since it runs with an independent PID and sends out outbound HTTP requests to a malicious host, Trojan.Brojack.A belongs to case II malware. A common feature is that neither pieces of malware carries appropriate referrers. CR-Miner successfully flags the malware traffic as vagabond, because these requests are not rooted by any user events 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

TABLE IV
SPYWARE TRAFFIC (WITH WITH ID 23) IS IDENTIFIED AS IT HAS -1 IN ITS PARENT ID FIELD, INDICATING NO DEPENDENCE WITH EXISTING 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**
...						

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 existing spyware such as FormSpy, FireSpyFox, and FFsniff.

An example of traffic-dependency graph, which is created by the CR-Miner framework, is shown in Table IV.

- 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 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, namely not being associated with a valid tree in the TDG.

This detection is possible, because *i)* the spyware activity is not qualified to be a subroot as its domain name does not match any valid user event, and *ii)* its domain name is not referred by any proceeding requests. Therefore, by finding dependencies in traffic, we show that our solution renders spyware and keyloggers useless, as their outbound communication channels are blocked.

VI. RELATED WORK

Not-A-Bot (NAB) is a system for authenticating traffic-generating user inputs such as mouse clicks on hyperlinks [9]. It can be used for defeating attacks such as click fraud. However, it does not analyze the relationship among network packets for anomaly detection as in CR-Miner. As explained in Section IV, CR-Miner can use NAB and similar techniques, such as [23], to ensure the integrity of user inputs collected.

Shirley and Evans [18] proposed to collect the complete history of user and program actions to improve the precision and expressiveness of access control policies, thus, detect the malware. To achieve the similar goal on the network security field, our work focuses on the user intention-based outbound requests, instead of the program actions.

BINDER [7] is an elegant host-based solution that detects break-ins on personal computers by correlating the timestamps of user events and traffic, including control traffic at the lower level of the network stack (such as TCP control packets). BINDER analyzes the outgoing traffic and identifies their temporal relations. The authors focus on the delay between user input and data arrival event in the detection algorithm, but they do not concern the semantics among traffic packets. Our CR-Miner aims to provide application-specific anomaly detection as opposed to kernel-level extrusion detection in [7]. Thus, our rules and system architecture enforce the fine-grained traffic dependence characteristics regarding the user interactions with an application.

Shieh and Gligor [17] presented a model that tracks both data and privilege flows within secure systems to detect context-dependent intrusions caused by operational security problems. The model may not be suitable for detecting new, unanticipated intrusion patterns. Enforcing Dependencies is also well known in the field of policy management. Kagal *et al.* [13] proposed to apply dependency tracking to provide explanations for policy management, understand how the results were obtained, and therefore improve the trust in the policy decision and enforcement. In our work, we adopt the idea of enforcing policies within the application to infer the dependency among network requests.

Patnaik *et al.* [14] proposed to use Dynamic Bayesian Network (DBN) for dependency mining. Their work shows that frequent episodes help identify nodes with high mutual information relationships and that such relationships can be captured by a DBN. We plan to investigate the feasibility of such techniques for automatic learning and enforcement of user intention-based traffic dependence in the future.

WebTap, developed by Borders and Prakash [2], is a tool to anomaly 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 improved the detection accuracy by pruning repetitive information (e.g., header fields) in [3]. Our user intention-based traffic dependence analysis is different – we do not require any knowledge of behavior patterns of any user groups. Our rules are derived from the properties of applications.

Srivastava and Giffin [20] presented a technique for discovering the origin of parasitic malware on a host through sophisticated OS-level diagnostic. Their solution can be used to pinpoint the origin of the malware, after CR-Miner has identified its stealthy traffic.

VII. 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 semantic 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.

For future work, we will formalize the traffic-dependency model based on the finite-state automaton and its constraints. Such a formal dependency model allows one to derive fine-grained requirements of legitimate event sequences, and is a specialized Schneider's execution monitor [16].

References

[1] X. An, D. N. Jutla, and N. Cercone. Privacy intrusion detection using dynamic bayesian networks. In *International Conference on Electronic Commerce (ICEC)*, pages 208–215, 2006.

[2] K. Borders and A. Prakash. Web Tap: Detecting covert web traffic. In *Proceedings of the 11th ACM Conference on Computer and Communication Security*, pages 110–120, 2004.

[3] K. Borders and A. Prakash. Quantifying information leaks in outbound web traffic. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2009.

[4] E. Bursztein and J. Goubault-Larrecq. A logical framework for evaluating network resilience against faults and attacks. In *Computer and Network Security, 12th Asian Computing Science Conference (ASIAN)*, pages 212–227, 2007.

[5] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41:15:1–15:58, July 2009.

[6] M. Christodorescu, S. Jha, and C. Kruegel. Mining specifications of malicious behavior. In G. Shroff, P. Jalote, and S. K. Rajamani, editors, *ISEC*, pages 5–14. ACM, 2008.

[7] W. Cui, Y. H. Katz, and W. tian Tan. Binder: An extrusionbased break-in detector for personal computers. In *In Proceedings: USENIX Annual Technical Conference*, page 4, 2005.

[8] D. E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, SE-13(2), February 1987.

[9] R. Gummadi, H. Balakrishnan, P. Maniatis, and S. Ratnasamy. 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)*, 2009.

[10] R. Hasan, R. Sion, and M. Winslett. Preventing history forgery with secure provenance. *TOS*, 5(4), 2009.

[11] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 296–304, May 1990.

[12] J. Jung, A. Sheth, B. Greenstein, D. Wetherall, G. Maganis, and T. Kohno. Privacy oracle: a system for finding application leaks with black box differential testing. In *Proceedings of Computer and Communications Security (CCS)*, 2008.

[13] L. Kagal, C. Hanson, D. J. Weitzner, and D. J. Weitzner. Using dependency tracking to provide explanations for policy management. In *POLICY*, pages 54–61, 2008.

[14] D. Patnaik, S. Laxman, and N. Ramakrishnan. Discovering excitatory relationships using dynamic bayesian networks. *Knowledge and Information Systems*, pages 1–31, 2010. 10.1007/s10115-010-0344-6.

[15] An SSL proxying technique to sniff HTTPs packets. http://www.charlesproxy.com/documentation/proxying/ssl-proxying/.

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

[17] 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.

[18] J. Shirley and D. Evans. The user is not the enemy: Fighting malware by tracking user intentions. In *NSPW '08: Proceedings of the 2008 workshop on New security paradigms*, pages 33–45, September 2008.

[19] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, T. Grance, L. T. Heberlein, C.-L. Ho, K. N. Levitt, B. Mukherjee, D. L. Mansur, K. L. Pon, and S. E. Smaha. A system for distributed intrusion detection. *COMPCOM Spring '91 Digest of Papers*, pages 170–176, February/March 1991.

[20] A. Srivastava and J. T. Giffin. Automatic discovery of parasitic malware. In *Recent Advances in Intrusion Detection (RAID)*, pages 97–117, 2010.

[21] D. Stefan, C. Wu, D. Yao, and G. Xu. 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)*, June 2010.

[22] H. S. Teng, K. Chen, and S. C.-Y. Lu. Adaptive real-time anomaly detection using inductively generated sequential patterns. *Security and Privacy, IEEE Symposium on*, 0:278, 1990.

[23] K. Xu, H. Xiong, C. Wu, D. Stefan, and D. Yao. Data-provenance verification for secure hosts. *IEEE Transactions on Dependable and Secure Computing*, (PrePrints), 2011.

[24] K. Xu, D. Yao, Q. Ma, and A. Crowell. Detecting infection onset with behavior-based policies. In *Proceedings of the Fifth International Conference on Network and System Security (NSS)*, September 2011.