COMPUTERS & SECURITY 58 (2016) 180–198

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Causality reasoning about network events for detecting stealthy malware activities [1]

Hao Zhang [a], Danfeng (Daphne) Yao [a,*], Naren Ramakrishnan [a], Zhibin Zhang [b]

[a] Department of Computer Science, Virginia Tech, Blacksburg, VA, USA
[b] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

ARTICLE INFO

Article history:
Received 31 July 2015
Received in revised form 20 December 2015
Accepted 11 January 2016
Available online 21 January 2016

Keywords:
Network security
Anomaly detection
Stealthy malware
Traffic analysis
Dependence analysis
Machine learning classification

ABSTRACT

Malicious software activities have become more and more clandestine, making them challenging to detect. Existing security solutions rely heavily on the recognition of known code or behavior signatures, which are incapable of detecting new malware patterns. We propose to discover the triggering relations on network requests and leverage the structural information to identify stealthy malware activities that cannot be attributed to a legitimate cause. The triggering relation is defined as the temporal and causal relationship between two events. We design and compare rule- and learning-based methods to infer the triggering relations on network data. We further introduce a user-intention based security policy for pinpointing stealthy malware activities based on a triggering relation graph. We extensively evaluate our solution on a DARPA dataset and 7 GB real-world network traffic. Results indicate that our dependence analysis successfully detects various malware activities including spyware, data exfiltrating malware, and DNS bots on hosts. With good scalability for large datasets, the learning-based method achieves better classification accuracy than the rule-based one. The significance of our traffic reasoning approach is its ability to detect new and stealthy malware activities.

1. Introduction

Recent advancements in information technology have raised concerns on the security risks posed by the prevalence of malicious software. A study showed that a significant portion of computers worldwide is infected with malware conducting clandestine activities (Panda Security Report). Malware may spy on the victim user, cause data exfiltration, and abuse the computer for conducting bot activities (e.g., command-and-control). The initial infection vector of most malware is usually through exploiting vulnerabilities of common networked software, e.g., heap overflow vulnerability in a web browser or its plug-ins (Cova et al., 2010). 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 inspecting individual network requests, a more effective network security approach is to discover characteristic behavioral patterns in network event attributes. For example, BINDER (Cui et al., 2005) detects anomalous network activities on personal computers through analyzing the correlation in network events by the temporal and process information. BotMiner (Gu et al., 2008) performs a correlation

[1] The preliminary version of this work appeared in the Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS), Kyoto, Japan, June 2014 (Zhang et al., 2012) and in the Proceedings of 33th IEEE Symposium on Security and Privacy Workshops (SPW), San Francisco, CA, May 2012 (Zhang et al., 2014). This work was supported in part by an NSF grant CAREER CNS-0953638, ARO YIP W911NF-14-1-0535, and L-3 communications.

* *Corresponding author.* Tel.: +1 540 231 7787.
E-mail address: danfeng@vt.edu (D.(D.) Yao).

http://dx.doi.org/10.1016/j.cose.2016.01.002

analysis across multiple hosts of a network for detecting similarly infected bots. King et al. (2005) constructed directed graphs from logs to show network connections for dissecting attack sequences. However, none of these above solutions is designed for detecting general stealthy malware activities. Thus, they cannot be directly applied to our problem.

In this work, we describe a new *triggering relation discovery* problem to construct the request-level causality structure in network traffic. There have not been systematic studies on network-request-level causal analysis for malware detection. Existing dependence analysis work (e.g. Natarajan et al., 2012; Zand et al.) focuses on the network services and is not designed for malware detection. For example, NSDMiner (Natarajan et al., 2012) addressed the problem of network service dependency for automatic manageability and network stability. Rippler (Zand et al.) is proposed to actively perturb or delay traffic to understand the dependencies between service and devices. In comparison, we aim at achieving the *request-level* causality structure in network traffic. This finer granularity (request vs. flow) requires different relation semantics and more scalable analysis methods.

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. 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. We develop rule- and learning-based methods that detect network activities of stealthy malware activities through reasoning their causal relationship. We design a new method *pairing* that produces special pairwise features in the learning-based approach, so that the discovery problem can be efficiently solved with existing binary classification methods (e.g., SVM). By enforcing a *root-trigger* policy, we can identify the suspicious events that lack of valid triggers, and thus ensure an application's correct responses to user activities.

The 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 (Green et al., 2008; Zhang et al., 2015). Thus, analyzing relations among network events provides important insights for identifying general stealthy malicious activities. The causality offers the logical interpretation to the vast amount of otherwise structureless and contextless network events. Our work demonstrates that triggering relations among cyberspace events enables the network assurance with structural evidence of the hosts.

Our contributions are summarized as follows.

1. We formalize the problem of triggering relation discovery in network requests, the data structure of triggering relation graph, and their applications for detecting stealthy malware activities.
2. We present two approaches for discovering the triggering relations among network events. First, we design a discovery algorithm based on empirically derived rules. In the rule-based approach, we inspect the temporal, semantic, and process-related attributes to reveal the causal relations among network requests. Then, we propose an advanced learning-based approach. A new feature extraction method is introduced as the *pairing* operation. It performs pairwise attribute comparisons, enabling the use of binary classifiers for our triggering relation discovery.
3. We adopt a new *root-trigger security policy* on discovered triggering relations for malware detection. This policy allows one to identify *vagabond events*, i.e., network events that do not have proper causes to justify their occurrences.
4. We extensively evaluate the classification and detection accuracy rates of our solution with DARPA dataset and real-world network traffic, including HTTP, DNS, and TCP traffic. Both rule- and learning-based methods are compared in identifying the dependencies and the learning-based one yields better prediction accuracy rates than rule-based one. We show that our dependence analysis is effective in detecting stealthy malware activities (e.g., HTTP-based malware and DNS bots).

Triggering relation discovery provides a new perspective for analyzing network traffic. It allows one to reason about the occurrences of network events, to detect unexplained network activities that are due to stealthy malware. The significance of our traffic reasoning approach is its ability to detect new and stealthy malware activities.

Major new results reported in this journal publication compared with previous conference papers (Zhang et al., 2012, 2014) are summarized as follows. We report the side-by-side comparisons of the rule-based and learning-based dependency analysis approaches. We perform new experiments evaluating the rule-based approach in discovering the triggering relations on HTTP traffic and in detecting HTTP-based malware (e.g., stand-alone data exfiltrating malware). We conduct new experiments on a DARPA dataset to evaluate our solution in detecting the stealthy malicious activities (exploits due to vulnerable applications). Last but not least, this journal version presents the rule-based and learning-based computation techniques under a unified triggering relation discovery framework. Such an abstraction can help motivate and inspire future research on causality reasoning for security.

1.1. *Organization*

We introduce the triggering relation graph and its security applications in Section 2. Then we present the rule- and learning-based methods in Sections 3 and 4, respectively. We illustrate the root-trigger policy and its use in Section 5. We systematically evaluation our solution with real-world malware and synthetic attacks in Section 6. Related work and conclusion are presented in Sections 7 and 8.

2. Model and overview

In this section, we introduce the concept of triggering relationship, 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 the reason that directly causes e_j to occur. The specific semantics of triggering relation depend on the type of events and environment. An event may be defined

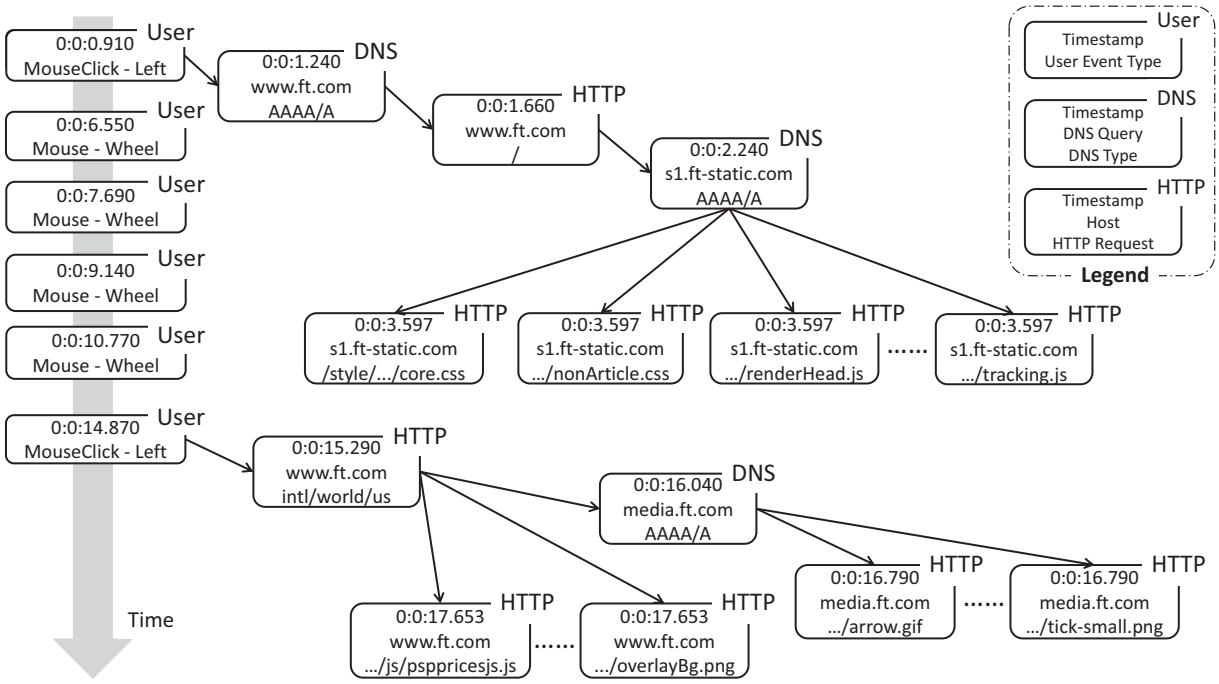

Fig. 1 – An example of triggering relation graph for outbound DNS and HTTP traffic on a host. The network and user events are denoted as rounded squares. The arrows represent the triggering relations between events.

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. A TRG provides a structural representation of triggering relations of observed events.

For specific types of network traffic, the TRG may manifest unique topology and properties. We present one concrete example to demonstrate the triggering relations existing in HTTP and DNS requests in Fig. 1. After a user clicks a link to Financial Times (http://www.ft.com), the browser first resolves the IP address by sending out a DNS query. Corresponding HTTP requests are issued after the IP address is known. Then, the user is directed to a news page by clicking a link. The TRG forms a forest of trees that consists of user events and network requests. For each tree in TRG, we name its user event as a *root-trigger*.

We summarize the properties of TRG as follows.

- Attribute-based: Each event in TRG has multiple attributes, describing its network and kernel information.
- Expandable: TRG is a forest of trees in arbitrary height. The degree of a node in a tree can be expanded as it grows. The legitimacy of the tree is determined by its *root*, i.e., the dependent nodes are benign as long as the root is legitimate.
- Acyclic: The edge in each tree connected between two events is unidirectional. Because of the temporal property of events, triggering relation graphs are free of cycles.
- Sparsity: A TRG is usually sparse, i.e., the number of a node's neighbors compared to the total numbers of nodes is small. The triggering relations occur in a short time range. Two events rarely form the triggering relationship, if they are far apart in terms of time.

The problem of *triggering relation discovery* is that given a set of events, to construct the complete triggering relation graph corresponding to the events. To solve this problem, one direction is to construct the TRG by incrementally inserting new events to a partially well-formed TRG by predefined algorithms. The algorithms need to be generated by domain experts using the application specific knowledge. The other direction is to adopt learning-based classifiers to infer the triggering relations. The TRG comprises a plural of triggering relations between two requests. Therefore, one can construct a complete TRG by given the existence of edges between pairs of nodes and the directions of the edges. We illustrate the TRG construction operation in Fig. 2.

We, therefore, transform the problem of discovering triggering relations among a set of events into discovering the

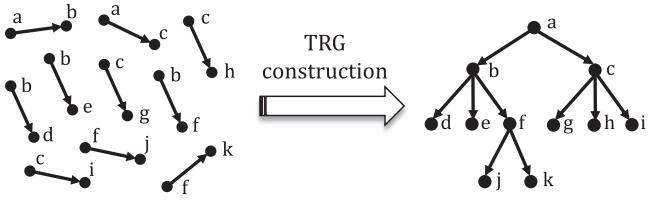

Fig. 2 – The triggering relation graph (TRG) on the right can be constructed from the pairwise triggering relations on the left.

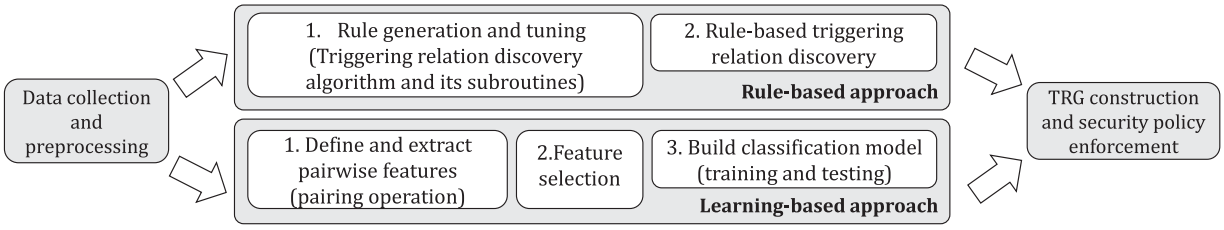

Fig. 3 – The system workflow containing major operations in the rule- and learning-based approaches.

triggering relations of pairs of events, which is defined as the *pairwise* triggering relation. In our context, the *pairwise* triggering relation discovery is a simpler problem, which is to determine whether a triggering relation exists in two events. To this end, we design a novel *pairing* operation that produces pairwise features, so that the discovery problem can be efficiently solved using classification tools. The main operations of rule- and learning-based approaches in our solution are illustrated in Fig. 3.

We further make a detailed comparison of properties between these two triggering relation discovery approaches in Table 1. The rule-based approach is only applicable for known patterns, and requires non-trivial human efforts in rule generation and tuning. Yet the learning-based approach extracts the pairwise features that can characterize the relationship between nodes, and thus the generalized triggering relation model adapts to diverse and complex patterns.

Our definition of event-level triggering relation relates to, but differs from the definition of service dependency in existing research work (Natarajan et al., 2012; Zand et al.). Service dependency refers to that one service that relies on another to function, e.g., a web service depends on the DNS resolution. Our event-level triggering relation refers to the causality of two events, e.g., the transmission of one network packet triggers the transmission of the other.

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. In a user-intention based security model, the user-input actions serve as the root triggers. The analysis can pinpoint 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 (Xu et al., 2013). These abnormal outbound DNS queries are automatically generated by malware on the host, typically with the 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.

With a built TRG, more security policies can be proposed depending on the host types (e.g., client or server), traffic types (e.g., HTTP, TCP, or mixed), and the definitions for network anomalies. The inferred dependency in a TRG illustrates the logic chains of the network requests, which reveals the origin and time range of the malicious activities. Our model can be used in combination with conventional signature-based and statistic-based detection. In Section 6.2.4, we further conduct an experiment on a dataset that contains a DDoS attack, which shows our model can be used as a complementary to existing solutions.

Our work on triggering relation discovery demonstrates a specific big data security approach, where we are capable of analyzing voluminous network traffic to identify anomalies. These discovered relations produce structural and contextual information for reasoning and justifying the occurrences of system and network behavior patterns. The advantage of our approach is the ability to provide proactive system defenses.

Table 1 – Comparison of properties between rule-based and machine learning (ML) based approaches.

Approach	Rule-based	ML-based
Sub-goal	To recognize parent–child relations based on event attributes.	
Operation	Rule generation, rule-based identification	Feature extraction, train & test
Model	Empirically derived rules	Automatic generated
Labeled data	No	Yes (for training)
Manual effort	Needed for generating rules	Minimal
Major cost	Rule tuning by human	Pairing operation
Flexibility	Low (cannot recognizes beyond rules)	High (can adapt to subtle cases)

3. Rule-based discovery

We describe a rule-based approach in this section. The method used is to identify the triggering relation between a new incoming event and existing events in a TRG.

A TRG can be constructed incrementally by inserting a new network event with unknown dependency to a well-formed TRG, which is suitable for real-time monitoring. The construction of TRG relies on the attributes of events and dependency rules derived from the specific application. In our work, we generate rules for discovering triggering relations of HTTP requests on a host. Our rules are derived based on the patterns of user interactions and the attributes of HTTP requests from the browser including their system properties (Zhang et al., 2012).

3.1. *Overview of rule-based discovery*

Given a new request, the triggering relation discovery (TRD) algorithm aims at identifying 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 TRG. The requests with known dependencies are chronologically organized into trees rooted by user events in the existing TRG. The root-triggers, thus, are also chronologically ordered.

The pseudocode of our rule-based triggering relation discovery procedure is shown in Algorithm 1. A new tree is created if the newly observed event p is a root request. Otherwise, existing trees are searched in reversed chronological order. The searching is stopped if: *i)* the triggering relation of p is found, so that it can be attached to an existing tree T_i; *ii)* no such tree is found after all nodes on the TRG are compared, which indicates p is a vagabond request, and thus suspicious. Based on what our experiments have shown, the incoming new request is commonly caused by recent requests and the trees in a constructed TRG. Therefore, our algorithm opts for a breadth-first traversal within a tree starting from the most recent root-triggers, a strategy that allows us to efficiently and effectively identify the parent node of the newly observed request.

Algorithm 1 Rule-based Triggering Relation Discovery (TRD)

Input: A newly-observed network event p; a TRG $\mathbb{T}$ of the chronologically ordered trees $\{T_1, \ldots, T_m\}$ of events rooted by root requests $\{r_1, \ldots, r_m\}$, where r_m is the most recent one; and a threshold τ.

Output: True, if the parent node of request p is found; False, otherwise.

```
1: if IsRoot(p) then
2:    create a new tree T_{m+1} with root p
3:    𝕋 ← 𝕋 ∪ {T_{m+1}}
4:    return True
5: else
6:    for i ← m to 1 do
7:       continue if p.time − r_i.upTime > τ or r_i.pid ≠ p.pid
8:       define a queue Q and enqueue r_i onto Q
9:       while Q ≠ ∅ do
10:         node n ← dequeue Q
11:         if IsChild(n, p) then
12:            UpdateTime(r_i.upTime)
13:            append {n → p} to T_i
14:            return True
15:         else if IsSibling(n, p) and not IsRoot(n) then
16:            UpdateTime(r_i.upTime)
17:            append {n.parent → p} to T_i
18:            return True
19:         else
20:            ▷ Breadth-first traversal by queue Q
21:            for all children of node n do
22:               enqueue the child nodes onto Q
23:            end for
24:         end if
25:      end while
26:   end for
27: end if
28: return False
```

We further optimize the algorithm by avoiding unnecessary comparisons. We achieve the speedup by leveraging the underlying consistency among attributes (e.g. PID). Besides, we adopt the UpdateTime function to refresh the upTime for each tree in $\mathbb{T}$, in which way we can track the freshness of a tree and skip the comparison between a newly coming request and an out-of-date tree. The worst-case complexity of this TRD algorithm requires traversing the entire TRG, and is $O(n)$ where n is the total number of network events on the TRG.

3.2. Details of classification rules

In the TRD algorithm (Algorithm 1), we instantiate three building blocks IsRoot, IsChild, and IsSibling to facilitate the process of inferring triggering relations.

The IsRoot procedure is used to test whether or not a request is the first one triggered by a user event. 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. Therefore, the corresponding root request 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 root request.

The IsChild procedure is used to test whether or not there is a triggering relation (i.e., parent–child relation) between requests. Given two requests, p_a and p_b, where p_a is a node on TRG with known dependency and p_b's dependency is unknown. The event p_a triggers p_b if and only if the following conditions are all satisfied:

- The interval between timestamps of p_a and p_b is within a threshold τ and event p_a proceeds p_b.
- The two outbound network requests p_a and p_b share the same (non-null) process ID.
- The domain name in p_b's referrer is identical to that of p_a.

The IsSibling procedure is used for the nodes whose parent nodes cannot be directly determined. Therefore, identifying the sibling relation of a request helps establish a triggering 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 the rules as follows.

- The interval between timestamps of p_a and p_b is within a threshold τ and event p_a proceeds p_b.
- The two outbound network requests p_a and p_b share the same (non-null) process ID.
- Referrers of both requests are non-null and identical.

The IsSibling procedure is a necessary complement to the IsChild procedure as IsSibling helps identify late-arriving child nodes whose intervals of timestamps with respect to the parents are larger than the specified threshold, yet whose intervals with respect to the (older) sibling are still within the threshold.

4. Learning-based discovery

We describe a machine learning approach to inferring triggering relations on network requests in this section. As compared in Table 1, the learning-based solution provides a more adaptive solution than the heuristic rules. 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 then show how binary classification algorithms can be used for the triggering relation discovery.

4.1. Overview of learning approach

The main operations in our learning-based method are Pairing, Data Labeling, Training, Classification, and TRG Construction. The Data Labeling, Training and Classification operations are standard for machine learning based methods. The new operations are Pairing and TRG Construction. Given an event e (e.g. an HTTP request) that has one or more attributes (A_1, … , A_m) describing its properties, we elaborate the learning operations as follows.

- Pairing is a new operation that we design for extracting pairwise comparison results (i.e., features) of events' attributes. Its inputs are two events $e=(A_1,\dots,A_m)$ and $e'=(A'_1,\dots,A'_m)$. The output is the event pair (e,e') with m pairwise attribute values (B_1, … , 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 events. The comparison function $f_i()$ (e.g., IsEqual, IsGreaterThan, WithinThreshold, 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 triggering relation. Moreover, we demonstrate an efficient pairing algorithm and advanced strategies to reduce the cost of pairing without compromising the analysis accuracy in Sections 4.3 and 4.4. 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., $\langle(e,e'),1\rangle$ represents that event e triggers e'. Data labeling is based on pairwise attributes (e.g., B_1, … , 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 $\{\langle(e_i,e_j),l_{ij}\rangle\}$, 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 build 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 and e_j are connected by a directed edge in the TRG.

We describe and highlight the design details of our new Pairing operation in this section. This feature extraction operation is unique in that the features enable the use of binary classifiers for pairwise directional relation discovery.

4.2. *Pairing operation for feature extraction*

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). 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. The *pairing* details are illustrated as follows.

- Numeric attributes (e.g., timestamps) are compared by computing their difference, e.g., the interval 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, the similarity index can be computed between two host domains, two referrer fields, and two request URLs.
- A composite attribute can be converted to primitive types, e.g., a destination address containing four octets for the IP address and an integer for the port. Therefore, the comparison of two composite attribute values is made by comparing the sub-attribute values separately.

4.3. *Efficient pairing algorithm*

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 events that occur within a certain time frame τ, assuming that events occurring far apart are unlikely to have triggering relations.

We describe the efficient pairing logarithm, a more sophisticated pairing heuristic. It prescreens attributes to quickly eliminate unqualified pair candidates. Shown in Algorithm 2, it takes a list of chronologically sorted network requests as the input and outputs a set of pairs of events. The efficient pairing algorithm uses a dictionary to store the recent network events, which are the candidates of triggers for the 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 2) 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. It is confirmed from our experiments that the efficient pairing algorithm significantly reduces the size of pairwise data, without scarifying the detection accuracy.

Algorithm 2 Efficient Pairing Algorithm (EPA)

Input: A list of chronological sorted events, $\mathbb{L} = \{e_i\}$.
Output: A set of event pairs, $\mathbb{P} = \{(e_i, e_j)\}$, $1 \leq i < j$.

```
define a set ℙ to store the compared pairs {(e_i, e_j)}
define a dictionary D = (d, {e}), where d is the domain of event and {e} is a set of events whose domain is d.
for each event e_j ∈ 𝕃 do
    d ← the domain of e_j's Host
    if e_j's Referrer is not null then
        dom ← the domain of e_j's Referrer
    else
        dom ← d
    end if
    if dom in D's keyset then
        for each event e_i in D[dom] do
            if pass the Screening(e_i, e_j) then
                ℙ ← ℙ ∪ Pairing(e_i, e_j)
            end if
        end for
        calculate the expire time and update D[d]
        add e_j in D[d]
    else
        add new entry (d, {e_j}) in D
    end if
end for
return ℙ
```

Table 2 – Semantics of values in a cost matrix.

		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.

Note: TR stands for triggering relation.

4.4. *Efficiency pairing using parallel computing*

Pairing is a time-consuming operation step in the learning-based approach. To improve the efficiency, we propose to leverage two advanced strategies to reduce the running time.

4.4.1. *Divide-and-conquer*

Given a list of records $\mathbb{R} = \{r_1, r_2, \ldots, r_n\}$, we divide $\mathbb{R}$ into k consecutive blocks and assign them to multiple machines. Each host outputs are the intra-block pairs. For every two neighbor blocks, we merge them and process pairwise comparisons until all records have been paired. From the second iteration, the task is the inter-block comparisons. The overall complexity is still bounded by $O(n^2)$. However, the pairing computation on disjoint data can be made parallel.

4.4.2. *MapReduce*

Using the same blocks described in the divide-and-conquer method, we assign each host two blocks T_i and T_j $(1 \le i \le j \le k)$ as inputs. The hosts perform the pairing operation between any two records a and b, where $a \in T_i$ and $b \in T_j$. The outputs are the intra-block pairs, if i and j are equal. Otherwise, the outputs are the inter-block pairs.

The MapReduce framework is more suitable for the pairing operation, as it is a type of data-intensive computation. The divide-and-conquer approach is easy to implement on both multi-thread and multi-host environments, while the workload issue may hinder its scalability.

4.5. *Feature selection and cost matrix*

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 (John and Langley, 1995), and a support vector machine (SVM) (Cortes and Vapnik, 1995).

Due to the sparsity of triggering relations existing in network traffic, we leverage the customized cost matrices (Elkan, 2001) to penalize missed relations during the training. In cost-sensitive classifiers, 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 2, the cost matrix used in our model 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 FNs and FPs. The cost-sensitive classification takes a cost matrix as an input. The trained model aims at minimizing the total penalty in imbalanced datasets. 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 6, we thoroughly evaluate how cost matrices improve our analysis accuracy.

5. Root-trigger security policy

Given the parent node of each request using the rule-based method, or given the predicted results of pairwise triggering relations in the learning-based method, we can construct a triggering relation graph (TRG). The graph serves as a source for locating anomalous network activities.

The security policy defines the legitimate and abnormal events, which can be used to analyze the TRG and make security decisions. A specific root-trigger security policy is based on the *user intention* (Zhang et al., 2012), 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.

Under the *root-trigger* security policy, 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. Anomalous events are those without a valid root trigger, namely the *vagabonds*. These events may be due to malware activities or server misconfiguration.

To enforce the root-trigger policy, we identify the root triggers of all the events. Each node on a valid TRG should have at most one parent, as per the TRG definition in Section 2.1. With the results obtained from the TRD algorithm, it is unambiguous to trace back to its root-trigger for each newly-observed event. However, in the learning-based approach, the pairwise classification results may lead to multiple parent events. We, therefore, design a root finding algorithm to obtain the root of an event, given all predicted pairwise triggering relations. Shown in Algorithm 3, to find the root of each event by traversal in TRG is equivalent to the transitive reduction of a directed graph.

The inputs of the root finding algorithm 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 $\mathbb{R}$ that includes all the roots of e_k. To compute the transitive reduction, we opt for a queue Q to perform the 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 $\mathbb{T}$, we add e to the return set $\mathbb{R}$ if it is a root-type event. 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, namely the vagabonds, are flagged and alerted to the administrator for further inspection.

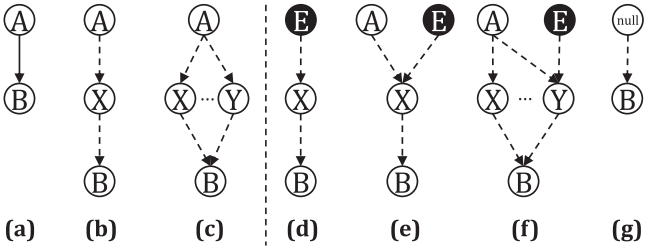

Fig. 4 – The illustration of various cases where *B*'s predicted root trigger is correct (a–c) or wrong (d–g) on the triggering relation graph constructed from pairwise triggering relations. Let the ground truth of *B*'s root trigger be *A*. Case (a) is where *B*'s parent is also *B*'s root. Cases (b) and (c) are where there are 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.

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

We infer the root-triggers by integrating the system and network information. For example, we can log the application information by a browser extension (e.g., Tlogger). The browser extension provides browser and tab events that can be used to identify the user-HTTP dependency and user-DNS dependency. Besides, we adopt a heuristic method to infer the root-triggers. The method determines root-triggers if the requests are larger than B bytes and have at least T millisecond away from the last root-trigger. The heuristic method applies for the lower level protocols (e.g., TCP) that are not easily linked to user's events.

We demonstrate the use of our method for detecting three types of common malware in Section 6, including

- spyware as a browser extension,
- data-exfiltrating malware as a stand-alone process,
- a DNS-based botnet command and control channel,
- a remote break-in due to the software vulnerability.

6. Evaluations and results

We implemented our solution and evaluated using different types of network data. We have conducted extensive tests on our triggering relation analysis and obtained positive results. The questions we seek to answer through our experiments are:

- How accurate is our learning-based approach for inferring the triggering relations on different types of network traffic? (Section 6.2)
- Can our solution detect outbound network activities caused by stealthy malware and real-world threats? (Section 6.3)
- Comparison to the rule-based TRD algorithm, how well does our machine learning approach perform? (Section 6.4)
- How efficient is the learning-based approach? What is the most time-consuming operation in learning-based approach? (Section 6.4)

6.1. *Experimental overview*

We describe the setup of our experimental evaluation in this section. Then, our evaluation results are presented in the next few sections.

6.1.1. *Datasets*

Our evaluation is mainly focused on the network traffic via HTTP and DNS protocols, which are commonly used communication protocol both by legitimate users and attackers, and most firewalls allow them (Xu et al., 2013). Because of the privacy concerns (e.g., most application layer requests are in plain text), there is no known public data source that includes both HTTP traffic and user's inputs, so we have to collect data on our own. We collect and analyze outbound HTTP and DNS requests from hosts, aiming to detect suspicious activities by stealthy malware. Additionally, we adopt our approach on a DARPA dataset (MIT Lincoln Laboratory). This dataset contains network requests over multiple subnets and it is originally created for assessing the intrusion detection systems. Last, we evaluate the scalability of our learning-based approach with a much larger TCP dataset collected from a server. The details of data collection are listed as follows.

- 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 valid GET, HEAD, or POST information in the headers.
- Dataset III (UDP and TCP). The dataset includes a DDoS attack scenario. The attackers first performed IPsweep and probed the hosts that run the Sadmind service. Then, three hosts got infected and were installed malicious scripts. Last, attackers launched a DDoS attack from the victim machines. In this experiment, we focus on our detection to the individual host and verify whether our method could be used for identifying the break-ins on each host.
- Dataset IV (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.

A summary of the experimental data is shown in Table 3. In dataset I, we manually check the legitimacy of vagabond requests after running TRD Algorithm. In dataset III, the malicious requests are labeled based on the IDMEF alerts and audit logs (MIT Lincoln Laboratory). In datasets II and IV, we set up a firewall, run antivirus software and install a commercial IDS when collecting the data, so we assume the collected data are clean.

Table 3 – An overview of four datasets in the experiments.

Dataset	I	II	III	IV
Type	HTTP	DNS (D) & HTTP (H)	TCP & UDP[1]	TCP
# of raw events	HTTP: 45,988	D: 35,882; H: 85,223	All types: 649,787	TCP: 3,010,821
Efficient pairing (η)[2]	94.7%	98.8%	96.8%	99.6%
# of event pairs	3,436,635	953,916	47,215,275	119,372,631
# of root-triggers	899	2,795	21,416	45,960
Size (MB)	229.5	55.1	3441.3	6697.1
TR labeling[3]	TRD (Algo. 1)	TRD + rules	TCPFLOW (TCPFLOW 1.3)	TCPFLOW
RT labeling[4]	Tlogger (Tlogger)	Tlogger	Rules	Rules

[1]Dataset III contains 33 different protocols, including TCP, UDP, DNS, and etc.
[2]η is the reduction percentage after using our efficient pairing algorithm (EPA).
[3]TR labeling describes the methods to label triggering relations (TR), e.g., rule-based TRD, TCPFLOW, or other rules (integral analysis of user-HTTP dependency and DNS-HTTP dependency for Dataset II).
[4]RT labeling describes the methods to label root-triggers (RT), e.g., Tlogger and the heuristic rules mentioned in Section 5.

Table 4 – Pairwise features defined on different types of network protocols in pairing operation.

Protocol	Feature category	Illustrations of pairwise features
All types	Address difference	Comparison between two IPs/ports
	Temporal relationship	Time difference, session duration, delta time
Transport layer	Protocol difference	Comparison/relationship between two protocols
	Flag difference	Comparison between control bits/flags in headers.
Application layer	Semantical similarity	Similarity between two request URLs/referrers/domain.
	File type difference	Comparison between file types (e.g., request and referrer).
	System info. difference	Comparison between system attributes (e.g, PIDs).
	Aggregate info.	# of the duplicated domains/referrers/DNS queries.

6.1.1.1. Effectiveness of EPA. We define $\eta \in [0, 1]$ as the reduction percentage in Equation 1, where $EPA(n)$ is the number of event pairs after using the efficient pairing algorithm in Section 4.3, and n is the total number of events.

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

6.1.2. *Pairwise feature extraction on network data*

The pairwise features are defined based on the features of the traffic types. We summarize the categories of pairwise features in Table 4. The comparison between two addresses (IP and port) and the temporal relation are commonly used for all types of network traffic.

For transport layer protocols, the extracted features are obtained in two categories: i) the relationship between two protocols (e.g., co-occurrence, succession, etc), which describes whether the protocol is kept the same and how protocol is changed from one to the other; ii) the comparison between control bits/flags, for example, the SYN and ACK values, the Length of a packet and the MSS (maximum segment size) of a session.

For application layer protocols, besides the aforementioned ones, we extract the pairwise features that have semantic meanings, e.g., the similarity between two request URLs, two referrers, and a domain and a DNS query. This type of features includes human-readable languages and can be analyzed using string comparison and advanced natural language processing techniques. In addition, the aggregate information over time is particularly useful to decide the triggering relationship, when there are some missing or incomplete values in the packet. All pairwise features are finally chosen by two feature ranking algorithms (InfoGain and GainRatio). In our evaluation, we select the most contributive features with the cut-off value 0.01.

6.1.3. *Classification setup of learning-based method*

Three common classification techniques are compared: naive Bayes classifier, a Bayesian network, and a support vector machine (SVM).[1] 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.

6.1.4. *Accuracy and security metrics*

- The conventional *precision and recall* measures (Baeza-Yates et al., 1999) 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.

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN}. \tag{2}$$

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

Table 5 – Pairwise classification results of train-n-test for four datasets.

Data	# of pairs in test sets	Cost matrix	Naive Bayes		Bayesian network		SVM	
			Prec.	Recall	Prec.	Recall	Prec.	Recall
I	3,318,328	$\begin{bmatrix} 0, & 1 \\ 10, & 0 \end{bmatrix}$	0.954	0.996	0.956	0.996	0.958	0.997
II	693,903	$\begin{bmatrix} 0, & 1 \\ 1, & 0 \end{bmatrix}$	0.959	0.998	1.000	1.000	1.000	1.000
III	25,694,154	$\begin{bmatrix} 0, & 1 \\ 30, & 0 \end{bmatrix}$	0.996	0.971	0.996	0.984	0.996	0.965
IV	1,191,926,877	$\begin{bmatrix} 0, & 1 \\ 3, & 0 \end{bmatrix}$	0.995	0.986	0.997	0.998	0.998	0.999

Note: Results are rounded before reporting. Prec. stands for precision in the pairwise classification.

- The *pairwise accuracy* of classification is the percentage of pairwise triggering relations that are predicted correctly by using machine learning classifiers. The pairwise accuracy is evaluated with respect to the ground truth.
- 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) TRG are correct with respect to the ground truth. The metrics allow the existence of one event having multiple paths to the same root in a TRG, which applies to both rule- and learning-based approaches.

6.2. *Evaluation of our learning-based approach*

We present our experimental findings using the learning-based approach on different types of network traffic (e.g., HTTP, DNS, and TCP) in this subsection.

6.2.1. *Evaluation on dataset I*

6.2.1.1. Accuracy of pairwise triggering relations. Table 5 shows a high prediction accuracy rate for pairwise triggering relations. These results indicate the effectiveness of our binary classification approach.

We vary the cost matrices and compute the pairwise accuracy rates of the three classifiers. The results are shown in Fig. 5a. The accuracy rate is consistently high for naive Bayes classifier, despite the changes of 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.

6.2.1.2. Correctness of root triggers. The purpose of this analysis is to identify the reasons for wrong predictions of root triggers. Running the root finding algorithm (in Section 5) on the predicted triggering relations, we obtain the root-triggers of all events and compare them with the ground truth. Fig. 5b shows the correctness of root-trigger analysis under different settings (classifier and cost matrices). The naive Bayes and Bayesian network classifiers 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.

6.2.2. *Detection of malicious traffic in dataset I*

As defined in Section 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 websites. 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). We analyze the geographic locations of the malicious servers based on their IP addresses. All of them are located in the US, except one IP located in Netherlands.

In our model, some vagabond requests are false alerts (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 bookmark icons.

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

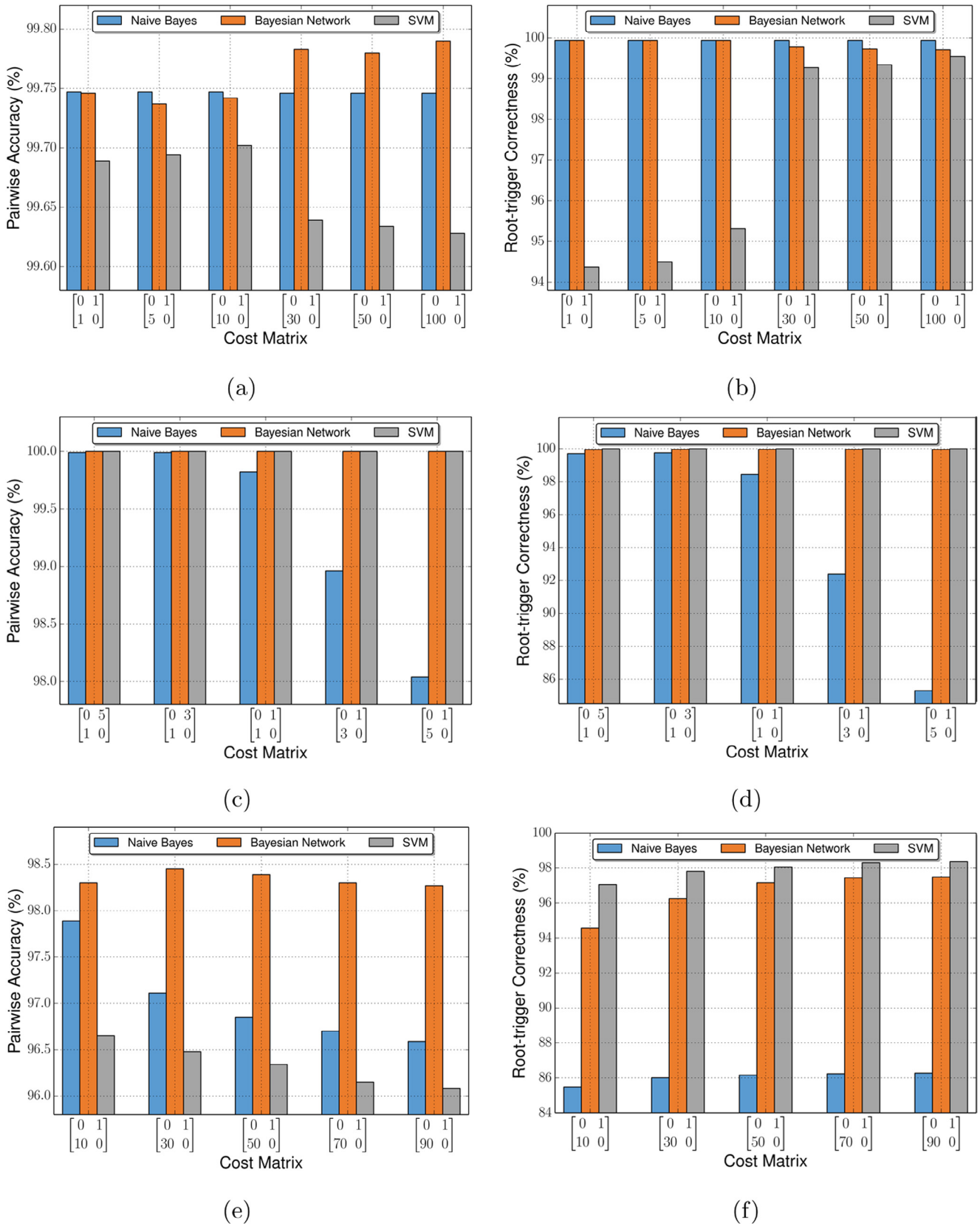

Fig. 5 – Accuracy and correctness results under various cost matrix conditions for datasets I, II and III. The results of pairwise accuracy are shown in (a, c, e). The results of root-trigger correctness are shown in (b, d, f).

Table 6 – Correctness of root triggers in Dataset I.

	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%

Note: Cases (a–g) refer to the various predicted root-trigger outcomes in Fig. 4 in Section 5.

6.2.3. *Evaluation on dataset II*

The goal of the experiment on dataset II is to find the triggering relation in traffic with mixed types, such as DNS and HTTP requests.

6.2.3.1. *Pairwise classification accuracy*. The pairwise classification results on dataset II are presented in Table 5. All three methods give high pairwise classification accuracy rates, confirming our method's ability to discover 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 Fig. 5c. 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.

6.2.3.2. *Correctness of root triggers*. We analyze the root-trigger accuracy for dataset II, and show the results in Fig. 5d. The root-trigger accuracy is high when using all three classifiers, with Bayesian network and SVM outperform the naive Bayes. We report the root-trigger correctness results under the cost matrix of $\begin{bmatrix} 0, & 1 \\ 1, & 0 \end{bmatrix}$ in Table 7.

6.2.4. *Evaluation on dataset III*

Dataset III (DARPA dataset) contains the network traffic collected from multiple subnets. We focus on three hosts that are compromised by attackers and report the results by weighting the number of requests on each host.

6.2.4.1. *Pairwise classification accuracy*. Results in Table 5 show a high prediction accuracy rate for pairwise triggering relations. The precision results are all above 0.99, while the recalls vary. By investigating the dataset, we found that the major reason of false positives is due to the change of protocols in one TCP session. Packets that have triggering relations with others usually share the same network protocol. However, in some rare cases, a packet using protocol A may trigger the another one using protocol B, especially when protocol A is a general one (e.g., TCP) and protocol B is a specific one (e.g., TELNET, SMTP and RSH protocols). A possible solution is to use features (in a vector) to characterize the change of protocols in a finer granularity, so that the classifiers can learn the patterns.

According to Fig. 5e, the Bayesian network classifier yields the highest pairwise accuracy among the three tools and it achieves the best accuracy rate using the cost matrix $\begin{bmatrix} 0, & 1 \\ 30, & 0 \end{bmatrix}$. For the two other classifiers, the pairwise accuracy results decline as the penalty of false negatives goes larger.

6.2.4.2. *Correctness of root triggers*. We run the root finding algorithm on dataset III to infer the root-triggers for each packet. Results show that the Bayesian network and SVM achieve better accuracy in finding the root-triggers, while the naive Bayesian classifier does not present a good prediction result, which is mainly due to the dependence of features in dataset III. Shown in Fig. 5f, the high root-trigger correctness results occur when using the cost matrices $\begin{bmatrix} 0, & 1 \\ 50, & 0 \end{bmatrix}$, $\begin{bmatrix} 0, & 1 \\ 70, & 0 \end{bmatrix}$ and $\begin{bmatrix} 0, & 1 \\ 90, & 0 \end{bmatrix}$. Further, we find that the results of the root-trigger prediction are not consistent with the pairwise accuracy results. In other words, the low pairwise accuracy results do not affect the result of root-trigger prediction, especially when increasing the penalty of false negatives in cost matrices. The reason is that the results of pairwise classification may be redundant or inaccurate, regarding the inference of root-triggers. Therefore, as long as an event can be traced back to its root according to the predicted classification results, the classifier is accurate regarding the root-trigger correctness.

Table 7 – Correctness of root triggers on Dataset II.

	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%

Note: Cases (a–g) refer to the various predicted root-trigger outcomes in Fig. 4 in Section 5.

We further introduce a new policy for detecting the malicious root-triggers, after finding the root-triggers using Algorithm 3. The policy is a heuristic one and based on our observations on dataset III. We highlight the high-risk protocols based on their frequencies and patterns of occurrence. In this policy, we found that network packets via Portmap protocol (port = 111) immediately followed (<0.1 s) by the Sadmind protocol (port = 32773) are malicious, which can be regarded as the period of infection. Besides, we discovered that the Portmap and Sadmind protocols occurred three times in a short period (<10 s) and the packets from an external IP (202.77.162.213) after each period are related to malicious activities (e.g., remote buffer overflow for rooting shell). This experiment suggests that data-specific security policies are more effective than general purpose ones. We show our model could be integrated with refined policies to detect malicious activities in the early age.

Algorithm 3 Root Finding Algorithm (RFA)

Input: An event e_k and $\mathbb{P}^* = \{(e_i \rightarrow e_j)\}$.
Output: A set $\mathbb{R}$, where each in $\mathbb{R}$ is a root of e_k.

```
1: define a set ℝ to store the results
2: define a queue Q and enqueue e_k onto Q
3: while Q ≠ ∅ do
4:    event n ← dequeue Q
5:    set 𝕋 ← find n's parent(s) based on ℙ*
6:    for each event e ∈ 𝕋 do
7:       if e is of type root then
8:          ℝ ← ℝ ∪ {e}
9:       else if e ∉ Q then
10:         enqueue e onto Q
11:      end if
12:   end for
13: end while
14: return ℝ
```

6.2.5. *Evaluation on dataset IV*

For dataset IV, the goal of the experiment is to find the triggering relationship on a large scale of data. The dataset contains both inbound and outbound TCP packets. The precision and recall results are given in Table 5. All three classifiers yield high pairwise accuracy rates, with the Bayesian network (99.72%) and SVM (99.82%) outperforming the naive Bayes classifier (98.92%). Performance results are reported in Section 6.4.

6.3. *Evaluation of stealthy malware activities*

To test the effectiveness of our solution, we evaluate several pieces of proof-of-concept and real-world malware. Results show that the all HTTP-based malware and DNS bots in our experiments are detected without triggering any false positives or false negatives.

6.3.1. *Malicious browser extension*

We write a proof-of-concept malicious Firefox extension, which is a piece of password-stealing spyware. The malware 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 triggering relation model and root-trigger policy, all malicious HTTP requests are detected. However, the default Windows Firewall does not alert the data leaks.

6.3.2. *Data exfiltrating malware*

We write another proof-of-concept data-exfiltrating malware. This malware runs as a stand-alone process, similar to the Pony bot (Pony botnet). 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 approaches successfully detect the network activities of the malware in that the malicious outbound requests do not have valid triggering relations, i.e., the requests lack of any user event as the root-trigger.

6.3.3. *Detection of real-world malware*

We find and investigate several pieces of real world malware.[2] We obtain the malicious requests by running the malware or synthesizing the traffic on a controlled virtual machine. To evaluate the capability of our traffic dependence analysis, we overlay the malicious traces to the normal traffic in dataset I. The malicious software is summarized below.

- Apache Qpid 0.30 Vulnerability: Remote attackers can trigger outgoing HTTP connections by a crafted message, due to the vulnerability in its XML exchange functionality. (CVE-2014-3629)

[2] Most malware samples are found at packetstormsecurity.com.

Table 8 – Percentages of triggering relations inferred by different subroutines of rule-based TRD on Dataset I.

Category	Triggering relation discovery (TRD)				TR not found
	IsRoot	IsChild	IsSibling	Total	
Percentage	1.9%	87.4%	8.6%	97.9%	2.1%

- Microsoft CryptoAPI Design Bug: Attackers can trigger HTTP requests due to a design bug in X.509 certificate chain validation. (CVE-2013-3870)
- Zend Framework vulnerability: It allows attackers to open files and trigger HTTP requests to leak information, due to the misuse of the PHP XML parser. (CVE-2012-5657)
- Linux/Cdorked.A: The servers affected by this backdoor redirect clients to a malicious website hosting a Blackhole exploit kit. After a series of redirects, a piece of malware is downloaded on the victim's computer via a GET request.
- MorXBrute Password Cracker: It is an HTTP dictionary-based password cracking tool. It supports users to customize their payloads to any specific HTTP software or websites (a password cracker).

All the network activities due to malware or software bugs are detected as vagabonds. These packets are either GET or POST requests. We run our solution on this dataset that the benign requests are interleaved with the malicious ones. Our solution can successfully detect the malicious requests, as they are not associated with any legitimate user's event or benign traffic, per the *root-trigger* security policy.

6.3.3.1. *Comparison to conventional firewall.* To detect the malware, most current solutions adopt the signature-based scanning, which requires the known of the malicious signatures. In our experiment, we collected the malicious/unauthorized network requests on a controlled virtual machine. We note that only a small portion (<9%) of requests sent to malicious host is reported by the Windows Firewall. These requests are triggered due to the server backdoor vulnerability (e.g., Linux/Cdorked.A). Most malicious requests are not reported and blocked. Therefore, we speculate that the blacklist in Windows Firewall is limited and the malicious domains are ever-changing. In our solution, we detect the malware's behaviors regardless the source code and functionality of the malicious software. Therefore, our solution is good to identify zero-day attacks that trigger unintentional network requests.

6.3.4. *DNS bot detection*

Botnet command and control channel using DNS tunneling (DNScat) is extremely stealthy and difficult to detect (Xu et al., 2013). 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., d1js21szq85hyn.cloudfront.net. We overlay the bot queries with a 2-hour DNS-HTTP traffic data (from dataset II), and then analyzed using our learning-based solution. Our evaluation confirms that our method successfully recognizes all the bot DNS queries as anomalies. These DNS queries do not have valid user-event root triggers.

6.4. *Comparison between rule-based and learning-based approaches*

The rule-based approach needs non-trivial human efforts to generate rules and algorithms. To compare with the results of our learning-based approach in Section 6.2.1, we present our evaluation of the rule-based solution on the dataset I. We infer the triggering relations by running the TRD algorithm (Algorithm 1) as a baseline method.

We calculate the percentage of requests whose triggering relations are inferred by IsRoot, IsChild, and IsSibling, respectively. The results in Table 8 show that most of the dependent relations (87.4%) are inferred by the IsChild procedure. Because the heuristic rules are generated based on the definition of *triggering relation*, the relations found by the TRD algorithm are all correct.

We inspected the requests that are missing triggering relation (2.1% in Table 8) based on the ground truth. We find that the precision of finding vagabond is about 60%, while the recall is 100%. The low precision is mainly due to the failure of finding triggering relations for some corner cases on the diverse and complex HTTP traffic data (e.g., heavy use of AJAX technique results in late-arriving packets). The recall is 100% because all malicious requests are flagged vagabonds by our TRD algorithm.

The root-trigger correctness of rule-based approach is 98.72%, which is less than that of the learning-based one (99.94% in Table 6). According to the rule-based TRD, most wrong root-trigger cases are due to the missing triggering relation for corners cases. One can always obtain better results by a hand-tuned algorithm, but this requires significant efforts from the domain experts. In contrast, the learning-based approach enables the triggering relation inference by using the *pairwise* features, which can precisely describe the corner cases from multiple dimensions.

6.4.1. *Performance of rule- and learning-based approaches*

All runtime results are obtained on a machine with Intel i5-3320 and 16 GB RAM. We measure the runtime on the dataset I to compare the performance for inferring the root-trigger on 1000 requests. Results show that it takes 2.68 s to run TRD algorithm per 1000 requests, while it takes 0.30 s using a Bayesian network classifier and the root-trigger policy.

6.4.2. *Performance breakdown of learning-based approaches*

We further investigate the performance of all processing operations across three classifiers. For each dataset, we report the runtime of Pairing, Training, Classification, and the root finding algorithm. The runtime of each operation is averaged from five runs and reported in Table 9. Standard deviations are negligible and not shown.

Shown in Table 9, the Training, Classification, and root finding algorithm are fast. The Pairing operation is the most time-

Table 9 – Averaged performance (in seconds) of pairing, training, classification, and root finding algorithm in the learning-based approach.

Data	Pairing	Training			Classification			Root Finding
		NB	BN	SVM	NB	BN	SVM	
I	965	0.8	1.9	5.9	15.9	14.8	13.2	1.6
II	603	1.7	2.1	10.4	3.2	2.8	5.1	0.6
III	3218	14.8	16.0	39.7	225.5	230.0	268.0	18.9
IV	7633	16.3	19.8	245.6	411.9	402.6	426.9	–

Note: NB and BN stand for naive Bayes and Bayesian network classifiers, respectively. Pairing time includes the pairwise feature extraction.

consuming task in our solution. To extract the pairwise features on 42 days of a server's TCP data (dataset IV), it takes as long as 2 hours to generate the pairs from 3 million TCP messages. Our experiments have determined that on a single day, at most 200 MB 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 3 minutes on average, indicating that our method is efficient enough for practical use.

6.5. *Summary*

We summarize our findings below.

- Bayesian network (BN) classifier outperforms the naive Bayes one, indicating the existence of dependencies in pairwise features. Both BN classifier and SVM yield high pairwise accuracy rates on all datasets, while SVM has higher runtime overhead than BN in general.
- 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. The detection accuracy can be improved by strategically defining the cost matrix.
- Our triggering relation analysis successfully reveals all the outbound traffic to 36 malicious domains, i.e., with zero false negative rate. Our solution also detects the stealthy network activities from several HTTP-based malware and DNS bots.
- The rule-based algorithm is inferior to the machine learning classification, in that the results show a low precision rate in finding vagabond requests. The coverage of various cases in TRD algorithm is crucial to determine the triggering relations in complex scenarios, so one needs manual efforts to generate a well-designed algorithm to obtain equivalent good results as the learning-based method does.
- With the proof of evaluations on datasets III and IV, our triggering relation model can be generalized to the events on the transport layer. The rationale of triggering relationship for a particular type of events determines the pairwise features. In the DARPA data (dataset III), we show the potential use of our model with other mining approaches/protocols for analyzing complex cases.

7. Related work

There exist solutions for discovering application or service dependencies for management and reliability purposes (Bahl et al., 2007; Chen et al., 2008; Kandula et al., 2008; Keller et al., 2000; King et al., 2005; Natarajan et al., 2012; Zand et al.). This line of research differs from our work in two main aspects. Dependency in these models refers to the reliance on services provided by others, not the triggering relation. Furthermore, our request-level triggering relations is more fine-grained than service- or application-level dependencies.

ReSurf (Xie et al., 2013), as the closest related work, aims at reconstructing web surfing activities from traffic traces via an analysis of request headers. The heuristic in ReSurf is a referrer-based approach. Our rule-based approach differs from theirs in two aspects. First, our proposed TRG is a more accurate model as we identify the triggering relations between any two web requests, while ReSurf does not tell the relations when a web object (e.g., an advertisement request) is commonly triggered from different root requests. Second, ReSurf identifies the head HTTP request by heuristic rules, while our solution precisely points out the triggering relations between user event and its triggered request by a browser extension, which guarantees the accuracy of our root-trigger policy. BINDER (Cui et al., 2005) detects break-ins on personal computers through analyzing the dependency of network events based on temporal and process information. BINDER processes the network events without inspecting its content, while our methods can semantically analyze and infer the triggering relation of network data. Our work describes a more specific user intention-based policy that supports application-specific dependence analysis with a much finer granularity.

The research on the interplay between human behaviors and system properties has been studied in the context of anomaly detection. ClickMiner (Neasbitt et al., 2014) is proposed to reconstruct user–browser interactions from network traces by actively replaying the recorded HTTP traffic within an instrumented browser. Unlike our model, ClickMiner focuses on user–browser interactions that cause the browser to initiate an HTTP request for a new web pages; therefore, it builds the referrer graph by pruning away the automatically generated requests. Besides, the applications of ClickMiner include aiding the forensic analysis of network incidents and identifying the malicious download, while our solution aims at detecting general malware activities that are not attributed to user's interactions. Not-A-Bot (Gummadi et al., 2009) is a system for authenticating traffic-generating user inputs such as mouse clicks on hyperlinks. It can be used for defeating DDoS attacks as well as click fraud. However, it does not analyze the triggering relations among network packets for anomaly detection as our solution does. In our root-trigger security policy, all application level requests should be attributed to legitimate user events, which distinguishes ours from their solution.

Previous studies proposed tools (e.g., WebTap (Borders and Prakash, 2004), WebShield (Li et al., 2010), SpyProxy (Moshchuk et al., 2007)) to ensure the web security and detect HTTP-based malware. WebTap (Borders and Prakash, 2004) is a tool to identify anomalies by identifying changes and deviations in aggregated flows patterns in terms of usages with statistical metrics. Both SpyProxy and WebShield proposed the execution-based web content analysis, so the web contents can be tested before reaching to user's browser. Their solutions are used to detect server-side malicious web content, whereas ours is focused on client-side anomalies. Other host-based anomaly detection solutions (Babić et al., 2011; Kolbitsch et al., 2009) focus on learning and generalizing from observed malware behaviors, for example, Babic et al. analyzed the malware behavior using tree automata inference in (Babić et al., 2011), Kolbitsch et al. built a behavior graph to tackle the malware's obfuscation and polymorphic in (Kolbitsch et al., 2009). Our work substantially differs from theirs, as we do not analyze malware on the system level, but pinpoint the stealthy malware's communication by isolating it from the normal one.

Machine learning approaches have been widely adopted in the security literature, since the work by Lee et al. (1999). Using learning-based approaches to classifying the network traffic and detecting threats has been reported in Bilge et al. (2011), Erman et al. (2007), Moore and Zuev (2005), Nguyen and Armitage (2008), Williams et al. (2006), Xie et al. (2010), and Zomlot et al. (2013). Compared to these aforementioned solutions, our approach utilizes classifiers to infer the *relations* between events, while theirs classify the events on an individual basis. The uniqueness of our work is the ability to automatically extract and recognize directional relations and structures.

Our triggering relation discovery problem may bear superficial similarities to the link prediction problem in the context of mining social network data (Backstrom and Leskovec, 2011; Getoor and Diehl, 2005; Kahanda and Neville, 2009; Liben-Nowell and Kleinberg, 2007). Getoor and Diehl (2005) surveyed the link mining problem and pointed out the sparsity in linked data. Liben-Nowell and Kleinberg (2007) formalized the link prediction problem and surveyed an array of methods on measuring the proximity of nodes in a network. Follow-up research applied advanced machine learning methods to social network data. These advanced methods include logistic regression, decision tree, and naive Bayesian (Kahanda and Neville, 2009) as well as supervised random walks (Backstrom and Leskovec, 2011). 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. However 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. To detect the ever-changing malware, our model can be further extended by applying more sophisticated security policies.

8. Conclusions

We introduced the problem of triggering relation discovery in network traffic and described its application in solving challenging network security problems, such as stealthy malware detection. We designed, developed, and compared both rule- and learning-based approaches for triggering relation discovery. Our evaluation on 10 + GB data (real-world and DARPA datasets) showed a high accuracy of the triggering relation prediction using the learning-based classification. Experimental results confirm the effectiveness of our traffic-reasoning technique against browser spyware, DNS bot, and data exfiltrating malware.

REFERENCES

Babić D, Reynaud D, Song D. Malware analysis with tree automata inference. In Computer Aided Verification. 2011. p. 116–31, 20.

Backstrom L, Leskovec J. Supervised random walks: predicting and recommending links in social networks. In Proceedings of the fourth ACM international conference on Web search and data mining. 2011. p. 635–44, 20, 21.

Baeza-Yates R, Ribeiro-Neto B, et al. Modern information retrieval, vol. 463. New York: ACM press; 1999. p. 13.

Bahl PV, Chandra R, Greenberg A, Kandula S, Maltz D, Zhang M. Towards highly reliable enterprise network services via inference of multi-level dependencies. In Proceedings of ACM SIGCOMM, August 2007. 19.

Bilge L, Kirda E, Kruegel C, Balduzzi M. EXPOSURE: finding malicious domains using passive DNS analysis. In Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS). February 2011. 20.

Borders K, Prakash A. Web Tap: detecting covert web traffic. In Proceedings of the 11th ACM Conference on Computer and Communication Security. 2004. p. 110–20, 20.

Chen X, Zhang M, Mao ZM, Bahl P. Automating network application dependency discovery: experiences, limitations, and new solutions. In Proceedings of OSDI. 2008. p. 117–30, USENIX Association. 19.

Cortes C, Vapnik V. Support-vector networks. Mach Learn 1995;20(3):273–97. 10.

Cova M, Kruegel C, Vigna G. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In Proceedings of 19th International World Wide Web Conference, 2010. 2.

Cui W, Katz YH, Tan WT. BINDER: an extrusion-based break-in detector for personal computers. In Proceedings: USENIX Annual Technical Conference. 2005. p. 4, 2, 20.

DNScat. A tool to tunnel traffic through DNS servers. http://tadek.pietraszek.org/projects/DNScat/. 2004. 18.

Elkan C. The foundations of cost-sensitive learning. In International Joint Conference on Artificial Intelligence, volume 17. 2001. p. 973–8, 10.

Erman J, Mahanti A, Aritt MF, Cohen I, Williamson CL. Semi-supervised network traffic classification. In Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2007, San Diego, California, USA, June 12–16, 2007. 2007. p. 369–70, 20.

Getoor L, Diehl CP. Link mining: a survey. SIGKDD Explor Newsl 2005;7(2):3–12. 20.

Green TM, Ribarsky W, Fisher B. Visual analytics for complex concepts using a human cognition model. In Visual Analytics Science and Technology, 2008. VAST'08. IEEE Symposium on. 2008. p. 91–8, 2.
Gu G, Perdisci R, Zhang J, Lee W. BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In Proceedings of the 17th USENIX Security Symposium. 2008. 2.
Gummadi R, Balakrishnan H, Maniatis P, Ratnasamy S. 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.
John G, Langley P. Estimating continuous distributions in Bayesian classifiers. In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. 1995. p. 338–45, 10.
Kahanda I, Neville J. 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. 20, 21.
Kandula S, Chandra R, Katabi D. What's going on? Learning communication rules in edge networks. In Proceedings of ACM SIGCOMM. August 2008. 19.
Keller A, Blumenthal U, Kar G. Classification and computation of dependencies for distributed management. In Proceedings of International Symposium on Computers and Communications. 2000. p. 78–83, 19.
King ST, Mao ZM, Lucchetti DG, Chen PM. Enriching intrusion alerts through multi-host causality. In Proceedings of Network and Distributed System Security (NDSS). 2005. 2, 19.
Kolbitsch C, Comparetti PM, Kruegel C, Kirda E, Zhou X, Wang X. Effective and efficient malware detection at the end host. In USENIX Security Symposium. 2009. p. 351–66, 20.
Lee W, Stolfo SJ, Mok KW. A data mining framework for building intrusion detection models. In Proceedings of IEEE Symposium on Security and Privacy. 1999. p. 120–32, 20.
Li Z, Zhang M, Zhu Z, Chen Y, Greenberg AG, Wang Y-M. WebProphet: automating performance prediction for web services. In NSDI, volume 10. 2010. 20.
Liben-Nowell D, Kleinberg J. The link-prediction problem for social networks. J Am Soc Inf Sci Tec 2007;58(7):1019–31. 20, 21.
MIT Lincoln Laboratory. Darpa intrusion detection evaluation. https://www.ll.mit.edu/ideval/data/2000/LLS_DDOS_1.0.html. 2000. 12.
Moore AW, Zuev D. Internet traffic classification using Bayesian analysis techniques. In Proceedings of the International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS 2005, June 6–10, 2005, Banff, Alberta, Canada. 2005. p. 50–60, 20.
Moshchuk A, Bragin T, Deville D. SpyProxy: execution-based detection of malicious web content. In Proceedings of the 16th USENIX Security Symposium. 2007. 20.
Natarajan A, Ning P, Liu Y, Jajodia S, Hutchinson SE. NSDMiner: automated discovery of network service dependencies. In INFOCOM. 2012. p. 2507–15, 2, 5, 19.
Neasbitt C, Perdisci R, Li K, Nelms T. ClickMiner: towards forensic reconstruction of user-browser interactions from network traces. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. 2014. p. 1244–55, 20.
Nguyen TTT, Armitage GJ. A survey of techniques for Internet traffic classification using machine learning. IEEE Commun Surv Tut 2008;10(1–4):56–76. 20.
Tlogger. An Firefox extension. http://dubroy.com/tlogger/. 2009. 11, 24.
Williams N, Zander S, Armitage G. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. SIGCOMM Comput Commun Rev 2006;36(5):5–16. 20.
Xie G, Iliofotou M, Karagiannis T, Faloutsos M, Jin Y. ReSurf: reconstructing web-surfing activity from network traffic. In IFIP Networking Conference, 2013, Brooklyn, New York, USA, 22–24 May, 2013. 2013. p. 1–9, 19.
Xie P, Li JH, Ou X, Liu P, Levy R. Using Bayesian networks for cyber security analysis. In Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on. 2010. p. 211–20, 20.
Xu K, Butler P, Saha S, Yao D. DNS for massive-scale command and control. IEEE Trans Dependable Sec Comput 2013;10(3):143–53. 5, 12, 18.
Zand A., Vigna G., Kemmerer R., Kruegel C. Rippler: delay injection for service dependency detection. In INFOCOM'14. 2014. p. 2157–65. 2, 5, 19.
Zhang H, Banick W, Yao D, Ramakrishnan N. User intention-based traffic dependence analysis for anomaly detection. In Security and Privacy Workshops (SPW), 2012 IEEE Symposium on. 2012. p. 104–12, 1, 3, 6, 10.
Zhang H, Yao D, Ramakrishnan N. Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery. In Proceedings of the 9th ACM Symposium on Information, Computer, and Communication Security (ASIACCS'14), June 2014. 2.
Zhang H, Sun M, Yao D, North C. Visualizing traffic causality for analyzing network anomalies. In Proceedings of International Workshop on Security and Privacy Analytics (IWSPA). 2015. p. 37–42, 1, 3.
Zomlot L, Chandran S, Caragea D, Ou X. Aiding intrusion analysis using machine learning. In Machine Learning and Applications (ICMLA), 2013 12th International Conference on, volume 2. 2013. p. 40–7, 20.
TCPFLOW 1.3. https://github.com/simsong/tcpflow. 24.
Panda Security Report. http://press.pandasecurity.com/press-room/reports/. 2015. 2.
Pony botnet. Botnet pony 1.9 malware. http://laboratoriomalware.blogspot.com/2013/01/botnet-pony-19-malware.html. 2013. 17.

Hao Zhang received his Ph.D. degree in Computer Science from Virginia Tech in 2015. He was a member of the Human-Centric Security Laboratory directed by Professor Danfeng Yao. He received his M.S. degree in Computer Science from Villanova University, PA in 2010. He holds a U.S. patent on his network anomaly detection technology. His current research interest is on designing machine learning methods for network and mobile security.

Danfeng (Daphne) Yao is an associate professor and L-3 Faculty Fellow in the Department of Computer Science at Virginia Tech, Blacksburg. She received her Computer Science Ph.D. degree from Brown University in 2007. She received the NSF CAREER Award in 2010 for her work on human-behavior driven malware detection, and most recently ARO Young Investigator Award for her semantic reasoning for mission-oriented security work in 2014. She received the Outstanding New Assistant Professor Award from Virginia Tech College of Engineering in 2012. Dr. Yao has several Best Paper Awards (e.g., ICICS '06, CollaborateCom '09, and ICNP '12) and Best Poster Awards (e.g., ACM CODASPY '15). She was given the Award for Technological Innovation from Brown University in 2006. She held a U.S. patent for her anomaly detection technologies. Dr. Yao is an associate editor of IEEE Transactions on Dependable and Secure Computing (TDSC). She serves as PC members in numerous computer security conferences, including ACM CCS. She has over 65 peer-reviewed publications in major security and privacy conferences and journals.

Naren Ramakrishnan is the Thomas L. Phillips Professor of Engineering at Virginia Tech. He directs the Discovery Analytics Center, a university-wide effort that brings together researchers from computer science, statistics, mathematics, and electrical and computer engineering to tackle knowledge discovery problems in important areas of national interest, including intelligence analysis, sustainability, and electronic medical records. He received his PhD in computer sciences from Purdue University.

Zhibin Zhang is an associate professor at Institute of Computing Technology, Chinese Academy of Sciences. He received his Ph.D. degree in Computer Science from Institute of Computing Technology, Chinese Academy of Sciences in 2007. His research interests lie in the area of network measurement and security, traffic classification, distributed system and machine learning.