Detection of Stealthy Malware Activities with Traffic Causality and Scalable Triggering Relation Discovery

Hao Zhang, Danfeng (Daphne) Yao, and Naren Ramakrishnan
Department of Computer Science, Virginia Tech
Blacksburg, VA, USA
{haozhang, danfeng, naren}@cs.vt.edu

ABSTRACT

Studies show that a significant portion of networked computers are infected with stealthy malware. Infection allows remote attackers to control, utilize, or spy on victim machines. Conventional signature-scan or counting-based techniques are limited, as they are unable to stop new zero-day exploits. We describe a traffic analysis method that can effectively detect malware activities on a host. Our new approach efficiently discovers the underlying triggering relations of a massive amount of network events. We use these triggering relations to reason the occurrences of network events and to pinpoint stealthy malware activities. We define a new problem of triggering relation discovery of network events. Our solution is based on domain-knowledge guided advanced learning algorithms. Our extensive experimental evaluation involving 6+ GB traffic of various types shows promising results on the accuracy of our triggering relation discovery.

Keywords

Network Security, Stealthy Malware, Anomaly Detection

1. INTRODUCTION

Stealthy malicious software poses serious threats to the security of networked computers and data. A recent study showed that a significant portion ($> 25\%$) of computers worldwide are infected with malware conducting clandestine activities [33]. Malware may spy on the victim user (e.g., stealing passwords such as in the newly discovered Pony botnet [34], tracking the user's activities, data exfiltration), abuse the computer for conducting bot activities (e.g., command-and-control, launching attacks from it).

Determining whether or not networked hosts are infected with stealthy malware is technically challenging. Virtually all malware activities require sending outbound network traffic from the infected machine. However, because of the low traffic volume of stealthy malware, frequency-based statistical anomaly detection is not effective. HTTP and DNS have been widely observed as the protocols for malware and botnet communications, as they are rarely blocked by firewalls.

ASIA CCS'14, June 4–6, 2014, Kyoto, Japan.
Copyright 2014 ACM 978-1-4503-2800-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2590296.2590309.

The initial infection vector of most malware is usually through exploiting vulnerabilities of common networked software, e.g., heap overflow vulnerability in web browser or its plug-ins [13]. Once the infection is successful (e.g., zero-day exploits), network requests from advanced malware may not exhibit distinct communication patterns. Because of this lack of signatures, pattern-based scanning is ineffective.

Compared to independently inspecting individual network requests, a more effective network security approach is to discover characteristic behavioral patterns in network event attributes, e.g., [14, 18, 24]. For example, BINDER [14] detects anomalous network activities on personal computers through analyzing the correlation in traffic events by the temporal and process information. BotMiner [18] showed the effectiveness of correlation analysis across multiple hosts of a network in detecting similarly infected bots. King *et al.* constructed directed graphs from logs to show network connections for dissecting attack sequences [24]. However, none of these above solutions is designed for detecting general stealthy malware activities. Thus, they cannot be directly applied to solve the problem.

Stealthy malware can cause data exfiltration, spy on victim machines, and botnet command and control traffic on a host. We refer *stealthy malware activities* as the network traffic sending out from a host without user's intention.

We develop a new traffic analysis tool that detects network activities of stealthy malware through reasoning the causality among network events. Higher-level information such as the underlying relations or semantics of events is useful for human experts' cognition, reasoning, and decision making in cyber security [17]. Thus, analyzing relations among network events may provide important insights for identifying network anomalies.

There have not been systematic studies on network-request-level causal analysis for malware detection. Existing dependency analysis work (e.g., [9,31,43]) is on network service level and is not designed for malware detection. For example, Orion [9] and NSDMiner [31] addressed the problem of network application/service dependency for network stability and automatic manageability. Rippler [43] is proposed to actively perturb or delay traffic to understand the dependencies between service and devices. We aim to achieve the request-level causality structure in network traffic. This finer granularity (request vs. flow) requires different relation semantics and more scalable analysis methods. The existing binary classification solutions designed for JavaScript analysis [29] and malware detection [11] cannot be directly applied to our triggering relation discovery problem.

Triggering relations of events provide contextual interpretations for the behaviors of systems and networks, illustrating why sequences of events occur and how they relate to each other. Because of the transitivity, the problem of discovering triggering relations among a set of events may be transformed into discovering the triggering relations of pairs of events, which is defined as the *pairwise* triggering relation.

In this work, we introduce the problem of triggering relation discovery in network traffic and describe its application in solving challenging network security problems, such as stealthy malware detection. We present a scalable learning-based technique to compute the triggering relations among network events. We use the discovered triggering relations to reason about the legitimacy of observed network activities. Our analysis method successfully detects several types of anomalies including spyware activities, and botnet command-and-control traffic, as well as compromised web servers and web server misconfiguration. The accuracy and scalability of our triggering relation discovery method are evaluated with 6+ GB real-world network traffic, including HTTP, DNS, and TCP traffic. Experimental results show that our method efficiently predicts pairwise triggering relations with high accuracy in all metrics. With our tool, we discover and report the detected malicious HTTP and DNS activities due to various tracking malware and malware-hosting servers.

Our analysis using machine learning is scalable, capable of rapidly processing a large amount of traffic. We extract around 10 pairwise features from each type of network requests, based on timestamp, process ID, destination IP, domain name, etc.[1] Using machine learning algorithms eliminates the need for manually deriving classification rules and thus simplifies the detection. In addition, it achieves very high classification accuracy.

Our work demonstrates that triggering relations among cyberspace events at all levels can provide structural evidences for system and network assurance. The causality provides the logical interpretation to the vast amount of otherwise structureless and contextless network events. Comparing to link prediction problem in social networks [5, 16, 28], our work on triggering relation discovery has two major differences: *i)* Conceptually, link prediction problems focus on finding the similarity among nodes, while our work is on discovering triggering relations among network-related events. These two problems have different setups and requirements. *ii)* Our method includes the TRG construction operation and root-trigger security analysis, which are unique and beyond the link prediction type of inference problem.

2. MODEL AND OVERVIEW

In this section, we define the problem of *triggering relation discovery* and present the security applications.

2.1 TRG Definitions and Properties

Triggering relationship between event e_i and event e_j describes the temporal relation and causal relation between them, specifically e_i precedes e_j and e_i is one of the reasons that directly or indirectly causes e_j to occur. The specific semantics of triggering relation depend on the type of events and environment. An event may be defined at any relevant type or granularity, including user actions (e.g., keyboard stroke, mouse click), machine behaviors (e.g., network request, function call, system call, file system access), and higher-level operations and missions (e.g., database access, obtaining Kerberos authorization, distributing video to select users).

Triggering relations of events may be represented in a directed graph – referred to by us as *triggering relation graph* (TRG), where each event is a node and a directed edge ($e_i \rightarrow e_j$) from e_i and to e_j represents the triggering relation. We also refer the triggering relation ($e_i \rightarrow e_j$) as the *parent-child* relation, where e_i is the parent trigger or parent and e_j is the child. One can construct the TRG by incrementally inserting new events to the current graph.

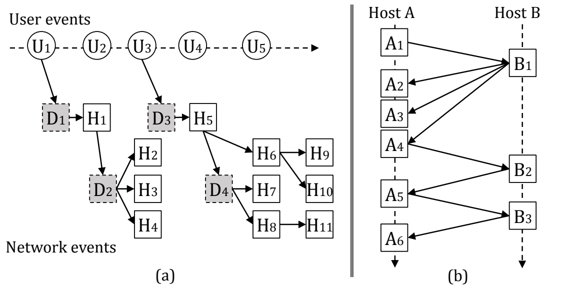

Figure 1: Schematic drawings of triggering relation graphs for outbound traffic from a host (a) and traffic between two hosts (b). In (a), the user events (e.g., U_3) such as entering a URL into the browser address bar are root triggers, which are followed by DNS queries (e.g., D_3) for translating the requested domain names. Then, one or multiple HTTP requests (e.g., H_5) are sent to the servers, and additional HTTP requests (e.g., H_6) may be triggered to fetch embedded objects. In (b), triggering relations in a TCP type of sessions are shown.

We illustrate two TRG examples in Figure 1. For specific types of network traffic, such a TRG may manifest unique topology and properties. For example, for outbound HTTP and DNS traffic from a host, the TRG forms a forest of trees, rooted by user inputs. The user input events are root triggers. Because of the temporal property of events, triggering relation graphs are free of cycles. This acyclic property differs the TRGs from social network graphs in link prediction problems. In a valid TRG, a node has at most one parent, thus, at most one root trigger. As confirmed by our experiments, a network triggering relation graph is usually sparse, i.e., the number of neighbors of a node compared to the total numbers of nodes is small. This sparsity is similar to what is observed in social network graphs [21].

The problem of triggering relation discovery is that given a set of events, to construct the complete triggering relation graph corresponding to the events. On the graph mode, the discovery problem is given a set of event nodes, to determine the existence of edges between pairs of the nodes and the directions of the edges. The *pairwise* triggering relation discovery is a simpler problem, which is to determine whether a triggering relation exists in two events. Given the pairwise triggering relations, we construct the complete the trigger-

[1]Examples of features are shown in Table 9 in the Appendix.

ing relation graph(s). We illustrate the TRG construction operation in Figure 2. A TRG provides a structural representation of triggering relations of observed events.

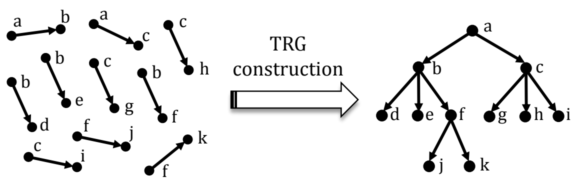

Figure 2: The triggering relation graph (TRG) on the right can be constructed from the pairwise triggering relations on the left.

Our definition of event-level causal relation relates to, but differs from the service dependency definition in existing service dependency research, such as network service dependency [31] and active delay injection [43]. Service dependency refers to that one service relies on another to function, e.g., web service depends on DNS name resolution. Our event-level causal relation refers to that one event triggers or causes the other event, e.g., the transmission of one network packet triggers the transmission of the other packet.

The TRG is defined differently from the parental dependency graph (PDG) in WebProphet [27] in terms of graph semantics and security applications. The PDG in WebProphet predicts the performance impact of webpage objects. Therefore it contains the timing information only, without capturing the causality among requests or objects. We rely on the semantic information (such as domain name, request string, etc.) of each request to build TRG, thus enabling the TRG to detect anomalous network events.

2.2 Security Applications of TRG

In our security model, network requests on TRG without valid root triggers are referred to as *vagabond* requests. They are anomalous events without legitimate causal relations, and likely due to stealthy malware activities.

The definition of root triggers may vary. For user-intention based triggering model, root triggers are user-input actions. The analysis pinpoints network activities that are not intended by users. Blocking these outbound malware network activities effectively isolates the malware, including

- websites collecting and reporting sensitive user data, affecting user privacy,
- spyware exfiltrating sensitive information through outbound network traffic from the monitored host,
- bots' command-and-control traffic, and attack activities (e.g., spam or DoS traffic) originated from the monitored host.

We describe a scenario for using our tool to detect stealthy outbound malware activities on a host. DNS tunneling has been abused by botnets for command and control communications [40]. These abnormal outbound DNS queries are automatically generated by malware on the host, typically with botnet-related payload. These surreptitious DNS activities are difficult to detect, because of their format resemblance to regular DNS queries. Our analysis tool reasons about the legitimacy of observed DNS traffic on a possibly infected host. Legitimate DNS queries are usually issued by an application (e.g., browser) upon receiving certain user inputs (e.g., entering a URL into the address bar). The application then issues additional DNS or other requests (e.g., HTTP, FTP). Botnet DNS queries lack of any matching user triggers. Our tool detects these vagabond events and reports them.

Data integrity We consider application-level malicious events, so the kernel-level system data (e.g., keyboard and mouse events) are assumed to be trustworthy. To prevent the forgery of user events, advanced keystroke and system integrity solutions such as [3, 19, 36, 41] can be incorporated in our work to further improve system-data assurance.

3. TRIGGERING RELATION DISCOVERY

The methods described in this section infer the triggering relation among network requests and construct triggering relation graphs. The automatic analysis pinpoints the occurrences of anomalous network requests from the voluminous traffic data, through reasoning triggering relations.

Given two network requests P and Q with P occurring before Q, one needs to decide whether P triggers Q, i.e., $P \to Q$. A straightforward approach for triggering relation discovery is the rule-based classification. One can define one or more rules summarizing the attribute properties of two parent-child requests, e.g., as shown in Example 3.1 below. Attribute names used in the example are described in Table 1.

EXAMPLE 3.1. *If $P.time \leq Q.time \wedge P.PID = Q.PID \wedge P.host = Q.referrer$, then P is the parent trigger of Q, i. e., $P \to Q$.*

However, the rule-based approach has several drawbacks that hinder its scalability and accuracy. It requires manual rule specification, which is time consuming. The rigid rule structures are not flexible enough to recognize complex traffic scenarios, resulting in low classification accuracy and false alarms.

Our approach utilizes probabilistic machine learning algorithms and achieves high scalability and detection accuracy. We introduce a scalable feature extraction method referred to as *Pairing*. This operation converts individual network events into event pairs with comparable pairwise attributes. We show how binary classification algorithms can be used for triggering relation discovery.

3.1 Overview of Our Approach

The main operations in our analysis are DATA COLLECTION, PAIRING, DATA LABELING, TRAINING, CLASSIFICATION, TRG CONSTRUCTION, and REPORT. The DATA LABELING, TRAINING and CLASSIFICATION operations are standard for machine learning based methods. The new operations are PAIRING and TRG CONSTRUCTION.

- DATA COLLECTION is to record and store the events to be analyzed. Each event e has one or more attributes $(A_1, \ldots, A_m)$ describing its properties.
- PAIRING is a new operation that we design for extracting pairwise comparison results (i.e., features) of events' attributes. Its inputs are two events $e = (A_1, \ldots, A_m)$ and $e' = (A'_1, \ldots, A'_m)$. The output is the event pair (e, e') with m pairwise attribute values $(B_1, \ldots, B_m)$, where a pairwise attribute $B_i (i \in [1, m])$ represents the comparison result of attributes A_i and A'_i. That is, $B_i = f_i(A_i, A'_i)$, where $f_i()$ is a comparison function for the type of the i-th attribute in the

ID	Time	PID	DestAddr	Request (Q)	Host	Referrer (R)	Q Type	R Type	ParentID
...									
4	22.723	2724	64.30.224.103:80	`/`	`www.cnet.com`	N/A	website	NULL	**0**
5	22.733	2724	198.82.164.40:80	`.../combined.js`	`i.i.com.com`	`www.cnet.com/`	javascript	website	**4**
6	22.973	2724	198.82.164.40:80	`.../matrix.css`	`i.i.com.com`	`www.cnet.com/`	css	website	**4**
...									
14	25.307	2724	198.82.164.40:80	`.../bgBody.gif`	`i.i.com.com`	`.../matrix.css`	multimedia	css	**6**
...									

Table 1: Original network events observed. Time, Q Type, R Type, and ParentID **stands for timestamp, request type, referrer type, and the ID of its parent event. The source IP of network events in this example is the same, while the source ports may differ (not shown).**

(ID1,ID2)	TimeDiff	PIDDiff	AddrDiff	RequestSim	HostSim	ReferrerSim	Q1	R2	Relation
(4,5)	0.00	1	1111000001	1	0.5	0	website	website	**1**
(4,6)	0.25	1	1111000001	1	0.5	0	website	website	**1**
(4,14)	2.584	1	1111000001	0.1667	0.5	0	website	css	**0**
(5,6)	0.24	1	1111111111	0.1356	1	1	javascript	website	**0**
(5,14)	2.574	1	1111111111	0.5593	1	0.5	javascript	css	**0**
(6,14)	2.334	1	1111111111	1	1	0.5	css	css	**1**

Table 2: Examples of pairwise attributes as outputs of the Pairing operation. Q1 **and** R2 **stand for the first event's request type and the second event's referrer type, respectively.**

events. The comparison function $f_i()$ (e.g., `isEqual`, `isGreaterThan`, `isWithinThreshold`, `isSubstring`, etc.) is chosen based on the type of attribute. The feature construction can be extended to comparing different traffic types. Pairing is performed on every two events that may have the parent-child triggering relation. Moreover, we demonstrate an efficient pairing algorithm to reduce the cost of pairing without compromising the analysis accuracy in Section 3.3. The pairwise features are used as inputs to the subsequent learning algorithms.

- Data Labeling is the operation that produces the correct triggering relations for the event pairs in a (small) training dataset. A binary label (1 or 0) indicates the existence or non-existence of any triggering relation in an event pair, e.g., $<(e, e'), 1>$ represents that event e triggers e'. Data labeling is based on pairwise attributes (e.g., $B_1, \ldots, B_m$) and may require manual efforts.
- Training is the operation that produces a machine learning model with labeled training data.
- Classification is the operation to use the trained machine learning model to predict triggering relations on new event pairs $\mathbb{P} = \{(e_i, e_j)\}$. E.g., the outputs of binary prediction results are in the form of $\{<(e_i, e_j), l_{ij}>\}$, where the binary classification result $l_{ij} \in \{0, 1\}$ represents whether event e_i triggers e_j in $\mathbb{P}$.
- TRG Construction is the operation to construct the complete triggering relation graph based on pairwise classification results. If event e_i triggers e_j in the event pairs $\mathbb{P}$, then e_i is the parent of e_j in the TRG.
- Report is the operation to apply security definitions to the triggering relation graph and report anomalous events. A user-intention based security definition for TRG analysis is presented in Section 3.5.

We describe details of our new Pairing operation in the next two sections. This feature extraction operation is unique in that the features enable the use of binary classification for pairwise directional relation discovery.

3.2 Pairing Operation

The Pairing operation extracts features of event pairs. Pairwise attributes $(B_1, \ldots, B_m)$ are computed by comparing the attribute values $(A_1, \ldots, A_m)$ and $(A'_1, \ldots, A'_m)$ of two individual events e and e'. A comparison function $f_i(A_i, A'_i)$ for $i \in [1, m]$ is selected based on the type of attributes A_i and A'_i. An event attribute is of the numeric, nominal, string/text, or composite type. After the pairwise feature extraction, binary classification algorithms is used for classification. The classification requires labeled pairs for training, where triggering relations among events (i.e., labels) are known. For test data, triggering relations are unknown and need to be predicted.

Without loss of generality, we illustrate a basic pairing procedure with outbound HTTP network events as an example. The approach can be generalized to other event types. In Table 1 we show examples of some HTTP events. The triggering relations, if known, are shown in the last column (under ParentID). The features in Table 1 are derived from the header of HTTP requests. As the header contains operating parameters of an application layer transaction, the casual/semantic relation can be measured by the features of the requests. These features are previously used to understand the behavioral model of web traffic [10], while our work further leverages them to build the trigger relations of network traffic for security purpose. The pairwise attributes are formed by aligning the same event features and comparing the relevant ones (e.g., the request type and the referrer type). Six event pairs are generated and their new pairwise attributes are shown in Table 2. For example, the HostSim, ReferrerSim and RequestSim give the similarity of two events in Host, Referrer and Request attributes, respectively, according to certain similarity measures. Each pair has a binary representation of the existence of a triggering relation (under Relation in Table 2). The *pairing* details are illustrated as follows.

- Numeric attributes (e.g., timestamps) are compared by computing their difference, e.g., the interval TimeDiff between the timestamps of two network events. That is, $B_i = A_i - A'_i$.

- A nominal attribute (e.g., file type, protocol type) categorizes the property of an event. Comparing nominal attributes usually involves string comparison, e.g., substring or equality tests.
- For the string type of attributes, we compute the similarity of the attribute values as the pairing attribute value. That is, $B_i = f_s(A_i, A'_i)$, where function f_s is a similarity measure, e.g., normalized edit distance. Take HTTP request as an example, we compute pair attributes HostSim and ReferrerSim by measuring the string similarities between two host fields and two referrer fields, respectively.
- A composite attribute is converted to primitive types, e.g., a destination address containing four octets for the IP address and an integer for the port. The comparison of two composite attribute values is made by comparing the sub-attribute values separately.

Given a list of n network events, the total number of event-pair candidates is bounded by $O(n^2)$. To reduce the computational cost, one may pair up the events that occur within a certain time frame τ, assuming that events occurring far apart are unlikely to have triggering relations. We design a more sophisticated pairing heuristic in the next section.

3.3 Efficient Pairing Algorithm

Our pairing algorithm pre-screens attributes to quickly eliminate unqualified pair candidates. The pseudocode of our algorithm is shown below. It takes a list of chronologically sorted network requests as the input and outputs a set of pairs of events.

Algorithm 1 Efficient Pairing Algorithm (EPA)

```
Input: a list of chronological sorted events, $\mathbb{L} = \{e_i\}$
Output: a set of event pairs, $\mathbb{P} = \{(e_i, e_j)\}$, $1 \le i < j$
 1: define a set $\mathbb{P}$ to store the compared pairs $\{(e_i, e_j)\}$
 2: define a dictionary $D = (d, \{e\})$, where $d$ is the domain
    of event and $\{e\}$ is a set of events whose domain is $d$.
 3: for each event $e_j \in \mathbb{L}$ do
 4:   $d \leftarrow$ the domain of $e_j$'s Host
 5:   if $e_j$'s Referrer is not NULL then
 6:     $dom \leftarrow$ the domain of $e_j$'s Referrer
 7:   else
 8:     $dom \leftarrow d$
 9:   end if
10:   if $dom$ in $D$'s keyset then
11:     for each event $e_i$ in $D[dom]$ do
12:       if pass the $Screening(e_i, e_j)$ then
13:         $\mathbb{P} \leftarrow \mathbb{P} \cup Pairing(e_i, e_j)$
14:       end if
15:     end for
16:     calculate the expire time and update $D[d]$
17:     add $e_j$ in $D[d]$
18:   else
19:     add new entry $(d, \{e_j\})$ in $D$
20:   end if
21: end for
22: return $\mathbb{P}$
```

Algorithm 1 uses a dictionary $D = \{(\text{key, value})\}$ to store the current network events. These events may be the parent triggers of future events. The key of the dictionary is the domain attribute of an event. The value is a set of requests, whose domain attribute is same as the key. Events with unmatched key values are filtered out (in *Screening* function of Algorithm 1), and not stored or paired, reducing both storage and computation overheads. As a result, a much longer time can be used to retire a domain, providing a more comprehensive coverage on pairs.

3.4 Feature Selection and Classification

Feature selection is to find an optimal set of representative features can greatly improve the effectiveness of machine learning classifiers. In our experiments, we use two different feature selection algorithms, namely *Information Gain* and *Gain Ratio*. Once a set of features is chosen, we train and classify the data using three common supervised machine-learning classifiers – Naive Bayes, a Bayesian network [20], and a support vector machine (SVM) [12].

Cost Sensitive Classifiers Because of the sparsity of triggering relations, we define customized cost matrices [15] to penalize missed relations during the training. The cost matrix can be defined to weigh the false positive (FP) and false negative (FN) differently. A false negative refers to the failure to discover a triggering relation. A false positive means finding triggering relation in a non-related pair.

Shown in Table 3, our cost matrix for classifying triggering relations is labeled by two categories: *with triggering relation* and *without triggering relation*. The values in the matrix are the weights for penalizing classification mistakes. We set positive values in the cells for FN and FP. The cost sensitive classification takes a cost matrix as an input. The trained model aims at minimizing the total penalty in imbalance data sets. For simplicity, we show the values and omit the labels of the cost matrix. For example, $\begin{bmatrix} 0, 1 \\ 1, 0 \end{bmatrix}$ is a cost matrix that has no bias on FPs and FNs; $\begin{bmatrix} 0, 1 \\ 10, 0 \end{bmatrix}$ penalizes the FNs 10 times more than FPs for a classifier. In Section 4, we thoroughly evaluate how cost matrices improve our analysis accuracy.

		Classified As	
		W/O TR	With TR
Ground Truth	W/O TR	TN: No penalty.	FP: penalty for finding triggering relations in non-related pairs.
	With TR	FN: penalty for failure to discover triggering relations.	TP: No penalty.

Table 3: Semantics of values in a cost matrix. TR stands for triggering relation.

3.5 TRG Construction and Root-Trigger Security

A list of pairwise triggering relations in network events can be used to construct the complete triggering relation graph (TRG). The resulting TRG then serves as a source for locating anomalous network activities. The security model, which defines legitimate and abnormal events, comes in many forms when used for analyzing TRGs.

Under the *root-trigger security* model, one determines the legitimacy of a network event e based on the legitimacy of e's root trigger, i.e., whether or not e has a legitimate root trigger. According to this definition, anomalous events are the events that do not have a valid root trigger. These events may be due to malware activities or host/server misconfiguration.

A specific root-trigger security definition is based on *user intention* [44], where a valid root trigger should be related to a user activity (e.g., a function call to retrieve user inputs, mouse clicks, or keyboard inputs). Other definitions for valid root triggers may be made according to the specific applications. We refer to the events that do not have any valid root triggers as the *vagabond* events.

In order to enforce the root-trigger security, the TRG CONSTRUCTION operation is used to calculate the discovered root triggers of all the events. To find the root of each event by traversal in TRG is equivalent to the transitive reduction of a directed graph. We design the root finding procedure (Algorithm 2) to return the root of an event, given all the pairwise triggering relations.

Algorithm 2 Root Finding Algorithm (RFA)

```
Input: an event $e_k$ and $\mathbb{P}^* = \{(e_i \to e_j)\}$.
Output: a set $\mathbb{R}$, where each in $\mathbb{R}$ is a root of $e_k$.
 1: define a set $\mathbb{R}$ to store the results
 2: define a queue $Q$ and enqueue $e_k$ onto $Q$
 3: while $Q \neq \emptyset$ do
 4:     event $n \leftarrow$ dequeue $Q$
 5:     set $\mathbb{T} \leftarrow$ find $n$'s parent(s) based on $\mathbb{P}^*$
 6:     for each event $e \in \mathbb{T}$ do
 7:         if $e$ is of type root then
 8:             $\mathbb{R} \leftarrow \mathbb{R} \cup \{e\}$
 9:         else if $e \notin Q$ then
10:             enqueue $e$ onto $Q$
11:         end if
12:     end for
13: end while
14: return $\mathbb{R}$
```

The inputs of Algorithm 2 are an event e_k and a set $\mathbb{P}^*$ containing all the pairwise triggering relations $\{(e_i \to e_j)\}$. The output is a set containing all the roots of e_k. In order to compute the transitive reduction of a directed graph, we use a queue Q to perform breadth-first traversal of TRG. In each iteration, we obtain the parent(s) $\mathbb{T}$ of a dequeued event n. For each event e in the set $\mathbb{T}$, the algorithm checks if it is a root-type event. If yes, then e is added to the return set $\mathbb{R}$. Otherwise (i.e., e is an intermediate node on the path from e_k's root to e_k), the algorithm enqueues e onto Q for further iteration. This analysis returns root triggers for the network requests. Network requests without valid root triggers are labeled as vagabond events. They are flagged and alerted to the administrator for further inspection.

We demonstrate the use of our method for detecting three types of common malware in Section 4, including

- spyware as a browser extension,
- data-exfiltrating malware as a stand-alone process,
- a DNS-based botnet command and control channel.

4. EVALUATION AND RESULTS

Our prototype implements all parts of the TRG discovery system. The questions we seek to answer through our experiments are: *i)* How accurate is the prediction for pairwise triggering relations? *ii)* How accurate is the prediction for root triggers of events? *iii)* Can the method detect outbound network activities caused by stealthy malware? *iv)* Can the method detect network connections to suspicious servers? *v)* Can the methods analyze different traffic types?

4.1 Experimental Overview

We have conducted extensive tests on our proposed traffic-causality-analysis solution and obtained positive results. In this section, we describe the setup of our experimental evaluation. Then, our evaluation results are presented in the next few sections.

4.1.1 Accuracy and Security Metrics

- The *pairwise accuracy rate* of classification is the percentage of pairwise triggering relations that are predicted correctly. The pairwise accuracy is with respect to the ground truth obtained through rule-based analysis and manual classification. An rule example is shown in Example 3.1.
- The conventional *precision and recall* measures [6] evaluate the classification accuracy of the positives (i.e., the existence of triggering relations). In the equations below, TP, FP, and FN stand for true positives, false positives, and false negatives, respectively.

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN}. \quad (1)$$

- The *root-trigger correctness rate* is computed based on the root of a node. It is the percentage of events whose roots in the constructed the triggering relation graph are correct with respect to the ground truth.

4.1.2 Datasets

Our evaluation is mainly focused on HTTP and DNS traffic, because they are very commonly used communication protocol both by legitimate users and attackers. Many botnets use HTTP or DNS as their communication protocol, because most firewalls allow them [40]. We collect and analyze outbound HTTP and DNS requests from hosts, aiming to detect suspicious activities by stealthy malware installed on the hosts. In addition, we also evaluate our algorithms with a much larger TCP dataset collected from a sever.

A summary of the experimental data is shown in Table 4. We define $\eta \in [0, 1]$ as the reduction percentage in Equation 2), where $EPA(n)$ is the number of event pairs after using the efficient pairing algorithm (in Section 3.3), and n is the total number of events.

$$\eta = 1 - \frac{EPA(n)}{n \times (n-1)/2} \quad (2)$$

- *Dataset I, HTTP.* We collected the user events and outbound HTTP traffic in a user study with 20 participants. Each participant was asked actively surf the web for 30 minutes on a computer equipped with our data collection program.
- *Dataset II, DNS and HTTP.* We used `tcpdump` to continuously collect the outbound DNS queries and HTTP requests on an active user's workstation for 19 days. We collected types `A/AAAA` DNS queries and the outbound HTTP requests that contain `GET`, `HEAD`, or `POST` information in their headers.
- *Dataset III, server TCP traffic.* We collected TCP packets on an active Linux server in a research lab. The inbound and outbound TCP packet headers were collected for 42 days using `tcpdump`.

4.1.3 Data Labels

Training data is labeled manually, with the use of simple rules such as in Example 3.1. The labeling process is

Data	Type	# of Events	η	# of Pairs	# of Feature	Size (MB)
I	Host-based HTTP	HTTP: 45,988; User: 899	94.7%	3,436,635	10	229.5
II	Host-based DNS and HTTP	DNS: 35,882; HTTP: 85,223	98.8%	953,916	9	55.1
III	TCP Traffic of a Server	TCP: 3,010,821	99.6%	119,372,631	9	6697.1

Table 4: An overview of datasets in the experiments. Number of events is the number of raw requests that have been collected. η is the reduction percentage after using our Efficient Pairing Algorithm. For Dataset I, the number of user events are also given in column 3.

Data	# of Pairs in Test Sets	Cost Matrix	Naive Bayes			Bayesian Network			SVM		
			Pairwise A.	Prec.	Recall	Pairwise A.	Prec.	Recall	Pairwise A.	Prec.	Recall
I	3,318,328	$\begin{bmatrix}0,1\\10,0\end{bmatrix}$	99.75%	0.954	0.996	99.75%	0.956	0.996	99.70%	0.958	0.997
II	693,903	$\begin{bmatrix}0,1\\1,0\end{bmatrix}$	99.82%	0.959	0.998	100.00%	1.000	1.000	100.00%	1.000	1.000
III	1,191,926,877	$\begin{bmatrix}0,1\\3,0\end{bmatrix}$	98.92%	0.995	0.986	99.72%	0.997	0.998	99.82%	0.998	0.999
Mean	–	–	99.50%	0.969	0.993	99.82%	0.984	0.998	99.84%	0.985	0.999

Table 5: Pairwise classification results of train-n-test for three datasets. The numbers are rounded before reporting. Pairwise A. and Prec. stands for pairwise classification accuracy and precision, respectively.

time consuming, and requires nontrivial human efforts. The labeling of DNS traffic requires the integral analysis of user-HTTP dependency and DNS-HTTP dependency, details of which are omitted. User events are labeled as root triggers, which are generated by leveraging `Tlogger` [2]. As a browser extension, `Tlogger` captures user inputs and tab events during the web browsing. By combining the data recorded on the kernel level, we generate the root-triggers used in TRG Construction for Dataset I and II.

4.1.4 Classification Setup

Three common classification techniques are compared: naive Bayes classifier, a Bayesian network, and a support vector machine (SVM).[2] Due to the sparsity of triggering relations in network traffic, we define a cost matrix that penalizes classifying false negatives more than classifying the false positives. CLASSIFICATION and TRG CONSTRUCTION operations are implemented in Java using the Weka library. We perform both 10-fold cross validation and train-n-test types of evaluation. The two evaluation methodologies yield similar classification results. We report the train-n-test results, unless otherwise specified.

4.2 Causality Analysis of Dataset I

Based on our two feature selection algorithms, 10 features out of a total of 12 are chosen for HTTP data. Selected features include three similarity indexes (RequestSim, ReferrerSim, HostSim), two nominal values to identify the file type (RequestType, ReferrerType), the nominal values to compare between particular attributes (PIDDiff, AddrDiff, TypeMatch, IndexOfSameRequest), and time difference (TimeDiff).

4.2.1 Accuracy of Pairwise Triggering Relations

The results in Table 5 show very good prediction accuracy for pairwise triggering relations. All classifiers give high precision and recall values, as well as the pairwise classification accuracy. These results indicate the effectiveness of our binary classification approach.

We vary the cost matrix used during the training and compute the pairwise accuracy results of the three classifiers for Dataset I. The results are shown in Figure 3 (a). The pairwise accuracy is consistently high for naive Bayes classifier with all cost matrices. Bayesian Network and SVM respond differently to the changes of penalty values in cost matrices. In Table 5, we report the accuracy results under the cost matrix of $\begin{bmatrix}0,1\\10,0\end{bmatrix}$. This matrix gives 10 units of penalty to a false negative and 1 unit of penalty to a false positive for the pairwise classification.

[2]SVM has a polynomial kernel function with a degree of 2.

4.2.2 Correctness of Root Triggers

The purpose of this analysis is to identify reasons for wrong predictions of triggering relations. Running the root finding algorithm (in Section 3.5) on the pairwise triggering relations, we identify the root triggers of all events and compare them to the ground truth values.

Figure 3 (b) shows the relationship between the cost matrix values and the accuracy of root-trigger analysis. The naive Bayes and Bayesian network yield nearly 100% accuracy of finding the root-triggers, both of which are not very sensitive to the cost matrices. In contrast, the accuracy of SVM increases significantly with increased false negative penalty in the cost matrix. In Table 6, we summarize the results of root trigger correctness for Dataset I. Our prediction of events' root triggers is accurate. It has a very small error rate, as low as 0.06%. These errors in finding root triggers generate false alerts. Wrong root triggers are mostly because of missing attributes in the original data or late-arriving requests. We further analyze false alerts later.

	Naive Bayes	Bayesian Network	SVM
Cost Matrix	$\begin{bmatrix}0,1\\10,0\end{bmatrix}$	$\begin{bmatrix}0,1\\10,0\end{bmatrix}$	$\begin{bmatrix}0,1\\100,0\end{bmatrix}$
Correct (case a-c)	99.94%	99.94%	99.37%
Wrong (case d-f)	0.00%	0.00%	0.28%
Wrong (case g)	0.06%	0.06%	0.35%

Table 6: Correctness of root triggers in Dataset I. Cases (a-g) refer to the various predicted root-trigger outcomes in Figure 4 in the Appendix.

4.3 Abnormal Traffic in Datasets I

4.3.1 Malicious Browser Extension

We wrote a proof-of-concept malicious Firefox extension, which is a piece of password-stealing spyware. The mal-

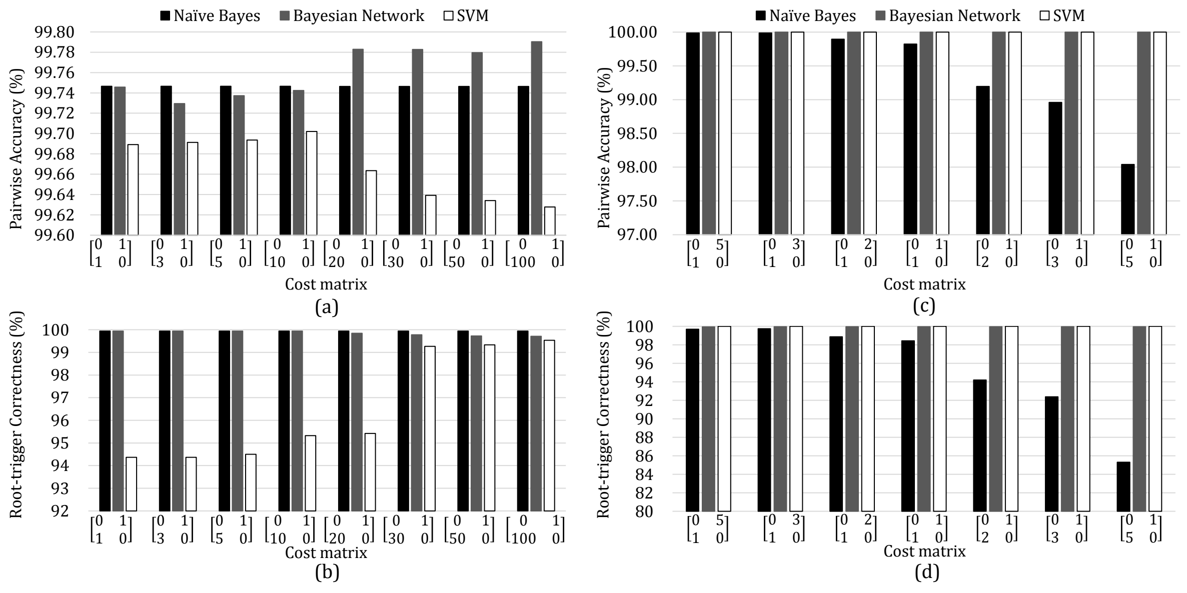

Figure 3: Accuracy and correctness results under various cost matrix conditions for Dataset I (pairwise classification accuracy in (a) and root-trigger correctness in (b)) and Dataset II (pairwise classification accuracy in (c) and root-trigger correctness in (d)).

ware sends the username and password when a user clicks on the `Submit` button in the browser. This spyware is similar to the existing spyware such as `FormSpy` and `FFsniff`. A victim user clicks the `Submit` to log on to various email services and Internet forums. The spyware requests, which contain the username and password in the `HTTP request` (`/query?id=user_id&ps=password`), are sent to its destination host. With our causality analysis tool, all malicious HTTP requests are detected by all three classifiers, without triggering any FPs and FNs.

4.3.2 Data Exfiltrating Malware

We write another proof-of-concept data-exfiltrating malware. This malware runs as a stand-alone process, similar to Pony bot. It sends out the `HTTP GET/POST requests` with system information to remote servers. The malware is programmed to transmit its payload right after the occurrence of a user event on the host, attempting to hide its communication among legitimate outbound traffic. The malicious communication may be a single request or a series of HTTP requests. Our method successfully detects the network activities of the malware in that the outbound malicious requests do not have valid triggering relations, i.e., the requests lack of any user event as the root-trigger.

4.3.3 Detection of Malicious Traffic in Dataset I

As defined in Section 3.5, vagabond events are those that do not have any valid user events as their root triggers. There are total 1.2% vagabond HTTP requests in Dataset I. Some of them are malicious traffic to known blacklisted websites. Our analysis finds in Dataset I that among these vagabond events, there are 169 suspicious requests sent to 36 distinct domains. Manual inspection reveals that these requests are to tracking sites, malware-hosting or blacklisted sites, and aggressive adware. They are partly due to users visiting compromised web sites. For example, some requests track the user's cookies and send back to remote hosts with known blacklisted sites (e.g., `2o7.net`, `imrworldwide.com`, `mediaplex.com`). We analyze the geographic locations of the malicious servers based on their IP addresses. All of them locate in the US, except one IP located in Netherlands. Some of the vagabond requests are false alerts (described in Section 4.3.4).

4.3.4 False Alerts

In our model, false alerts refer to the network requests that are vagabond requests (i.e., requests without proper triggers), but are legitimate (benign). False alerts in Dataset I are due to four main reasons:

- Automatic and periodic system and application updates that occur without user triggers. In Dataset I there are 157 update requests that are sent to 13 well-known legitimate domains. Whitelisting can be used to eliminate these alerts.
- Missing or incomplete attributes in the original data due to server configuration, e.g., redirection without properly setting the referrer field. There are 244 misconfigured requests that are sent to 38 different domains, usually image/video hosting websites.
- Unconventional attribute values, e.g., requests to `googlesyndication.com` (for Google Map) usually have long referrers that our prototype does not expect.
- Requests sent out much later than their parent request trigger, e.g., requests for favorites or bookmark icons.

Reducing false alerts can be achieved through more sophisticated inference methods under incomplete information, which will be investigated in our future work.

4.4 Causality Analysis of Datasets II

For dataset II, the goal of the experiment is to find the triggering relation in traffic with mixed types, such as DNS and HTTP requests. Features used for classification are given in Table 9 in the Appendix.

4.4.1 Pairwise Classification Accuracy

The pairwise classification results on dataset II are presented in Table 5. All three methods give high pairwise classification accuracy, confirming our method's ability of discovering triggering relations in mixed traffic types. Bayesian network and SVM yield better results than naive Bayes classifier, indicating that there are dependencies among attributes.

The pairwise classification accuracy under various cost matrices is shown in Figure 3 (c). Bayesian network and SVM consistently give high classification accuracy. In contrast, the performance of naive Bayes classifier decreases, as the cost matrix penalizes FNs more than FPs. We highlight the pairwise classification accuracy results under the cost matrix $\begin{bmatrix}0,1\\1,0\end{bmatrix}$ in Table 5.

4.4.2 Correctness of Root Triggers

We analyze the root-trigger accuracy for Dataset II, and show the results in Figure 3 (d). The root-trigger accuracy is high when using all three classifiers, with Bayesian network and SVM outperform the naive Bayes. We highlight the root-trigger accuracy results under the cost matrix of $\begin{bmatrix}0,1\\1,0\end{bmatrix}$ in Table 7.

	Naive Bayes	Bayesian Network	SVM
Cost Matrix	$\begin{bmatrix}0,1\\1,0\end{bmatrix}$	$\begin{bmatrix}0,1\\1,0\end{bmatrix}$	$\begin{bmatrix}0,1\\1,0\end{bmatrix}$
Correct (case a-c)	98.44%	100.00%	100.00%
Wrong (case d-f)	1.37%	0.00%	0.00%
Wrong (case g)	0.19%	0.00%	0.00%

Table 7: Root-trigger results on Dataset II. Cases (a-g) refer to the various predicted root-trigger outcomes in Figure 4 in the Appendix.

4.5 DNS Bot Detection

Botnet command and control channel using DNS tunneling [1] is extremely stealthy and difficult to detect [40]. We write a proof-of-concept bot that communicates with its bot master by tunneling command and control messages in DNS traffic. The bot generates carefully crafted outbound DNS queries whose payload contains encoded data e.g., `NBSWY3DPFQQHO33SNRSA000.domain.com`, `d1js21szq85hyn.cloudfront.net`. These bot queries are mixed with a 2-hour DNS-HTTP traffic dataset, which is then analyzed by our causality tool. Our evaluation confirms that our method successfully recognizes all the bot DNS queries as anomalies. These DNS queries do not have the valid user-event root triggers.

4.6 Causality Analysis of Datasets III

For Dataset III, the goal of the experiment is to find the triggering relation between inbound and outbound TCP packets by using our machine learning method. The accuracy results of pairwise triggering relation are in Table 5. All three classifications yield high values for the pairwise classification accuracy, with Bayesian network and SVM outperforming naive Bayes classifier. The features used for classification are shown in Table 10 in the Appendix.

4.7 Precision and Recall

Our methods result in high precision and recall for all data sets. In addition, the methods produce high pairwise classification accuracy and root-trigger correctness. Of particular significance are Bayesian Network and SVM, which yield the precision and recall of 1.0, a 100% pairwise classification accuracy, and a 100% root-trigger correctness for Dataset II.

Precision values are slightly lower than recall values in general, indicating more false positives than false negatives in the classification results. (False positive means finding triggering relations in non-related pairs. False negative is the failure to discover triggering relations.) The reason for slightly lower precision values is partly due to the customized penalty weights in the cost matrix.

4.8 Performance

Runtime results are obtained on a machine with Intel Duo Processor E8400, 3GB RAM and 250GB HDD. For each data set, we report the runtime of pairing, training, classification, and find-root operations. The means reported in Table 8 are averaged from five runs. Standard deviations are negligible and not shown.

Data	Pair	Train			Classify			Find
		NB	BN	SVM	NB	BN	SVM	Root
I	1848	0.5	1.2	79.9	22.7	16.8	14.2	1.7
II	622	0.8	2.1	13.4	4.0	2.2	2.1	0.6
III	14686	2.6	7.9	546	431	411	446	–

Table 8: Averaged performance (in seconds) of Pairing, Train, Classification, and Find-root operations. NB and BN stand for Naive Bayes and Bayesian Network classifiers, respectively. Pairing time includes feature extraction.

According to Table 8, the train, classification, and root-finding operations are fast. The Pairing operation is the most time-consuming task in our method. For example, it can take as long as 4 hours to generate the pairs from 3 million TCP messages (42 days of a server's TCP data). Our experiments have determined that on a single day, at most 200MB of pairwise data can be generated from a server's TCP packet headers. As for the processing time, generating the daily pairwise data takes only 6 minutes on average, indicating that our method is efficient enough for practical use.

4.9 Summary

We summarize our experimental findings below.

- Bayesian network gives the best analysis accuracy for all datasets. The naive Bayes classifier gives the lowest accuracy, indicating the existence of dependencies in pairwise features. The accuracy can be improved by strategically defining the cost matrix.
- Precision and recall metrics are more sensitive to the quality of the classification results than the pairwise accuracy metric. The fundamental reason for this difference is the sparsity of the triggering relations, which results in different sizes of the denominators in these metrics.
- Our causality analysis successfully reveals all the outbound traffic to 36 malicious domains, i.e., with zero false negative rate. Our tool also detects the stealthy

network activities from our proof-of-concept browser spyware, DNS bot, and stand-alone data-exfiltrating malware.

- *Limitations* In our optimized prototype, PAIRING operation (for extracting pairwise features) has high computational overhead. This overhead is due to the quadratic complexity in pairing. Heuristics for improving the pairing efficiency may result in decreased analysis accuracy. We will investigate this tradeoff in our future work.
  Our current feature extraction method does not handle well HTTP requests involving incomplete or unconventional attributes. The failure of recognizing the causality in these requests results in false alerts. Advanced inference techniques are required to improve this recognition.

5. RELATED WORK

The classification and discovery of application or service dependencies for management and reliability purposes have been recently reported [7,9,22,23,31,43]. These existing service dependency analysis solutions differ from our triggering relation discovery work in two aspects.

- The semantics of relations to be discovered are different, as the dependency in those papers refers to the reliance on services provided by others, not the triggering relation.
- The granularity of analysis differs, requiring completely different techniques; our request-level triggering relations is more fine-grained than service- or application-level dependencies.

Machine learning approaches have been widely adopted in the security literature, since the work by Lee, Stolfo, and Mok [26]. The solutions described in [13, 29] use machine learning techniques to capture characteristics of Javascript code and identify malicious Javascript code. Xie *et al.* [39] proposed to use Bayesian network on justifying the important types of uncertainty in real time security analysis. However, their work is not designed for analyzing request-level network traffic. EXPOSURE [8] is designed to detect domains involved in malicious activities by conducting large-scale and passive DNS analysis at network level. Authors extracted 15 features of DNS traffic and used J48 classifier to find the malicious domains. EXPOSURE classifies the DNS request on an individual basis, while we use the machine learning tools on the pairwise relations. Besides, our method can be adopted to various types of network traffic.

Nguyen and Armitage surveyed on Internet traffic classification using machine learning methods in [32]. Williams *et al.* [38] did an empirical study on summarizing the features from payload-independent features. Their work classifies IP traffic flows. Besides computer traffic classification, learning-based security research includes database intrusion detection [35], identifying botnet traffic [30], and SMS/social network spam detection [37, 42]. Compared to these aforementioned learning-based security solutions, the uniqueness of our triggering relation discovery model and technique is the ability to automatically extract and recognize directional relations and structures. Our problem is beyond the conventional binary classification problem.

Our triggering relation discovery problem may bear superficial similar to the link prediction problem in the context of mining social network data [5, 16, 21, 28]. Liben-Nowel and Kleinbergz [28] formalized the link prediction problem and surveyed an array of methods on measuring the proximity of nodes in a network. Follow-up works applied advanced machine learning methods to social network data. These advanced methods include logistic regression, decision tree, and naive Bayesian [21] as well as supervised random walks [5]. Besides the obvious semantic differences in the two problems, our work differs from social network link prediction.

- Links in social networks connect nodes that are considered equivalent by a given logical relationship. While, in our model, links are triggered by a hierarchical relationship between nodes. This conceptual difference makes it possible for our model to create pairwise features for finding the semantic relations, rather than analyzing the similarity of the nodes, or the link strength in a network.
- Our TRG construction operation and root-trigger security analysis are unique and beyond the link prediction type of inference problem.

Malware analysis studies build similar dependency graphs to generalize the malware behaviors [4, 25]. Kolbitsch *et al.* [25] analyzed malware programs and extract the dependency between the system calls. Besides the research domain, differences between their work and ours are significant. We adopted the machine learning tools to draw the TRG, while they used specification construction algorithm to generate the behavior graphs. In addition, we use the *vagabonds* in the TRG to identify the malicious network requests, while they use malicious behavior graphs to match the behavior of unknown programs. Similarly, Babic *et al.* [4] continued that line of research and built data-flow dependency graph based on their inference algorithm. Therefore, the construction mechanism and application of the dependency graphs in our work differ from those in the aforementioned approaches.

6. CONCLUSIONS AND FUTURE WORK

We presented a new traffic-reasoning technique for detecting the network activities of stealthy malware. The analysis approach exploring request-level traffic structures and semantic triggering relations is new. We demonstrated the use of triggering relation discovery as a useful security analysis approach, and showed its effectiveness against browser spyware, DNS bot, and data exfiltrating malware. Our evaluation showed high accuracy of the triggering relation prediction. Our analysis identified several types of network anomalies caused by traffic to malicious or misconfigured servers. For future work, we plan to design more complex security definitions and models for utilizing the triggering relation graphs to detect stealthy malware activities. We also plan to investigate the model retraining for practical deployment.

7. ACKNOWLEDGMENTS

We are grateful to the anonymous reviews for their insightful comments. This work has been supported in part by NSF grant CAREER CNS-0953638.

8. REFERENCES

[1] DNScat. A tool to tunnel traffic through DNS servers. `http://tadek.pietraszek.org/projects/DNScat/`.
[2] Tlogger. An Firefox extension. `http://dubroy.com/tlogger/`.

[3] H. Almohri, D. Yao, and D. Kafura. Process authentication for high system assurance. *IEEE Transaction on Dependable and Secure Computing (TDSC)*, 2014.

[4] D. Babić, D. Reynaud, and D. Song. Malware analysis with tree automata inference. In *Computer Aided Verification*, pages 116–131. Springer, 2011.

[5] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 635–644. ACM, 2011.

[6] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[7] P. V. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proceedings of ACM SIGCOMM*, August 2007.

[8] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding malicious domains using passive DNS analysis. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*, February 2011.

[9] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl. Automating network application dependency discovery: Experiences, limitations, and new solutions. In *Proceedings of OSDI*, pages 117–130, 2008. USENIX Association.

[10] H.-K. Choi and J. O. Limb. A behavioral model of web traffic. In *Network Protocols, 1999.(ICNP'99) Proceedings. Seventh International Conference on*, pages 327–334.

[11] M. Christodorescu, S. Jha, and C. Kruegel. Mining specifications of malicious behavior. In *ISEC*, pages 5–14, 2008.

[12] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[13] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In *Proceedings of 19th International World Wide Web Conference*, 2010.

[14] W. Cui, Y. H. Katz, and W. tian Tan. BINDER: An Extrusion-based Break-In Detector for Personal Computers. In *Proceedings: USENIX Annual Technical Conference*, page 4, 2005.

[15] C. Elkan. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, volume 17, pages 973–978, 2001.

[16] L. Getoor and C. P. Diehl. Link mining: a survey. *SIGKDD Explor. Newsl.*, 7(2):3–12, Dec. 2005.

[17] T. M. Green, W. Ribarsky, and B. Fisher. Visual analytics for complex concepts using a human cognition model. In *Visual Analytics Science and Technology, 2008. VAST'08. IEEE Symposium on*, pages 91–98. IEEE, 2008.

[18] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium*, 2008.

[19] 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.

[20] G. John and P. Langley. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann, 1995.

[21] I. Kahanda and J. Neville. Using transactional information to predict link strength in online social networks. In *Proceedings of the Third International Conference on Weblogs and Social Media (ICWSM)*, 2009.

[22] S. Kandula, R. Chandra, and D. Katabi. What's going on? Learning communication rules in edge networks. In *Proceedings of ACM SIGCOMM*, August 2008.

[23] A. Keller, U. Blumenthal, and G. Kar. Classification and computation of dependencies for distributed management. In *Proceedings of International Symposium on Computers and Communications*, pages 78–83, 2000.

[24] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen. Enriching intrusion alerts through multi-host causality. In *Proceedings of Network and Distributed System Security (NDSS)*, 2005.

[25] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X.-y. Zhou, and X. Wang. Effective and efficient malware detection at the end host. In *USENIX Security Symposium*, pages 351–366, 2009.

[26] W. Lee, S. J. Stolfo, and K. W. Mok. A data mining framework for building intrusion detection models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 120–132. IEEE, 1999.

[27] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. G. Greenberg, and Y.-M. Wang. WebProphet: Automating performance prediction for web services. In *NSDI*, volume 10, 2010.

[28] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[29] P. Likarish, E. E. Jung, and I. Jo. Obfuscated malicious JavaScript detection using classification techniques. In *Proceedings of 4th International Conference on Malicious and Unwanted Software*, 2009.

[30] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer. Using machine learning techniques to identify botnet traffic. In *2nd IEEE LCN Workshop on Network Security (WoNS) 2006*, pages 967–974, 2006.

[31] A. Natarajan, P. Ning, Y. Liu, S. Jajodia, and S. E. Hutchinson. NSDMiner: Automated discovery of network service dependencies. In *INFOCOM*, pages 2507–2515, 2012.

[32] T. T. T. Nguyen and G. J. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials*, 10(1-4):56–76, 2008.

[33] Panda Security Report. 2013. `http://press.pandasecurity.com/press-room/reports/`.

[34] Botnet Pony 1.9 Malware. `http://laboratoriomalware.blogspot.com/2013/01/botnet-pony-19-malware.html`.

[35] A. Srivastava, S. Sural, and A. Majumdar. Database intrusion detection using weighted sequence mining. *Journal of Computers*, 1(4):8–17, 2006.

[36] 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.

[37] H. Tan, N. Goharian, and M. Sherr. $100,000 prize jackpot. Call now!: Identifying the pertinent features of SMS spam. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 1175–1176. ACM, 2012.

[38] N. Williams, S. Zander, and G. Armitage. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *SIGCOMM Comput. Commun. Rev.*, 36(5):5–16, Oct. 2006.

[39] P. Xie, J. H. Li, X. Ou, P. Liu, and R. Levy. Using Bayesian networks for cyber security analysis. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pages 211–220. IEEE, 2010.

[40] K. Xu, P. Butler, S. Saha, and D. Yao. DNS for massive-scale command and control. *IEEE Trans. Dependable Sec. Comput.*, 10(3):143–153, 2013.

[41] K. Xu, H. Xiong, C. Wu, D. Stefan, and D. Yao. Data-provenance verification for secure hosts. *IEEE Trans. Dependable Sec. Comput.*, 9(2):173–183, 2012.

[42] C. Yang, R. C. Harkreader, and G. Gu. Die free or live hard? Empirical evaluation and new design for fighting evolving twitter spammers. In *Recent Advances in Intrusion Detection*, pages 318–337. Springer, 2011.

[43] A. Zand, G. Vigna, R. Kemmerer, and C. Kruegel. Rippler: Delay injection for service dependency detection. Technical report, UCSB, 2013.

[44] H. Zhang, W. Banick, D. Yao, and N. Ramakrishnan. User intention-based traffic dependence analysis for anomaly detection. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, pages 104–112. IEEE, 2012.

APPENDIX

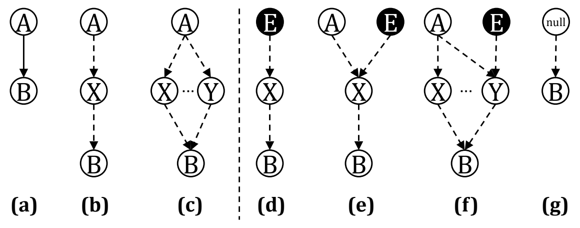

Figure 4: The illustration of various cases where B's predicted root trigger is correct (a-c) or wrong (d-g) on the triggering relation graph constructed from pairwise triggering relations. Let the ground truth of B's root trigger be A. Case (a) is where B's parent is also B's root. Cases (b) and (c) are where there is one or more paths from the single root A to B, respectively. Cases (d), (e), and (f) are where the predicted root of B is or includes a node other than A (e.g., E). Case (g) is where the predicted root of B is null, i.e., no root trigger.

By the definition of triggering relation graph (TRG) in Section 2, each node on a valid TRG should have at most one parent and thus at most one root trigger. In reality, we relax the definition in that this property may not hold in the TRGs constructed from pairwise classification results, e.g., a node may have multiple paths leading to the same root, or multiple paths leading to different roots. Therefore, our TRG construction algorithm needs to find *all* the root triggers of a network event, which makes the problem equivalent to compute the transitive reduction of a direct graph.

We illustrate the various cases where an event's predicted root trigger is correct (a-c) or wrong (d-g) on the triggering relation graph constructed from pairwise triggering relations in Figure 4. Our root-trigger definition allows the existence of multiple intermediate parents for a node, as long as the root trigger is correct, e.g., Figure 4 (c).

Feature	Rank		Brief Definition
	IG	GR	
HTTPRank	1	1	Rank of B in HTTP.
QueryHostSim	2	2	$Sim(A.query, B.host)$.
QueryDomainSim	3	3	$Sim(A.query, Dom(B.host))$.
TimeDiff	4	4	Time difference (ms).
QueryRefSim	5	7	$Sim(A.query, Dom(B.ref))$.
MissingRef	6	5	If $B.ref$ is null.
HTTPType	7	6	If B is IPv4 or IPv6.
DuplicatedDNS	8	8	# of same DNS after A.
PIDDiff	9	9	If both PIDs are equal.

Table 9: Feature ranking by InfoGain (IG) and GainRatio (GR) Selection on dataset II. Denote DNS and HTTP requests as A and B, and define $Dom(URL)$ to get the domain of a URL.

Feature	Rank		Brief Definition
	IG	GR	
DiffAck2Seq1	1	2	Calculate $B.ack - A.seq$.
ExpectedAck	2	1	If $B.ack = A.seq + A.len$.
TimeDiff	3	7	Time difference (ms).
Flag1	4	3	Control bits in A.
Len1Zero	5	5	If $A.len$ is 0.
Len1Large	6	4	If $A.len \geq$ MSS.
MatchAck1Seq2	7	8	If $A.ack = B.seq$.
Flag2	8	9	Control bits in B.
MissingAck1	9	6	If $A.ack$ is null.

Table 10: Feature ranking by InfoGain (IG) and GainRatio (GR) Selection on dataset III. Two TCP packets are A and B, which B follows A.