Discovery of Triggering Relations and Its Applications in Network Security and Android Malware Detection

Hao Zhang

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

October 8, 2015
Blacksburg, Virginia

Keywords: Network Security, Stealthy Malware, Anomaly Detection

Discovery of Triggering Relations and Its Applications in Network Security and Android Malware Detection

Hao Zhang

ABSTRACT

An increasing variety of malware, including spyware, worms, and bots, threatens data confidentiality and system integrity on computing devices ranging from backend servers to mobile devices. To address these threats, exacerbated by dynamic network traffic patterns and growing volumes, network security has been undergoing major changes to improve accuracy and scalability in the security analysis techniques.

This dissertation addresses the problem of detecting the network anomalies on a single device by inferring the traffic dependence to ensure the root-triggers. In particular, we propose a dependence model for illustrating the network traffic causality. This model depicts the triggering relation of network requests, and thus can be used to reason about the occurrences of network events and pinpoint stealthy malware activities. The triggering relationships can be inferred by means of both rule-based and learning-based approaches. The rule-based approach originates from several heuristic algorithms based on the domain knowledge. The learning-based approach discovers the triggering relationship using a pairwise comparison operation that converts the requests into event pairs with comparable attributes. Machine learning classifiers predict the triggering relationship and further reason about the legitimacy of requests by enforcing their root-triggers. We apply our dependence model on the network traffic from a single host and a mobile device. Evaluated with real-world malware samples and synthetic attacks, our findings confirm that the traffic dependence model provides a significant source of semantic and contextual information that detects zero-day malicious applications.

This dissertation also studies the usability of visualizing the traffic causality for domain experts. We design and develop a tool with a visual locality property. It supports different levels of visual based querying and reasoning required for the sensemaking process on complex network data.

The significance of this dissertation research is in that it provides deep insights on the dependency of network requests, and leverages structural and semantic information, allowing us to reason about network behaviors and detect stealthy anomalies.

Acknowledgments

I owe my sincerest appreciation to my advisor, Dr. Danfeng Yao, for giving me invaluable advice not only on research but also on various life issues. During the past five years, she has spent countless hours to help me identify many important and interesting research topics, push ideas one level further, and sharpen my skills as a competent researcher. Her patience and trust gave me the confidence to overcome the hurdles in my research, and her high standards of scholarship inspired me to challenge myself much more than I would have otherwise.

I would like to express my gratitude to Dr. Wenjing Lou, Dr. Naren Ramakrishnan, Dr. Chris North, and Dr. Xinming Ou for serving on my committee. This dissertation would not have been possible without many collaborative efforts from Dr. Ramakrishnan and Dr. North. I am deeply grateful for their brilliant insights on data analysis, thoughtful discussions, and detailed comments on my work. I am also very grateful to Dr. Lou and Dr. Ou for taking the time to read my dissertation and help sharpen my work. Their early feedback to my papers and presentations was very helpful for me to finalize this dissertation. I would like to thank all committee members for being enthusiastic mentors.

Beyond direct guidance from the committee members, many people contributed to my graduate study and made Blacksburg an unforgettable and rewarding experience. I am indebted to Dr. Eli Tilevich for demonstrating and generously sharing his remarkable writing skills as well as inspiring me to do my best throughout my Ph.D study. Maoyuan Sun demonstrated hard work and clear thinking that I learned by collaborating with him. I have thoroughly enjoyed discussing with Xiaokui Shu and I am grateful to him for polishing my ideas in several projects. It has been a pleasure to get to know and to work with so many smart friends in our lab: Huijun Xiong, Hussain Almohri, Kui Xu, Xiaokui Shu, Karim Elish, Fang Liu and Ke Tian. I give my appreciation to Xiao Lin, Bo Li, and Jian Zhang for helping me go through many long days. I have been blessed with many friends along the way, and I am grateful for their friendship.

Last but not least, I am fortunate to have such loving, supporting, and encouraging parents. Since my childhood, they taught me the importance of being diligent, curious, optimistic, and confident. I would like to acknowledge the immeasurable contribution of my parents, whose love and sacrifice have enabled me to chase my dreams. This dissertation is dedicated to them with the utmost love.

Contents

Chapter 1

Introduction

Anomaly detection – a field pioneered by Denning [49] – defines, specifies, and enforces normal traffic and interaction patterns in a network or on a host. Anomalies or outliers refer to any activities that do not conform to regular behaviors. Statistical techniques modeled under specific domain knowledge have been proposed for anomaly detection [49, 74, 119, 123]. For example, dynamic Bayesian networks can be used to detect abnormal data access patterns by malicious insiders to a sensitive database [23]. However, achieving general anomaly detection is challenging, especially for complex and diverse behaviors involving activities spanning users, hosts, and networks.

In this dissertation, we describe a *triggering relation discovery* solution to construct the request-level causality structure in network traffic. There have not been systematic studies on the request level causal analysis for malware detection. Existing dependence analysis work focuses on the network services and is not designed for malware detection. For example, Orion [39] and NSD-Miner [105] addressed the problem of network service dependency for automatic manageability and network stability. Rippler [152] is proposed to actively perturb the network traffic to understand the dependencies between service and devices. In comparison, we propose to achieve the *request-level* causality structure of network traffic. This finer granularity (request vs. flow) requires different relation semantics and more scalable analysis methods.

In our solution, we aim at reasoning about the network incidents and identifying malicious requests by analyzing the dynamic behaviors of malware. Our goal is to profile the legitimacy of the traffic pattern of benign applications, and thus to *detect the malicious network requests that are sent without user's awareness*.

1.1 Motivation and Challenges

A study showed that a significant portion of computers worldwide is infected with malware conducting clandestine activities [13]. Meanwhile, recent advancements in information technology have raised concerns on the security risks posed by the prevalence of malicious apps on mobile devices. Malware, either on hosts or mobiles, may spy on the victim users, cause data exfiltration, and abuse the device 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 [45]. Once the infection is successful (e.g., zero-day exploits), virtually all malware activities require sending outbound network traffic from the infected machine, though network requests from advanced malware may not exhibit distinct communication patterns.

Determining whether or not networked hosts and devices are infected with stealthy malware is technically hard. Common network analysis methods typically follow two approaches: *i)* inspecting requests individually using pattern matching with known malicious signature strings, and *ii)* or performing statistical analysis (e.g., [69, 134]). Most commercial intrusion detection systems (IDS) focus on detecting the specific intrusive and malicious patterns. They work well if the attack signatures or behaviors are known or can be modeled *a priori*. However, modern attacks and stealthy malicious software (malware) evolve constantly, e.g., executables in Mariposa botnets change every 48 hours, making it difficult for intrusion detection systems to identify emerging and zero-day exploits. Therefore, the conventional patten-based scanning is limited, as it is challenging to obtain the signature for a zero-day exploit at its early stage. Frequency-based and counting-based statistical methods are not effective due to the low traffic volume of stealthy malware and diverse behaviors of malicious activities. When the definitions for normal behaviors are restrictive, false positives (i.e., false alarms) may be high; whereas broad definitions for normal behaviors may result in high false negatives.

Firewalls are introduced as the conventional means to prevent attacks that try to hide themselves and communicate with remote hosts. Most firewalls work on a per packet basis, by filtering the process ID, IP, port, protocol, and other attributes assigned by the applications. Therefore, if the application is compromised, then the malicious packets that contain forged information cannot be detected by firewalls. Besides, worms and Trojans are constantly changing their behaviors and activities. Hence, it is impossible for network administrators to define all rules on the firewalls.

Compared with the aforementioned security techniques, a more effective network security approach is to discover characteristic behavioral patterns in network event attributes and construct the causality structure in network traffic, e.g., [47, 68, 86]. For example, BINDER [47] detects anomalous network activities on personal computers through analyzing the correlation in network events by the temporal and process information. BotMiner [68] performs a correlation analysis across multiple hosts of a network for detecting similarly infected bots. King *et al.* [86] constructed directed graphs from logs to show network connections for dissecting attack sequences. However, none of these above solutions present a formulation of the traffic dependence on the request-level for security purposes.

Our solution generalizes the traffic dependence model on hosts and mobile devices. We explore direct and indirect dependencies in how a user interacts with applications and how applications respond to the user's requests following the specifications of the applications. The traffic dependence cannot be easily discovered from the raw logs. Our solution leverages the stateful information and builds story-telling scenarios to help people comprehensively understand the traffic and detect anomalies. We investigate both rule- and learning-based approaches to identify such relation.

Last, monitoring network traffic and detecting anomalies are essential tasks that are carried out routinely by security analysts. The sheer volume of network requests often makes it difficult to investigate attacks and pinpoint their causes. The missing gap between the information from data engine and human experts is huge, as they need to leverage and apply the domain knowledge on voluminous logs for making the security decisions.

1.2 Research Contributions

In this dissertation, we consider stealthy malware residing on a single device, which either is a stand-alone process or piggy-backing on a legitimate process such as a malicious browser extension. We assume that the malware is actively making network connections for attacks (i.e., command and control). Our work aims to demonstrate the feasibility of traffic dependence analysis for detecting these suspicious network activities.

We formalize the problem of *triggering relation discovery* in network requests, the structure and presentation of *triggering relation graph*, and their applications for detecting stealthy malware activities. The causality analysis of network requests enhances the understanding of the context-aware security. We describe a *root-trigger* security policy, which is used for detecting unauthorized network traffic from devices. Our dependence model has advantages over conventional pattern-based solutions (e.g. [38, 41, 131]), because it does not require *a priori* knowledge or assumptions about the normal data patterns.

We demonstrate with experiments that discovering fine-grained causality in network traffic is feasible. We extensively evaluate and compare the detection accuracy of the rule- and learning-based approaches with DARPA dataset and real-world network data, including HTTP, DNS, and TCP traffic. Both approaches are proved to be effective in detecting stealthy malware activities, with the learning-based method yielding better results than rule-based one.

The contributions of my dissertation are summarized as follows.

1. We introduce the problem of *triggering relation discovery* in network traffic. The new definition of *triggering relationship* between user actions and network events is given in the security context. We describe how to construct a triggering relation graph as a dependence analysis model and demonstrate its application in solving challenging network security problems, such as stealthy malware detection.

2. We present a rule-based approach for discovering the triggering relations on network data. The discovery algorithm is generated based on empirically derived rules. It inspects the temporal, semantic, and process-related attributes to reveal the causal relations among network requests. Our proposed traffic dependence solution cannot be realized by the conventional (stateful) firewall, because the inference of triggering relation requires complex algorithmic computation on system events beyond the simple rule-based filtering.

3. We describe an advanced learning-based approach for efficiently discovering the underlying triggering relations on a large scale of network events. A new feature extraction method is introduced as the *pairing* operation. It performs pairwise attribute comparisons, enabling the use of machine learning classifiers for the triggering relation discovery.

4. 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. We also propose to distinguish the root-triggers using dependency features extracted from the dependence graph. The features, revealing the patterns of automatically-generated benign, are unique and enables the prediction of malicious activities at a high accuracy.

5. We develop a visualization tool for security analysts to efficiently display the network traffic dependency. This tool has a *visual locality* feature that can optimize the displaying of structured data. The visual representation is a *radial* layout based on a curved timeline display, which maximizes the use of the screen and enables the high visual locality. We conduct a user study that confirms our design can enhance the readability and perceptibility of the network traffic causality.

1.3 Organization

The rest of the dissertation is organized as follows. In Chapter 2, we survey the related work and existing solutions for detecting malicious activities. Then, we introduce our dependence analysis model and its security applications in Chapter 3. Our solution for the host-based anomaly detection is given in Chapter 4. We further extend the solution to address the new challenges on mobile devices in Chapter 5. We present a visualization tool for security analysts to efficiently analyze the traffic dependence in Chapter 6. Chapter 7 concludes our work and discusses the future directions.

Chapter 2

Literature Review

This chapter describes the background of our research work, surveys the literature, and gives an overview of how our work fits in. We divide our discussions of related literature into four domains: discovering and enforcing network traffic dependence (§2.1), machine learning approaches to detecting network anomalies (§2.2), Android network traffic analysis for detecting malicious apps (§2.3), and visual analytics solutions for security forensics (§2.4).

2.1 Discovering and Enforcing Network Traffic Dependence

We compare our work with the existing solutions on enforcing or discovering dependencies [47, 61, 86, 110, 119, 142], user intention-based security [71, 121, 149], web security [33, 34, 91, 102, 118, 132], as well as general malware detection techniques [38, 41, 127]. We summarize the selected literature in Table 2.1. The table is not intended to be exhaustive and the papers are not mutually exclusive in each topic.

The dependence analysis provides an effective way to pinpoint the origins of the vulnerability, reason about the complex structures, and maintain the service reliability. There exist solutions for inferring the dependence among data flow [119], intrusion detection logs [68, 86], and network service [28, 39, 83, 85, 86, 105, 152]. However, the dependence analysis on network requests/packets has not been well studied, with a few exceptions [47, 92, 141].

BINDER [47] is a host-based solution that detects break-ins on personal computers through analyzing the dependency of network packets based on temporal and process information. The detection algorithm in BINDER considers three kinds of delays, and thus to identify the temporal relations of the network packets. Though both BINDER and our rule-based solution correlate the outgoing network traffic and process information with user activities, we further semantically infers the

Table 2.1: Selected literature on network traffic dependence analysis.

Major Domain	Paper	Topic
Dependence Analysis	Orion [39]	Network application dependency discovery.
	BINDER [47]	Host-based break-ins detection.
	LogMaster [61]	General-purpose event correlation.
	Kagal *et al.* [80]	Dependence analysis for policy management.
	King *et al.* [86]	Traffic dependency on IDS (Snort) logs.
	WebProphet [92]	Dependence analysis of web for performance.
	NSDMiner [105]	Dependence analysis of network service.
	Rippler [152]	Traffic watermarking for analyzing services.
User Intention based Security	Not-A-Bot [71]	Authenticating traffic-generating user inputs.
	Gyrus [77]	Verifying user intents from a UI.
	BLADE [98]	Analysis of user events and file creation.
	WebCapsule [106]	Analysis of user events and web sessions.
	ClickMiner [107]	Reconstructing user-browser interactions.
	WebWitness [108]	Analysis of the malware download path.
	UIBAC [121]	Analysis of user events and program events.
	DeWare [145]	Analysis of user events and file system.
Web Security	Borders *et al.* [33, 34]	Identify anomalous HTTP traffic.
	WebShield [91]	A proxy for testing malicious JavaScript.
	SpyProxy [102]	Execution-based web content analysis.
	Doppelganger [118]	Enforce cookie policies.
	Louw *et al.* [132]	Ensure extensions on broswers.

dependence of network packets. The dependency inference algorithm in our solution considers the semantics and contextual information from the request headers, which makes ours beyond the temporal analysis of network events. For example, BINDER treats network events equally without inspecting their contents. Consequently, it does not defend against piggybacking attacks. However, we describe our rule-based approach as a more powerful solution that supports dependence analysis in a much finer granularity.

WebProphet [92], developed by Li *et al.*, extracts the dependencies of web requests to diagnose the performance bottlenecks in backend infrastructures. The assumption of WebProphet is that the performance of loading a webpage depends on the download time of embedded objects. Therefore, the authors utilize WebProphet to simulate the page load process of a browser and extract the dependence of web objects. Our work differs from theirs, as we focus on the inference of dependency on application layer requests and detection of anomalous network events. Besides, the parental dependency graph (PDG) in [92] is to decompose the components of one web page. However, our model is used to describe the dependencies of network requests over a long period, which enables us to build story-telling scenarios and detect anomalies.

ReSurf [141], as one of the closest related solutions, 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, the triggering relation model accurately identifies the dependency between any two web requests. However, ReSurf does not tell the relations when a web object (e.g., an advertisement request) is commonly triggered by different root requests. Second, ReSurf identifies the root requests using heuristic rules. In our solution, we precisely point out the triggering relations between user event and its triggered request by correlating system level information and network data, which guarantees the accuracy of our root-trigger policy.

In SpiderWeb [129], the authors proposed to create redirection graphs by aggregating redirection chains that lead to a specific webpage. The features extracted from the redirection graph and visiting users are then used to classify the webpage as malicious or benign.

Other existing dependence analysis work (e.g., [39, 105, 152]) is on network service level and is not designed for malware detection. For example, Orion [39] and NSDMiner [105] addressed the problem of network application/service dependency for network stability and automatic manageability. Rippler [152] presents a solution to obtain the dependencies between devices and services by actively perturbing the network traffic. The authors propose to introduce delays into network flows and evaluate how other service reacts using statistical tests. In [29], the authors inject watermark signatures into the network flow to identify co-residency of virtual machine instances. In general, correlating events to find dependency is a commonly used approach in detecting attacks (e.g., [36, 60, 86]). Unlike the work mentioned above, we aim to achieve the request-level causality structure in network traffic.

User intention-based security. The research on the interplay between human behaviors and system properties has been studied in the context of anomaly detection. ClickMiner [107] is proposed to reconstruct user-browser interactions from network traces by actively replaying the recorded HTTP traffic within an instrumented browser. Unlike our triggering relation model, ClickMiner focuses on the user's events 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.

WebWitness [108] is proposed to trace back and label the sequence of events (e.g., visited web pages) preceding malware downloads. It leverages automatically labeled malware download paths to better understand the attack trends, especially how users reach attack pages on the web.

WebCapsule [106] is a lightweight forensic tool for analyzing the web browsing sessions. It records all user interactions, the rendered web contents, and runtime events/signals. It allows a forensic analyst to fully analyze the past web sessions. By *replaying* all non-deterministic inputs, WebCapsule enables the discovery of short-lived phishing and social engineering attack webpages.

Gyrus [77] is designed to verify whether a system's behavior matches the users' intention by confirming with users from a UI. The authors focus on the text-based network applications and use Gyrus to ensure the data integrity. Gyrus can secure these applications from malicious behaviors such as spam and fraud by allowing only outgoing traffic with the content that matches the user's intention. Our solution systematically differs from Gyrus, as ours infers the user intentions by analyzing the semantic between events, rather than explicitly asking users for validation.

Not-A-Bot [71] is a system for authenticating traffic-generating user inputs such as mouse clicks on hyperlinks. It is designed to defeat DDoS attacks as well as click fraud. However, it does not analyze the triggering relations among network requests for anomaly detection as our solution does. As explained in §4.5.2, our solution can use Not-A-Bot and similar techniques (e.g., [144]) to ensure the integrity of user inputs collected.

Besides the traffic dependence studied in this dissertation, the approach of user intention-based dependence analysis is widely adopted for anomaly detection in file-system events. The enforcement of dependencies between user actions and file-system events enables the detection of unauthorized file-system activities, such as download, read, or write. For example, DeWare [145], BLADE [98], and UIBAC [121] leverage user behaviors for certain file-access regulations. Our host-based solution is built for analyzing network events, as opposed to file-system events in above three solutions. Therefore, we address new technical challenges in this dissertation.

Web security. Previous studies proposed tools (e.g., [33, 92, 102]) to ensure the web security and detect HTTP-based malware. WebTap, developed by Borders and Prakash [33], is a tool to monitor the changes and deviations in aggregated flows patterns. It identifies anomalies in the HTTP traffic by evaluating the metrics such as request regularity, bandwidth usage, inter-request delay time, and transaction size. Besides, both WebShield [92] and SpyProxy [102] propose the execution-based web content analysis, so the web contents can be tested before reaching to user's browser.

Xiong *et al.* proposed to predict the outbound requests by parsing the web pages requested by users [142]. Their solution serves as a parallel universe to the browser, so that it can predict the legitimacy of outgoing requests in real-time. They report the anomalies when there are discrepancies in network events from the parallel universe and the browser. However, dynamic web components (e.g., JavaScript requests) are heavily used in modern websites and usually generated on-the-fly, which makes it impossible to predict accurately. Last, because their tool needs to fetch web objects independently from the browser, it doubles the bandwidth overhead.

Louw *et al.* [132] addressed the problem of malicious browser extensions. They propose to ensure the browser by allowing only extensions installed by users and detects unauthorized changes made to installed extensions. In our host-based solution, we sign and verify the outbound network requests by appending a keyed hash value, which guarantees the data integrity, so that no forged request can be sent out.

2.2 Machine Learning Approaches to Network Security

Machine learning approaches have been widely adopted in the security literature, since the work by Lee, Stolfo, and Mok [90]. Existing solutions of machine learning approaches to detecting threats are studied in the form of code analysis [45, 94], network traffic classification [56, 101, 109, 139], malicious domain classification [32, 99, 111], and intrusion/botnet analysis [69, 96, 162].

Sommer and Paxson [125] did pioneer work on employing machine learning in network intrusion detection, and they provided a set of guidelines meant to strengthen future research on anomaly detection. Machine learning approaches are often employed to reduce the labor burden of research scientists. For example, Zomlot *et al.* [162] aim at filling the semantic gap between Intrusion Detection System (IDS) outputs and malicious activities understood by users. They proposed a prediction model to classify the interesting IDS outputs that need further forensic analysis by domain experts.

Network traffic classification. Nguyen and Armitage surveyed on Internet traffic classification using machine learning methods in [109]. Their paper presents an overview of how machine learning techniques can be used in network security and motivates our work to infer the triggering relation of network traces in a much finer granularity.

Williams *et al.* [139] did an empirical study on summarizing the features from payload-independent features, such as packet length and inter-arrival time distributions. Their work [139] compares different machine learning algorithms to classify IP traffic flows, while ours desires to unveil the underlying causal relations of HTTP requests. Therefore, we focus on the semantic features extracted from the request headers, rather than the per-flow statistics.

Moore and Zuev [101] proposed to use the header-derived features to classify the Internet flow in different categories. Roughan *et al.* [114] used a statistical signature-based approach to classifying the IP traffic. Our work differs with theirs as the following reasons. *i)* The goals are different. We aim to discover the triggering relation among network requests, then to identify anomalies. Their solutions are used to cluster the Internet traffic, provide different Quality of Service, and thus optimize the usage of the existing network infrastructure. *ii)* They focus on the Internet traffic flows, while our scope is confined to HTTP requests. Therefore, our granularity is much finer and the resolving power is much stronger.

Erman *et al.* [55] used two unsupervised algorithms (namely K-means and DBSCAN) to cluster the traffic by transport layer statistics. Their evaluation indicates that their solution is better than the conventional port-based and payload-based analysis. Livadas *et al.* [96] proposed to use machine learning techniques to detect the IRC-based botnet traffic. Compared with these traffic clustering and classification solutions, the uniqueness of our triggering relation model and learning-based approach is the ability to automatically extract and recognize directional relations and structures.

Link prediction problem. Our triggering relation discovery problem may bear superficial similar to the link prediction problem in the context of mining social network data [26, 64, 81, 93]. Getoor and Diehl [64] surveyed the link mining problem and pointed out the sparsity in linked data. Liben-Nowel and Kleinbergz [93] formalized the link prediction problem and surveyed an array of methods for 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 [81] as well as supervised random walks [26]. Besides the obvious semantic differences in the two problems, our learning-based solution differs from the social network link prediction.

- Links in social networks connect nodes that are considered equivalent to 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 triggering relation modeling (§4.3 and §5.3) and root-trigger security analysis (§4.4) 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.

Malicious domain classification. Also related are studies that proposed classification methods for identifying the suspicious domains or phishing URLs [32, 62, 99, 138]. EXPOSURE [32] is designed to detect domains involved in malicious activities by conducting large-scale and passive DNS analysis at the network level. Authors extracted 15 temporal and DNS query-based features, and used machine learning classifier to identify the malicious domains. Ma *et al.* proposed to examine the lexical features of the URLs and features of the sites' hosting information to identify the malicious URLs [99]. Authors in [138] utilized page content-related features to detect phishing webpages.

Our detection approach systematically differs from theirs. The above solutions classify the events on an individual basis, while we utilize classifiers to infer the relations between events. Most features used in previous solutions can be categorized as *i)* time-based, *ii)* lexical and *iii)* host-based features. However, they do not include the dependency features for the security purpose as ours. Our study makes an important step towards addressing the significance of dependency knowledge in the detection of malicious domains and requests.

2.3 Android Network Traffic Analysis and Malware Detection

Existing work on protecting Android system and detecting malicious apps is studied in the form of static analysis [19, 24, 46, 63, 72, 95, 97, 112, 140, 148, 150, 153], taint analysis [37, 51, 65, 147, 151], and privilege control [130, 136, 159]. Most existing static analysis solutions take advantage of the APIs, system calls and bytecode for detecting malicious apps. For example, SCS-Droid [95] identifies malicious apps by extracting subsequences of system calls. ViewDroid [153] detects repackaging apps by analyzing the bytecode of Android apps. Drebin [24] extracts features including APIs, permissions, components to characterize and classify apps.

Dynamic analysis, as a complementary to the static analysis, monitors and detects the runtime behaviors of the apps. For example, Crowdroid [37] is a framework to dynamically analyze application behaviors. It collects the Android system traces from real users through crowdsourcing and processes the data at a central server. TaintDroid [51] monitors the sensitive data flow on the system-wide, but it cannot provide the insights of how the data are triggered from the user's perspective. AppIntent [151] considers the user-intended data transmission on Android and builds the missing leak between the data leaking and user's interactions However, AppIntent requires tremendous efforts in static taint analysis to preprocess and extract all possible data transmission paths, as it treats the apps as whitebox. SmartDroid [160] proposes a hybrid analysis method to identify UI-based event trigger conditions using the sensitive APIs. Our dependence analysis systematically differs from the aforementioned solutions, as ours identifies unauthorized network activities and detects malicious apps by analyzing the dependencies between user interactions and app's behaviors (e.g., network traffic).

App Guardian [158] proposes to temporally pause suspicious background process to prevent the potential data leaking. We do not solve the side channel attacks as App Guardian does, but our solution prevents the data exfiltration by identifying the suspicious outbound traffic. Droid-Scope [147] proposes to reconstruct the OS-level and Java-level semantics simultaneously and seamlessly. Therefore, it can be used to profile the API-level activities and track information leakage through both the Java and native components using taint analysis. Additionally, A5 is proposed as a hybrid system combining both static and dynamic analysis techniques [133]. Our dependence analysis on Android-generated traffic continues the trend of dynamic analysis. Our solution collects the data including the system information and network events. We build the dependence graph of the network requests, trace the origins of events, and thus detect anomalies.

The classification and profiling of Android network traffic have been recently reported in the literature [48, 103, 137, 146]. Authors in [146] proposed to distinguish the mobile traffic by investigating the network traffic using HTTP headers. Mulliner [103] conducted a study to detect the data leaks when people use mobile devices to access websites. Conti *et al.* [43] shows that attackers can recognize user's actions on Android apps by analyzing the network traffic, even it is encrypted. In comparison, our solution is focused on inferring the traffic dependence for identifying the malicious requests. Other existing solutions (e.g., [48, 137]) on profiling Android network traffic are not used to detect the malicious requests, therefore, do not solve our problem.

2.4 Visual Analytics for Network Security

Information visualization has been widely adopted in the network security area. A collection of the security visualization tools is summarized in [100]. Many visualization designs have been proposed as the intuitive and efficient means of displaying complex structures that cannot be explicitly seen in the raw data [120]. GraphPrism [82] adopts node-link diagrams and heatmaps to compactly visualize the large and complex networks topology. ClockView [87] provides a scalable glyph representation of host activities. The advantage of the circular design in ClockView is that people can read the aggregated overview data and smoothly switch between different levels of the hierarchy. Hviz [70] is a visualization tool designed for explore anomalies based on cross-computer correlation analysis. Authors propose to group and aggregate the HTTP traffic into abstract events (e.g., according to requests' effective second level domain). In contrast, we focus on the host-based data and condense the display based on the structural information and legitimacy of the data. Besides, the visualization we proposed is space-efficient regarding the dependency structure.

Traffic Circle [31] and Radial Traffic Analyzer [84] are two network visualization tools that adopt the radial layout. Both solutions are used for monitoring the network (e.g., identifying communication patterns) by displaying the raw packet or flow information. In our solution, we use concentric arcs to display different dependency levels, in such way that our tool visualize the logical relations of the network traffic in an information-rich fashion.

Visual analysis of network anomalies. In recent years, many research efforts have been dedicated to visualizing the network traffic and highlight the anomalies [25, 35, 42, 75, 115]. For example, Conti *et al.* [42] created two complementary security visualization systems that help find anomalous behavior on a high level and perform rapid in-depth packet level analysis. TVi [35], as a visual querying tool, is proposed to monitor the network and detect anomaly from Snort logs. BURN [115] is a tool to display autonomous systems exhibiting rogue activity and helps at finding misbehaving networks. BURN can be used for manual inspection of the root-cause, but the visualization tool does not provide a structural view of the data.

Traffic Causality Graph (TCG) [25] enables the profiling of network application through the temporal and spatial causality of flows. This solution aggregates the packets into flows and focuses on the causality of them. The causality finding algorithm in [25] is based on the protocol, IP and port information of flows, while our fine-grained algorithm relies on the application layer information and has better descriptive power. Traffic Dispersion Graph (TDG) [75] is used to display the network-wide flows by aggregating the packets. TDG captures the interaction between hosts in a network, while our triggering relation model is built on the host-based network traffic. Last, our visualization tool differs from TCG and TDG, as they aim at classifying the traffic or detecting port-related threats (e.g., port scanning). However, ours is used for identifying the general anomalous requests through revealing their triggering relations.

Chapter 3

Triggering Relation Model

In this chapter, we formalize our *triggering relation model*, introduce the terminology used in the model, and present the security applications.[1] The chapter is organized as follows. We give our model and new definition of security in §3.1. We then introduce the *triggering relation discovery* problem in §3.2. The security applications of our dependence model are described in §3.3.

3.1 Triggering Relation Graph: Definitions and Properties

In this section, we introduce the concept and properties of the triggering relation model.

Definition 3.1.1 (Triggering relationship). *The triggering relation of two events e_i and e_j exists if e_j cannot be issued unless e_i is sent out first. We denote e_i triggers e_j by $e_i \rightarrow e_j$.*

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 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). We also refer the triggering relation $(e_i \rightarrow e_j)$ as the *parent-child* relationship, where e_i is the parent trigger or parent and e_j is the child.

Triggering relations of events can be represented in a directed graph that is 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.

[1]The content of this chapter is mainly based on our papers [154, 157].

We illustrate two TRG examples in Figure 3.1. 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. A TRG provides a structural representation of triggering relations of observed events. 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.

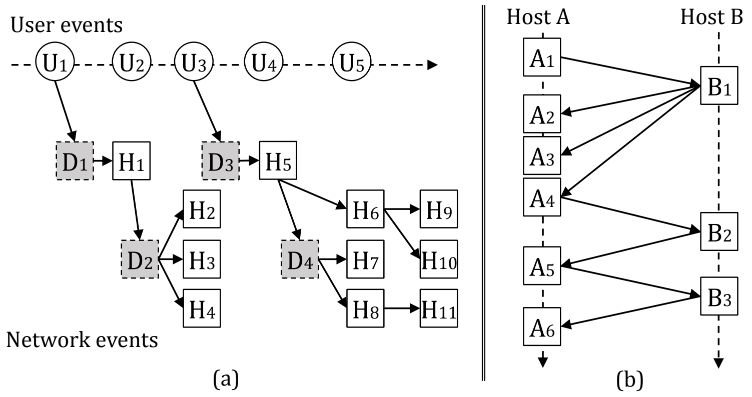


Figure 3.1: Schematic drawings of triggering relation graphs for outbound traffic from a host (a) and traffic between two hosts (b).

Definition 3.1.2 (Triggering Relation Graph). *A triggering relation graph (TRG) is a forest of trees of arbitrary depths with directed edges representing the dependencies among network events and user actions. Each tree is rooted at a legitimate user event or network request. The internal and leaf nodes of the trees are network events. A directed edge from event e_i to e_j represents that event e_j is caused by e_i. The trees in the forest are chronologically ordered, so are the children of a node.*

The TRG is composed of two types of nodes, user events and network events. *User events* refer to the user's actions to the application through input devices such as the keyboard or mouse. A user event in TRG is legitimate if and only if it is not forged by any malicious software. We give several practical techniques for ensuring the authenticity and provenance of user events in §4.5.2. In the context of a browser, we consider two main types of traffic-inducing user events: mouse clicks on hyperlinks and keyboard inputs to the address bar.

Network events refer to the outgoing HTTP requests from the host. They are further categorized into different levels according to their relative dependencies.[2] We refer the first request that triggers

[2]We use the phrases network event and network request interchangeably in the dissertation.

others as the *root-trigger* request. For example, the root-trigger requests may fetch web objects by generating additional outgoing requests from a browser, i.e., fetching the images or JavaScript referred to by an HTML page. The root-trigger requests are caused by legitimate user activities or generated by benign apps. By our Definition 3.1.2, a node in TRG has at most one parent, thus, at most one root-trigger.

The edges in a TRG refer to the triggering relations that describe the causal relationship between nodes. We further distinguish two types of edges:

- *Root-trigger dependency* is the relation between a legitimate user's input (e.g., mouse clicks on hyperlinks) and its generated first network request.
- *Inter-request dependency* is the relation between two network requests where one directly triggers the other.

For specific types of network traffic, the TRG may manifest unique topology and properties. In Figure 3.2, we present one concrete example to demonstrate the triggering relations existing in HTTP and DNS requests. After a user clicks a link to Financial Times (`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.

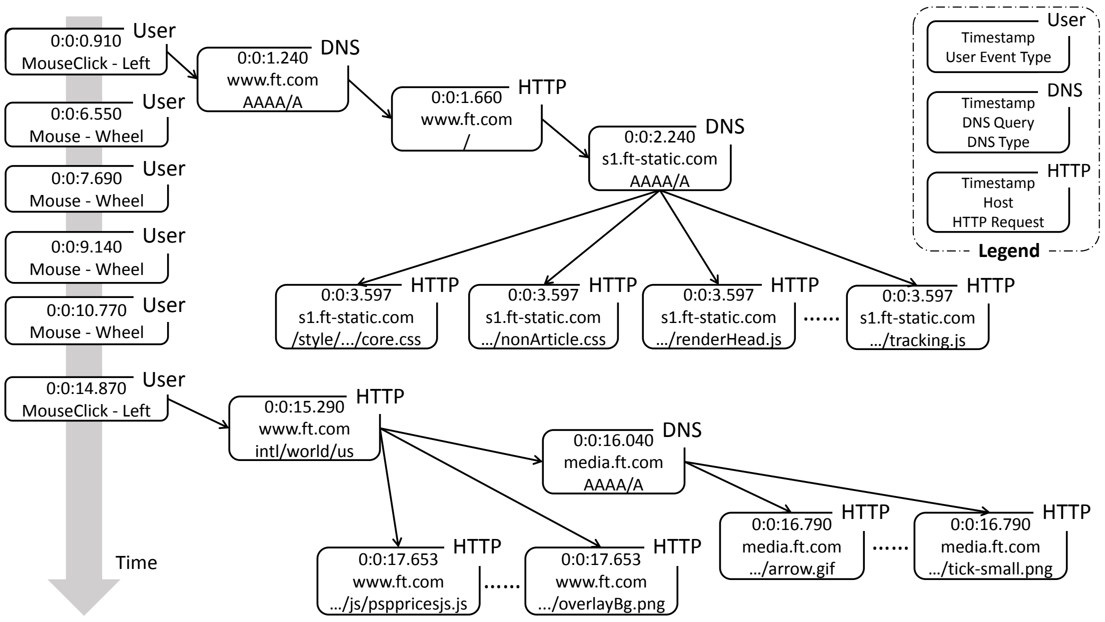


Figure 3.2: An example of triggering relation graph for outbound DNS and HTTP traffic on a host. The network and user events are denoted as rounded squares. The arrows represent the triggering relations between events.

The TRG is composed of a list of trees, which are rooted by their *root-triggers* and can be sorted chronologically. Each tree depicts one scenario that how user requests the network resources (e.g., browsing a website, downloading a song). The graph allows one to understand the causality among network events and find anomalous ones (i.e., requests) based on their root-triggers. A TRG satisfying its definition is *well-formed*. We give the definition of security below.

Definition 3.1.3 (Security). *In the user intention-based security model, a legitimate network event belongs to a tree in the TRG as defined in Definition 3.1.2. That is, the network event e is either a root-trigger, i.e., the child node of a legitimate user action, or e's ancestor node is a legitimate root-trigger. Otherwise, the network event is a vagabond request and considered suspicious.*

TRG has its unique properties, enabling us to adopt different strategies to infer the triggering relationship. We summarize the properties of TRG $\mathbb{G}$ as follows.

- **Attribute-based**: Each event in the TRG has multiple attributes, describing its network and system features that can be used to infer the triggering relationship.
- **Expandable**: TRG is a forest of trees in arbitrary height. The degree of a node in a tree can be expanded as it grows. If one legitimate network event e is appended to $\mathbb{G}$, the TRG still meets the security definition.
- **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 enforcement of dependence has not been previously studied in the literature as a general method for anomaly detection. Our work focusing on the dynamic interactions between the user and application lends a novel security methodology that is not limited to the ever-changing anomalous patterns. One important use of the TRG model is to enforce the *legitimacy inheritance* of the nodes in dependence graph. In this policy, the legitimacy of the tree is determined by its *root-trigger*, i.e., the dependent nodes are benign as long as the root is legitimate.

3.2 Triggering Relation Discovery

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 create the TRG by incrementally inserting new events to a partially well-formed TRG by predefined algorithms. Specifically, the tree-based TRG enables us to apply the breadth-first traversal when inferring dependencies. 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 events. 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 Figure 3.3.

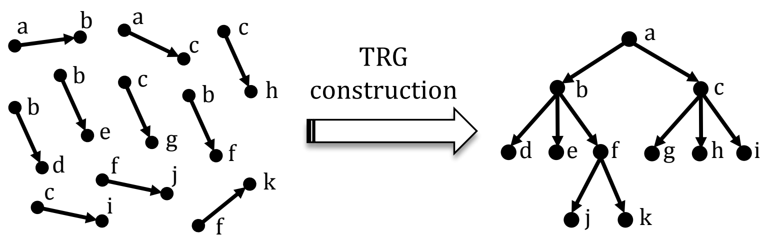


Figure 3.3: The triggering relation graph (TRG) on the right can be constructed from the pairwise triggering relations on the left.

We, therefore, transform the problem of discovering triggering relations among a set of events into discovering the triggering relations of pairs of events, which is defined as the *pairwise* triggering relationship. 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 the pairwise features, so that the discovery problem can be efficiently solved using classification tools. In Figure 3.4, we give an overview of main operations used in rule- and learning-based approaches. The details of both approaches are given in §4.

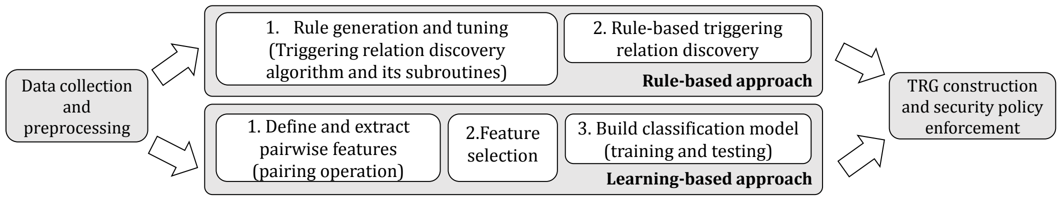

Data collection and preprocessing
1. Rule generation and tuning (Triggering relation discovery algorithm and its subroutines)
2. Rule-based triggering relation discovery
Rule-based approach
1. Define and extract pairwise features (pairing operation)
2.Feature selection
3. Build classification model (training and testing)
Learning-based approach
TRG construction and security policy enforcement

Figure 3.4: System workflow containing major operations in the rule- and learning-based approaches.

Our definition of event-level triggering relation discovery relates to, but differs from the service dependency inference in existing research work (e.g., [105, 152]). The service dependency refers to that one service relies on another to function, e.g., a web service depends on the DNS resolution. Our event-level triggering relationship refers to the causality of two network events, e.g., the transmission of one network packet triggers the transmission of the other.

Our triggering relation graph is defined differently from the parental dependency graph (PDG) in WebProphet [92] in terms of graph semantics and security applications. The PDG in WebProphet predicts the performance impact of webpage objects. Therefore, the graph is built based on the timing information only, without capturing the causality among requests or objects. We rely on the semantic information (e.g., domain name, request string) of each request to build TRG, thus enabling the TRG to detect anomalous network events.

3.3 Security Applications of Triggering Relation Graph

We consider the malware that behaves as a stand-alone user-space application on a host or a device. In the stealthy malware taxonomy [116], the family of application-level malware is referred to as *Type-0 malware* – the malware that “does not interact with any part of the operating system (nor other processes) using any undocumented methods”. Type-0 malware does not intend to break kernel’s functionality or inject code into kernel’s API during its execution. Type-0 malware differs from rootkits as it does not modify data or code sections of the kernel or the running processes, or intended to gain root access to the operating system.

In this dissertation, our traffic dependence analysis is designed to detect general stealth malware on hosts and mobile devices. The stealthy malware behaves as a user-level application, e.g., certain instances of spyware, malicious bots performing data exfiltration, and botnet launching denial-of-service attacks. Specifically, we consider three types of threats on hosts as follows.

- Malware is an extension or add-on component of an existing legitimate application, e.g., spyware as a malicious Firefox browser extension or parasitic malware [127]. Malware runs along with the host program and has the same process ID as the host one.[3] A specific example of such a type of spyware is `FFsniff`, which secretly sends out victim’s ID along with the password to the remote host.
- Malware is a stand-alone user-level application and runs with a unique process ID (e.g., `Trojan.Brojack.A`). This type of malware commonly threaten the data confidentiality and system integrity.
- Software may perform undesirable and unauthorized network activities that are not causally related to the user’s inputs due to inadvertent software flaws or software errors (e.g., software

[3]If malicious extensions can modify the existing DOM of a visited page loaded in a browser, a user may be tricked to click malicious links, which sends outbound requests for unintended objects. To detect this attack, one needs to utilize additional techniques (e.g., WebCapsule [106], WebWitness [108], or a parallel universe for predicting browser contents [142]).

behaviors that deviate from specifications). Identifying stealthy unwanted traffic is important, as these packets may leak user information, consume bandwidths, and cause further security vulnerabilities [79].

For protecting the Android security, we adopt our TRG model to detect the popular mobile threats: repackaged apps that result in data leak, drive-by download apps, and Android bots.

- **Repackaged apps** refer to the malicious apps created by repackaging the existing benign ones [161]. They are known to contain malign payloads that may cause malicious requests, which are usually generated without user's consent, e.g., `AnserverBot` malware family.
- **Drive-by download apps** fetch malicious payloads at runtime. They lure users to install a new malicious app or create a shortcut icon to some malicious or advertisement sites, e.g., `com.Punda.Free`. Besides, drive-by download attacks take place when users view a compromised web page. For example, attackers may insert a hidden iframe in the bottom of a hacked website, which directs the link to a malicious download when users access the website using a browser app [3].
- **Android bots**, being controlled by bot masters through the network, can be used to conducted remote attacks. Android bots are not exclusive to the other two categories. Their behaviors often include the stealthy network communication to remote command and control (C&C) servers. Due to the lack of signatures for newly invented bot apps, it is quite challenging to detect the ever-changing bots.

Based on the TRG model, we introduce a *user-intention* based security policy, which enforces that *all network events must be attributed to the legitimate user events*. This security policy classifies the network requests on TRG without valid triggers as *vagabond* requests. The anomalous events without legitimate causes are likely raised by stealthy malware activities. Hence, by inferring the triggering relations on traffic data, our analysis can reason about the network incidents and pinpoint suspicious activities that are not intended by users. Blocking these outbound requests 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 dependency model to detect stealthy malware activities on a host. DNS tunneling has been abused by botnets for command and control communications [143].

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. Our tool detects these vagabond events and reports them.

In our dependence analysis, 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. With a built TRG, advanced security policies can be proposed depending on the traffic types (e.g., HTTP, TCP, or mixed), host types (e.g., client or server), and the definitions for network anomalies. Our model can be used in combination with conventional signature-based and statistic-based detection.

Chapter 4

Host-based Network Anomaly Detection via Triggering Relation Discovery

In this chapter, we design and compare general approaches for building triggering relation graph on network data.[1] By enforcing the *root-trigger* policy, we can ensure an application's correct responses to user activities and identify anomalous activities based on the structural information from TRG model.

The chapter is organized as follows. We present the motivations, specific problems, and an overview of our approaches on the host-based security in §4.1. The rule- and learning-based approaches are described in §4.2 and §4.3, respectively. The implementation details and security analysis are given in §4.5. We conduct extensive evaluations on our host-based solution and present the results in §4.6. §4.7 concludes the chapter.

4.1 Motivation and Design Goals

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. In this chapter, we propose to discover the triggering relations on network traffic data and leverage the structural information to identify anomalous activities that cannot be attributed to a legitimate user request (i.e., user intent).

Discovering user intention-based traffic dependencies is challenging, because modern applications such as web browsers often automatically fetch content and generate requests without explicit

[1]The content of this chapter is mainly based on our papers [154, 157].

user actions. The dependencies of those legitimate requests should be properly identified without triggering false alarms. One needs to discover not only the dependencies among user actions and network events, but also the layered dependencies of those network events. This chapter focuses on outbound HTTP packets by the web browser, which can be generalized to other types of applications and network-flow types.

We apply the *triggering relation* model (in §3.1) on network data 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.

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 [67]. Thus, analyzing relations between network events provides important insights for identifying network anomalies. 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.

4.1.1 An Overview of Our Approaches

We design and compare rule- and learning-based approaches to infer the triggering relations on network data, and thus detect network activities of stealthy malware.

Based on our observation from the real world network traffic, a rule-based approach is created to detect anomalous HTTP requests on a host. The algorithm is designed to efficiently infer the triggering relations of outbound requests, i.e., to find the network event that causes the newly-observed outbound request. We explore the direct and indirect dependencies in how a user interacts with applications and how applications respond to the user's requests following the specifications of the applications. The rule-based method serves as a baseline for the comparison with our learning-based approach.

We design a new learning-based method that scales. General techniques for learning and recognizing directional triggering relations of network events do not exist. Most of the existing machine learning based security studies are on binary classification problems, where an unknown instance (e.g., email, code, or network request) needs to be classified into two classes – legitimate or suspicious. In our approach, we design a new function *pairing* that produces distinctive pairwise features, so that the discovery problem can be efficiently solved with existing binary classification methods (e.g., SVM). Our analysis using machine learning is scalable, capable of rapidly processing a large amount of traffic. Using machine learning algorithms eliminates the need for manually deriving classification rules and thus simplifies the detection.

A triggering relation graph can be built with the outputs from either the rule- or the learning-based approach. We further introduce a *root-trigger* security policy for pinpointing anomalous activities based on the TRG. By enforcing this user-intention based policy, we can identify the vagabond events (i.e., suspicious requests) that are lack of valid triggers, and thus ensure an application's correct responses to user activities. *Vagabond events* refer to outbound network events that are not generated by any user actions and may hence be due to anomalies.

4.1.2 Comparisons Between Rule- and Learning-based Approaches

We further make a detailed comparison of properties between these two triggering relation discovery approaches in Table 4.1. The rule-based approach is only applicable to known patterns, and requires non-trivial human efforts in rule generation and tuning. Yet, the machine 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.

Table 4.1: 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)

4.1.3 Contributions

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. We demonstrate with experiments that discovering fine-grained causality in network traffic is feasible. The significance of our traffic reasoning approach is its ability to detect new and stealthy malware activities.

Our contributions are summarized as follows.

1. We present a rule-based solution for discovering the triggering relations on network events. The triggering relation discovery algorithm, based on empirically derived rules, inspects the temporal, semantic, and process-related attributes to reveal the causal relationships.

2. We propose a scalable learning-based approach that applies to multiple types of network traffic. We introduce a new operation called *pairing*. This operation converts individual network event into pairs with comparable attributes, enabling the use of binary classifiers for the triggering relation discovery.

3. We utilize a new *root-trigger* security policy to infer the origins of malicious activities on discovered triggering relations (i.e., TRG). 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 our solution on 10+ GB data (including a DARPA dataset and real-world network traffic). Our results indicate that both rule- and learning-based approaches successfully detect stealthy malware activities on hosts. With good scalability for large datasets, the learning-based method achieves better classification accuracy than the rule-based one.

4.2 Rule-based Triggering Relation Graph Construction

In this section, we describe a rule-based approach for discovering triggering relations of HTTP requests on a host. A triggering relation discovery algorithm is designed to identify the triggering relationship 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. Our rules are generated based on the patterns of user interactions and the attributes of HTTP requests from the browser.

4.2.1 Triggering Relation Discovery Procedure

This section describes our breadth-first search (BFS) based algorithm for the TRG construction. The algorithm utilizes the building blocks (namely `IsChild`, `IsSibling`, and `IsRoot`), which are presented in the next section.

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.

Algorithm 1 Triggering Relation Discovery (TRD) Algorithm

```
Input: A newly-observed network event $p$; 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 $T_i \in \mathbb{T}$; and a threshold $\tau$.
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:     $\mathbb{T} \leftarrow \mathbb{T} \cup \{T_{m+1}\}$
 4:     return True
 5: else
 6:     for $i \leftarrow m$ to 1 do
 7:         continue if $p.\mathsf{time} - r_i.\mathsf{upTime} > \tau$ or $r_i.\mathsf{pid} \neq p.\mathsf{pid}$
 8:         define a queue $Q$ and enqueue $r_i$ onto $Q$
 9:         while $Q \neq \emptyset$ do
10:             node $n \leftarrow$ dequeue $Q$
11:             if IsChild($n, p$) then
12:                 UpdateTime($r_i$.upTime)
13:                 append $\{n \rightarrow p\}$ to $T_i$
14:                 return True
15:             else if IsSibling($n, p$) and not IsRoot($n$) then
16:                 UpdateTime($r_i$.upTime)
17:                 append $\{n.\mathsf{parent} \rightarrow p\}$ 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
```

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 $\mathbb{T}$ 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 $T_i \in \mathbb{T}$; *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 ones in a constructed TRG. Therefore, our algorithm opts for a breadth-first traversal of 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.

We further optimize the TRD algorithm by avoiding unnecessary comparisons. We achieve the speedup by leveraging the underlying consistency of attributes (e.g., PID). Besides, we adopt the `UpdateTime` function to refresh the upTime for each tree in $\mathbb{T}$, in which way we track the freshness of a tree and skip the comparison between a newly coming request and an out-of-date tree.

If the triggering relation is not found after all nodes in the tree are compared, then the next root-trigger and its descendant nodes are compared. Intuitively, the process terminates if either a dependence is found or all existing requests have been compared. The worst-case complexity of this TRD algorithm requires traversing the entire TRG, and is $O(n)$ where n is the total number of network events on the TRG.

4.2.2 Details of Sub-procedures

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 the timestamps of p_a and p_b is within a threshold τ and 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 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.

Finding sibling relations is useful in identifying new triggering relations in our causality reasoning analysis. 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.

4.2.3 Discussion on the Usability Under Complex Web Scenarios

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

Based on the algorithm and building blocks described in §4.2.1 and §4.2.2, we implement a prototype **CR-Miner**, as our rule-based triggering relation discovery solution. In this section, we discuss the usability of CR-Miner under complex web scenarios.

- **Requests to third-parties** such as `doubleclick` or `facebook` are automatically issued by the browser. The triggering relation of these records can be recognized in our solution, because the referrer fields of the third-party traffic match the hosting domain. Alternatively, parsing and analyzing the content of proceeding webpages have been used to predict future (legitimate) outbound requests in [142], which is compared with our approach in §2.1.

- **Redirection** allows the browser to issues an HTTP GET request from the URL A' different from the user's original request to the URL A. The triggering relation of redirected traffic can be identified by CR-Miner as the request for A' contains the original domain in its referrer.

  Along with other attributes, one may establish that A' is a child node of A by our triggering relation discovery (TRD) algorithm in CR-Miner. Therefore, our algorithm ensures the causal chain.

- **AJAX technique** allows users to retrieve information from the web server without interfering the display of the current page. Our experiments (with AJAX traces) confirm that CR-Miner handles the AJAX traffic, in terms of discovering the dependencies.

- **HTTPS traffic** contains encrypted HTTPS packet that can be sniffed by a known SSL proxying technique [2]. The technique allows to generate a certificate for the server and signs it with its own root certificate, so that the client's outbound traffic is relied by the man-in-the-middle and can be inspected by our tool.
- **Automatic updates** do not have explicit user actions that trigger them (e.g., system updates and RSS feeds). One mitigation is to recognize the periodic update requests with a predefined whitelisting. Besides, a learning-based approach to generating a whitelisting is also introduced in §5.3.3.
- **Browser caching** does not affect CR-Miner. According to RFC2616 [57], even when a browser caches HTML files locally, it still sends requests to remote servers for checking the freshness of the cache. The servers may return the code 304 (Not Modified), or the new contents. Thus, CR-Miner still captures all the necessary requests for the dependence analysis.
- **Social engineering attacks** allow an attacker to fool a user to click on a malicious link and visit an attacker's website. As a result, the traffic to attacker's website has the proper traffic dependence (i.e., triggering relation) and are not deemed suspicious. Hence, CR-Miner requires additional mechanisms to educate users about social engineering attacks.

Summary. Compared with the traditional approaches of analyzing network requests independently in isolation, our approach provides more structural and contextual information for anomaly detection on network activities. We confirm the usability of our prototype under complex web scenarios, such as requests to third-party hosts, redirection, AJAX calls, HTTPS traffic, and automatic updates.

4.3 Learning-based Triggering Relation Graph Construction

We describe a machine learning approach to inferring triggering relations among network requests in this section. Compared with the heuristic rules, the learning-based solution can achieve better classification 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 then show how binary classification algorithms (e.g., SVM) can be used for the triggering relation discovery. Last, evaluations confirm that our proposed learning-based approach achieves high scalability and detection accuracy.

Compared with the learning-based approach described in this section, the rule-based one (in §4.2) 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. In the presence of a large number of features or due to the limitations of human knowledge, the rules may not be generated.

Our work on triggering relation discovery using probabilistic machine learning algorithms 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.

4.3.1 An Overview of Learning-based Approach

The primary operations in our learning-based method are Data Collection, 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.

- Data Collection is to record and store the events $\mathbb{E} = \{e_1, e_2, \ldots, e_n\}$ to be analyzed. Given one event $e_i \in \mathbb{E}$, e_i has m attributes and each is denoted as $e_i.attr$, where $attr$ is one type of features (e.g., time, IP, port, host, URL, etc). All the features can be obtained by inspecting the event e_i.

- Pairing is a new operation that we design for extracting pairwise features of events' attributes. Its inputs are two events e_i and e_j. Pairing operation outputs the event pair (e_i, e_j) with m pairwise attribute values. Each pairwise feature is obtained by computing $f_k(e_i.attr_k, e_j.attr_k)$, where f_k is a comparison function for the k-th attribute in the events $(1 \leq k \leq m)$. Therefore, the output of Pairing operation on events e_i and e_j can be written as

$$P(e_i, e_j) = \{f_1(e_i.attr_1, e_j.attr_1), \ldots, f_m(e_i.attr_m, e_j.attr_m)\}. \tag{4.1}$$

  The comparison function f_k (e.g., `IsGreaterThan`, `WithinThreshold`, `IsSubstring`, `IsEqual`, 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 §4.3.3. 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 relationship in an event pair, e.g., $(P(e_i, e_j), 1)$ represents that event e_i triggers e_j. The Data Labeling operation can be done by running simple rules and verifying by domain experts, both of which require manual efforts.

- Training is the operation that produces a machine learning model with labeled training data. It takes as inputs the features of event pairs and their labels.
- Classification is the operation that leverages the trained machine learning model to predict triggering relations on the new event pairs $\mathbb{P} = \{P(e_i, e_j)\}$, where $e_i \neq e_j$. The outputs of binary prediction results are in the form of $\mathbb{P}^* = \{(P(e_i, e_j), l_{ij})\}$, 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 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.

Figure 4.1 illustrates the workflow of our learning-based approach. We describe and highlight the design details of our new Pairing operation in the next section. The feature extraction using pairwise comparison is unique and enables the use of binary classifiers for solving the triggering relation discovery problem.

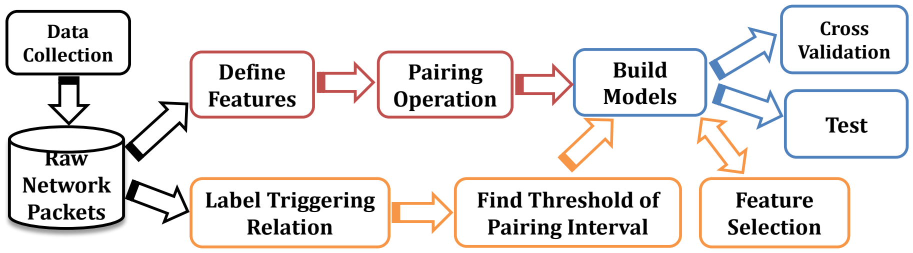


Figure 4.1: The workflow of the machine learning approach.

4.3.2 Pairing Operation

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 HTTP requests as an example. The approach can be generalized to other event or traffic types, which is evaluated in §4.6.

In Table 4.2, we show examples of some HTTP events. The triggering relations, if known, are shown in the last column (under ParentID). The features in Table 4.2 are derived from the header

Table 4.2: Original network events observed.

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**
...									

Note: 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).

of HTTP requests. As the header contains operating parameters of an application layer transaction, the triggering relation can be measured by the attributes of the requests. These attributes are previously used to understand the behavioral model of web traffic [40], while our work further leverages them to build the trigger relations of network traffic for the security purpose.

Table 4.3: Examples of pairwise attributes as outputs of the Pairing operation.

(ID1,ID2)	TimeDiff	PIDDiff	AddrDiff	RequestSim	HostSim	ReferrerSim	Q1	R2	Relation
(4,5)	0.01	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**

Note: Q1 and R2 stand for the first event's request type and the second event's referrer type, respectively.

Six event pairs are generated and their new pairwise attributes are shown in Table 4.3. For example, the HostSim measures the string similarity of the Host attributes between two events. Each pair has a binary representation of the existence of a triggering relation (under Relation in Table 4.3). 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.
- 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, $f_{sim}(e_i.attr, e_j.attr)$, where function f_{sim} is a similarity measure, e.g., normalized edit distance. Take HTTP request as an example, we compute pairwise features HostSim and ReferrerSim by measuring the string similarities between two host fields and two referrer fields, respectively.

- 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.

Our structural scalable and semantic-aware approach is suitable to discover triggering relation of events in time series. The events, such as network requests, are partially ordered in time and have a list of comparable attributes. Because the relation of events is *asymmetric* and *transitive*, the pairwise triggering relationship is unidirectional between two events. Our approach pairs any two events at most once and outputs the set $\mathbb{P}$.

4.3.3 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 [156].

We describe the efficient pairing algorithm – 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.

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.

Divide-and-conquer. Given a list of events $\mathbb{E}$, we divide $\mathbb{E}$ 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 pairing computation on disjoint data can be made parallel. However, the overall complexity is still bounded by $O(n^2)$.

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

Algorithm 2 Efficient Pairing Algorithm (EPA)

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

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.

4.3.4 Feature Selection and Classification

Feature selection is to find an optimal set of representative features can significantly improve the effectiveness of machine learning classifiers. 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 standard supervised machine-learning classifiers – Naive Bayes, a Bayesian network [78], and a support vector machine (SVM) [44].

Cost Sensitive Classifiers. Due to the sparsity of triggering relations existing in network traffic, we leverage the customized cost matrices [50] 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 4.4, 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 §4.6, we thoroughly evaluate how cost matrices improve our analysis accuracy.

Table 4.4: 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 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 from the learning-based approach, 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*, 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.

Based on the results obtained from the TRD algorithm in §4.2.1, 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. Therefore, we design a root finding algorithm (RFA) to obtain the root(s) for a given event e_k, based on all predicted pairwise triggering relations.

Algorithm 3 Root Finding Algorithm (RFA)

```
Input: An event $e_k$; $\mathbb{P}^* = \{(P(e_i, e_j), l_{ij})\}$, where $e_i \neq e_j$ and $l_{ij} \in \{0, 1\}$.
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 $\mathbb{F} \leftarrow$ find $n$'s parent(s) based on $\mathbb{P}^*$
 6:   for each event $e \in \mathbb{F}$ 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 $\mathbb{R}$
```

Shown in Algorithm 3, the inputs of the root finding algorithm are an event e_k and a set $\mathbb{P}^*$ containing all the predicted pairwise triggering relations. The output is a set containing all the roots of e_k. To compute the transitive reduction, we opt for a queue Q to perform breadth-first traversal of TRG. In each iteration, we obtain the parent(s) $\mathbb{F}$ of a dequeued event n. For each event e in the set $\mathbb{F}$, 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.

By the definition of triggering relation graph (TRG) in §3.1, 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 root finding algorithm needs to identify 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 constructed TRG in Figure 4.2. Our root-trigger policy allows the existence of multiple intermediate parents for a node, as long as the root trigger is correct, e.g., Figure 4.2 (c).

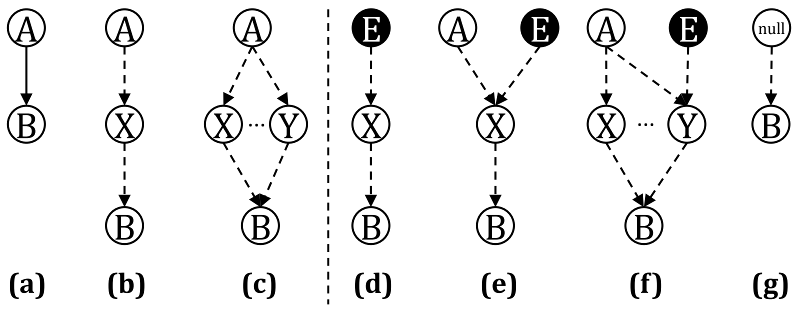


Figure 4.2: The illustration of various cases where B's predicted root trigger is correct (a-c) or wrong (d-g) on the triggering relation graph constructed from pairwise triggering relations. Let the ground truth of B's root trigger be A. Case (a) is where B's parent is also B's root. Cases (b) and (c) 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 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` [17]). 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 §4.6, 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.

4.5 Prototype Implementation and Security Analysis

We present *CR-Miner*, a prototype of our host-based solution in §4.5.1. Then, we analyze the robustness of CR-Miner against several potential circumvention attacks in §4.5.2.

4.5.1 Prototype Implementation

We develop the prototype *CR-Miner* in the Windows 7 operating system. The detailed architecture of our prototype is shown in Figure 4.3. We build CR-Miner (refer to the darker parts in Figure 4.3) between the application and the kernel layers.

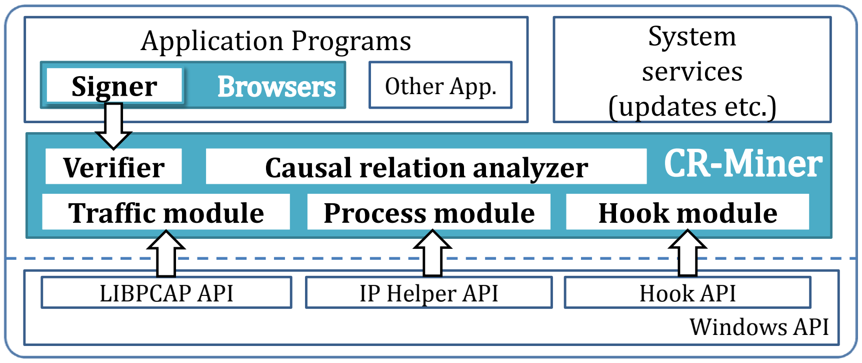


Figure 4.3: The architecture of the CR-Miner prototype.

There are three sensors deployed to collect data on the host. The causal relation analyzer computes the triggering relations based on the algorithms and rules described in §4.2. The Windows APIs (namely hook API, IPHelper API, and libpcap API) are used in the implementation. Our implementation details are described next, including process identification, traffic monitoring, and user event collection.

The traffic module collects the outbound HTTP `GET` and `POST` requests via the `libpcap` API. We record the HTTP request header information, including the timestamp, IP address, get URL, host, and etc. The process module obtains network and system information of the active connections. We obtain the IP table, a kernel data structure, by using `GetExtendedTcpTable()` in `IPHelperAPI.dll` and associate the TCP connections to corresponding process IDs.

The hook module collects the kernel-level user events. By using the existing Windows `hook` API, this module installs the hooks to log keyboard and mouse events (including mouse click, mouse double click, mouse wheel, and key press). Furthermore, we obtain the process ID of the current foreground window by using `GetWindowThreadProcessId()`. In such way, we can figure out the corresponding process for each user event. Repetitive user events that do not generate traffic such as mouse movements are ignored.

We also record user events at the application level through the use of `Tlogger`. It is a Firefox extension for capturing the information of mouse clicks during web browsing [17]. Tlogger provides the URL information associated with mouse-click events and the tab-related information. Attributes of user events collected at the application level and kernel level are both used for identifying the root triggers in the `IsRoot` procedure (see §4.2.2).

4.5.2 Security Analysis

In this section, we answer the question *Can CR-Miner be tricked?* We examine the data integrity in our prototype, as the data collection operation solely relies on the security of CR-Miner. Once the data is collected, the dependence analysis may be conducted on a separate trusted machine. Thus, the main security threats come during the data collection phase. Our threat model (in §3.3) considers application-level malware. Therefore, we analyze the security and defense of CR-Miner against two types of attacks: *i) forgery attack* where an adversary modifies attributes of his network activities to make them appear legitimate, and *ii) piggybacking attack* where an adversary strategically determines when to send outbound requests and exploits CR-Miner's temporal rules. We then summarize the effectiveness of CR-Miner in achieving our security goal of identifying anomalous network activities.

Integrity of traffic data. Our triggering relation discovery solution relies on the integrity of the data collected and analyzed, specifically the outbound HTTP header. Malware may attempt to spoof the header fields in its outgoing request, e.g., forging its referrer field in the HTTP header so that it appears to be referred by a valid root-trigger.

To prevent this problem, we equip the browser with a *signer* (in Figure 4.3), which implements a lightweight message authentication code to ensure the integrity of the HTTP header created. Then, the signed headers are verified by a trusted program called *verifier* on the same host. The signer and the verifier share secret keys that are used for signing and verification. Traditional key distribution mechanisms, such as the Diffie-Hellman key exchange scheme, can be used to set up the shared secret key as the operating system starts.

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

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

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

Our cryptographic operations are orthogonal to those provided by SSL, which is for end-to-end security with a remote server, whereas the purpose of signer and verifier is to guarantee the network requests are not tampered with by malware before leaving the host. In our prototype, both the signer and verifier reside on the same host. Our above integrity verification solution may bear superficial similarity with the web referral architecture against DDoS [135]. However, our verification mechanism is specific to web browsers and is designed to protect against stealthy malware on the host.

Integrity of system data and user events. Our threat model considers application-level stealthy malware. Therefore, the kernel-level system data, including the process ID, keyboard and mouse events, is trusted. Process information (e.g., PID) can be obtained through known APIs. The user-space malware cannot forge process information without the root privilege. Besides, the recent work on cryptographic identification of natives applications in the operating system can be adopted to prevent the forgery of process ID information [20].

Because user activities are used for identifying root-trigger events, the integrity of user events is important. User events may be forged or deleted by user-space application leveraging known APIs. To ensure the integrity of system data, advanced keystroke-integrity solutions such as the provenance verification in [71, 73, 128] can be incorporated in CR-Miner to further improve the system data assurance, which is a useful fail-safe mechanism to guard against potential operational errors. These methods provide provable assurance to the origin of user events and effectively prevent event forgery (i.e., injection of fake user input events). For example, Hasan *et al.* [73] show how to provide strong integrity and confidentiality assurances for data provenance information at the kernel, file system, or application layer, which may be applied to prevent the forgery of user events in our prototype.

Defense against forgery attack and piggybacking attack. Our cryptography-based verification method effectively prevents the forgery attacks, because the headers are tamper-resistant once the browser creates them.

In a piggybacking attack, the adversary sends network requests (to the attacker's server) immediately after a legitimate network event. Such an attack would be effective in a naive temporal-only analysis. However, our dependency rules inspect the context and property of traffic such as domain names, referrers, and PIDs. Therefore, piggybacking requests can be easily detected as vagabond requests, as malware traffic lacks the required attributes. The similar piggybacking attacks are discussed by Xu *et al.* in [145] in the context of detecting drive-by-download attacks.

4.6 Evaluation 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? (§4.6.3, §4.6.4, §4.6.5, and §4.6.6)
- Can our solution detect outbound network activities caused by stealthy malware and real-world threats? (§4.6.7)
- Comparison to the rule-based TRD algorithm, how well does our machine learning approach perform? (§4.6.8)
- How efficient is the learning-based approach? What is the most time-consuming operation in learning-based approach? (§4.6.9)

4.6.1 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 [143]. 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 [6]. 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 sever. 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 to 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. By attempting some remote-to-root exploits, 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`.

Table 4.5: An overview of four datasets in the experiments.

Dataset	**I**	**II**	**III**	**IV**
Type	HTTP	DNS (D) & HTTP (H)	TCP & UDP[†]	TCP
# of Raw Events	HTTP: 45,988	D: 35,882; H: 85,223	All types: 649,787	TCP: 3,010,821
Efficient Pairing (η)[‡]	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[*]	TRD (Algo. 1)	TRD + rules	`TCPFLOW` [16]	`TCPFLOW`
RT labeling[¶]	`Tlogger` [17]	`Tlogger`	Rules	Rules

[†] Dataset III contains 33 different protocols, including TCP, UDP, DNS, and etc.
[‡] η is the reduction percentage after using our Efficient Pairing Algorithm (EPA).
[*] 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).
[¶] RT labeling describes the methods to label root-triggers (RT), e.g., Tlogger and the heuristic rules mentioned in §4.4.

A summary of the experimental data is shown in Table 4.5. In dataset I, we manually check the legitimacy of vagabond requests after running TRD algorithm (Algorithm 1). In dataset III, the malicious requests are labeled based on the IDMEF alerts and audit logs [6]. 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.

Effectiveness of EPA. We define $\eta \in [0, 1]$ as the reduction percentage in Equation 4.2, where $EPA(n)$ is the number of event pairs after using the efficient pairing algorithm in §4.3.3, and n is the total number of events.

$$\eta = 1 - \frac{EPA(n)}{n \times (n-1)/2} \tag{4.2}$$

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 4.6. The comparison between two addresses (IP and port) and the temporal relation are commonly used for all types of network traffic.

Table 4.6: Pairwise features defined on different types of 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.

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 features above, 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. The number of pairwise features in datasets I and II is 12 to 15, while in dataset III and IV, there are about 10 pairwise features.

Classification Setup of Learning-based Method. Three common classification techniques are compared: naive Bayes classifier, a Bayesian network, and a support vector machine (SVM)[2]. 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.

4.6.2 Accuracy and Security Metrics

- The conventional *precision and recall* measures [27] 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.

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN}. \tag{4.3}$$

- 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 with respect to the ground truth obtained through rule-based analysis and manual classification.

- 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.

4.6.3 Detection Accuracy on Dataset I

We present our evaluation results of learning-based approaches on the dataset I in this section.

Accuracy of Pairwise Triggering Relations. We present our experimental findings using the learning-based approach from this subsection. Table 4.7 shows a high prediction accuracy rate for pairwise triggering relations. These results indicate the effectiveness of our binary classification approach.

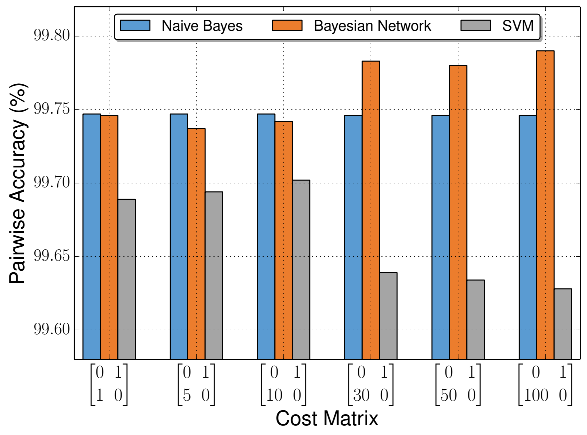
[2]SVM has a polynomial kernel function with a degree of 2.

Table 4.7: Pairwise classification results of train-n-test for four datasets.

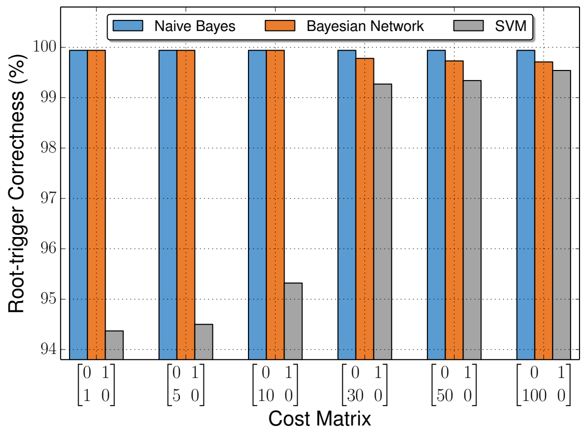
Dataset	# of Pairs in Test Sets	Cost Matrix	Naive Bayes		Bayesian Network		SVM	
			Prec.	Recall	Prec.	Recall	Prec.	Recall
I	3,318,328	$\begin{bmatrix}0,1\\10,0\end{bmatrix}$	0.954	0.996	0.956	0.996	0.958	0.997
II	693,903	$\begin{bmatrix}0,1\\1,0\end{bmatrix}$	0.959	0.998	1.000	1.000	1.000	1.000
III	25,694,154	$\begin{bmatrix}0,1\\30,0\end{bmatrix}$	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.

We vary the cost matrices and compute the pairwise accuracy rates of the three classifiers. The results are shown in Figure 4.4a. 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 4.7, 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.



(a)



(b)

Figure 4.4: Accuracy and correctness results under various cost matrix conditions for datasets I. The results of pairwise accuracy are shown in (a). The results of root-trigger correctness are shown in (b).

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 §4.4) on the predicted triggering relations, we obtain the root-triggers of all events and compare them with the ground truth.

Figure 4.4b 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 4.8, 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.

Table 4.8: 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 Figure 4.2 in §4.4.

Detection of Malicious Traffic in Dataset I. As defined in §4.4, 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 locate in the US, except one IP located in Netherlands. Some of the vagabond requests are false alerts, details of which are given next.

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.

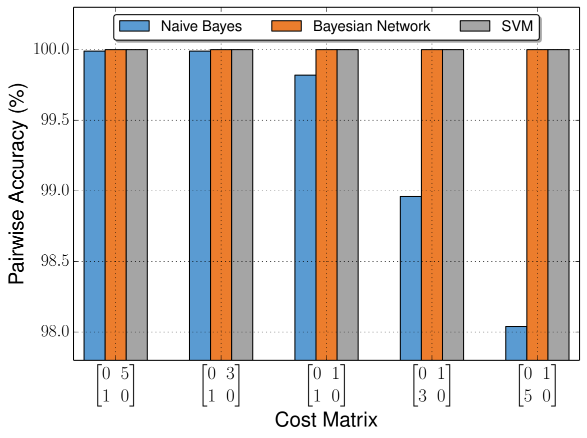
- Missing or incomplete attributes in the original data due to server configuration, e.g., redirection without properly setting the referrer field. There are 244 misconfigured requests that are sent to 38 different domains, usually image/video hosting websites.
- Unconventional attribute values, e.g., 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 bookmark icons.

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

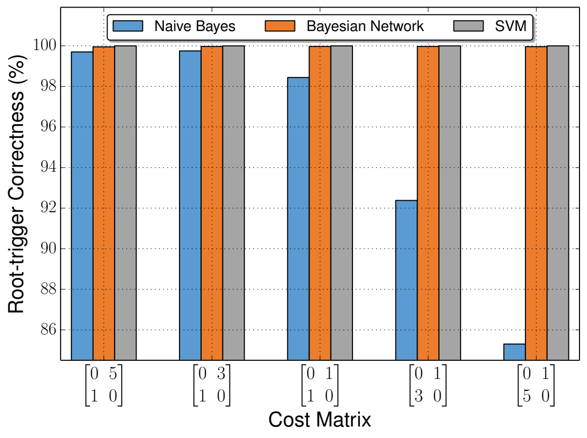
4.6.4 Detection Accuracy on Dataset II

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.

Pairwise Classification Accuracy. The pairwise classification results on dataset II are presented in Table 4.7. All three methods give high pairwise classification accuracy, 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.



(a)



(b)

Figure 4.5: Accuracy and correctness results under various cost matrix conditions for datasets II. The results of pairwise accuracy are shown in (a). The results of root-trigger correctness are shown in (b).

The pairwise classification accuracy under various cost matrices is shown in Figure 4.5a. 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 4.7.

Correctness of Root Triggers. We analyze the root-trigger accuracy for dataset II, and show the results in Figure 4.5b. 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 correctness results under the cost matrix of $\begin{bmatrix}0,1\\1,0\end{bmatrix}$ in Table 4.9.

Table 4.9: 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%

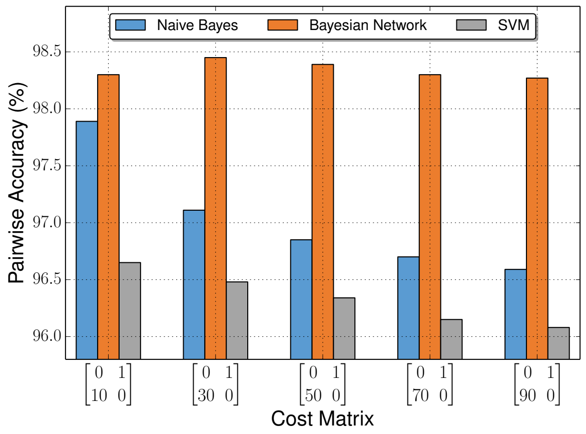
Note: Cases (a-g) refer to the various predicted root-trigger outcomes in Figure 4.2 in §4.4.

4.6.5 Detection Accuracy on Datasets III

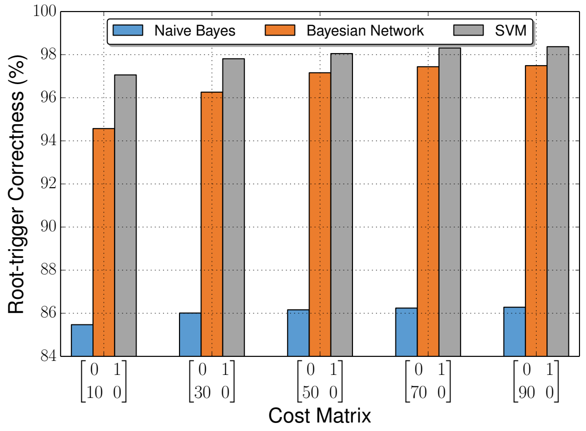
Dataset III (DARPA dataset) contains the network traffic collected from multiple subnets. We focus on three hosts that are compromised by attackers and report the results by weighting the number of requests on each host.

Pairwise Classification Accuracy. Results in Table 4.7 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 other 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 Figure 4.6a, 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



(a)



(b)

Figure 4.6: Accuracy and correctness results under various cost matrix conditions for datasets III. The results of pairwise accuracy are shown in (a). The results of root-trigger correctness are shown in (b).

two other classifiers, the pairwise accuracy results decline as the penalty of false negatives goes larger.

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 that is mainly due to the dependence of features in dataset III. Shown in Figure 4.6b, 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, in terms of inferring the 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 in terms of the root-trigger correctness.

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 second) 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 seconds) 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.

4.6.6 Detection Accuracy on Datasets 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 4.7. 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 §4.6.9.

4.6.7 Evaluation of Stealthy Malware Activities

To test the effectiveness of rule- and learning-based methods, we assess our solution on several pieces of proof-of-concept and real-world malware.

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.

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 [4]. It sends out the HTTP `GET` or `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.

Detection of real-world malware. We find and investigate several pieces of real world malware, as well as the software bugs that may trigger HTTP requests.[3] 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)
- **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 bug 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. Both rule- and learning-based approaches 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.

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.

[3]Most samples are found at `packetstormsecurity.com`.

DNS Bot Detection. Botnet command and control channel using DNS tunneling [7] is extremely stealthy and difficult to detect [143]. 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., a DNS request to `d1js21szq85hyn.cloudfront.net`. We overlay the bot queries with a 2-hour DNS-HTTP traffic dataset, 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 root-triggers.

4.6.8 Detection Accuracy of Rule-based Approach

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 §4.6.3, 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 4.10 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.

Table 4.10: Percentages of triggering relations inferred by different subroutines of rule-based TRD on Dataset I.

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

We inspected the requests that are missing triggering relation (2.1% in Table 4.10) 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, and missing referrer information in request headers[4]). 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 4.8). In the results obtained from the rule-based method,

[4]Depending on the version of the browser and JavaScript engine, the referrer information in the request header may be suppressed [14].

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 time and 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.

4.6.9 Performance of Rule-based and Learning-based Approaches

All runtime results are obtained on a machine with Intel i5-3320 and 16GB RAM. First, we measure the runtime on the evaluation of dataset I to compare the rule- and learning-based approaches. Results show that it takes about 2.68 seconds to run TRD algorithm per 1000 requests, while it takes about 0.30 seconds to classify and infer the root-trigger per 1000 requests using a Bayesian network classifier.

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 4.11. Standard deviations are negligible and not shown.

Table 4.11: Averaged performance (in seconds) of Pairing, Training, Classification, and root finding algorithm.

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 feature extraction.

Shown in Table 4.11, the Training, Classification, and root finding algorithm are fast. The Pairing operation is the most time-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 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 3 minutes on average, indicating that our method is efficient for practical use.

4.6.10 Summary of Evaluations

We summarize our findings below.

- Bayesian network classifier gives the highest pairwise accuracy rate among all datasets. It outperforms the naive Bayes classifier, indicating the existence of dependencies in pairwise features. The accuracy can be improved by strategically defining the cost matrix. 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 causality 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 our proof-of-concept browser spyware, DNS bot, stand-alone data-exfiltrating malware, as well as several real-world vulnerabilities.

- 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.

- *Limitations.* In our optimized learning-based 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 trade-off in our future work.

4.7 Summary: Host-based Triggering Relation Discovery

Analyzing the triggering relation between network traffic and user activities has not been systematically investigated as a general approach for anomaly detection. Compared with the conventional signature-based and counting-based detection approaches, our traffic dependence model provides the semantic-aware security analysis. The analysis approach exploring request-level traffic structures and semantic triggering relations is new.

The proposed triggering relation graph extracts and models the causal relations between observed events and their triggers. Triggering relations exist in different types of network traffic. These discovered relations produce structural and contextual information for reasoning and justifying the occurrences of system and network behavior patterns. Moreover, they help one understand the context of network communications and identify the security threats. The advantage of our approach is the ability to provide proactive system defenses.

In this chapter, we describe the applications of our TRG model 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. Machine learning approaches enable us to cover a high dimension of features, which cannot be realized by simple rule-based approaches. In the learning-based approach, the pairing operation is *unique* and extracts the pairwise features, which make the binary classification feasible in discovering triggering relation among network requests. With a specific user intension based policy, we leveraged TRG as a tool to identify the potential threats.

We performed extensive experimental evaluation on our host-based solution. Our results indicate the feasibility of enforcing the network traffic dependencies. Our evaluation on 10+ GB data (real-world and DARPA datasets) shows 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 bots, and data exfiltrating malware. For future work, we plan to design more complex security definitions and policies for utilizing the triggering relation graph to detect network incidents.

Chapter 5

Android-based Network Anomaly Detection via Triggering Relation Discovery

Malicious Android applications pose serious threats to mobile security. They threaten the data confidentiality and system integrity on Android devices. Code obfuscation is commonly seen in malicious apps, therefore monitoring runtime activities serves as an important way to analyze the dynamic app behaviors. In this chapter, we utilize the triggering relation model for dynamically analyzing network traffic on Android devices.

The chapter is organized as follows. We describe the motivations and an overview of our approach in the context of mobile security in §5.1. The new challenges on Android platforms and our design goal are given in §5.2. We present the details of our dependence analysis approach in §5.3. To label the triggering relation on general apps, we introduce a delay inject technique in §5.4. We conduct extensive evaluations on our solution and present the results in §5.5. §5.6 concludes the chapter.

5.1 Motivation and Overview

The openness of Android application development mechanism poses security challenges to smartphone users. Malicious apps (malware) may be created by repackaging popular apps. At runtime, they may directly fetch and run code on-the-fly without the user's awareness [66]. Thereafter, malicious apps may spy on the victim users, stealthily collect and exfiltrate user's information.

Existing static analysis solutions inspect the source code, binaries or call sequences for anomalies [19, 24, 63, 95, 148, 153]. For example, Drebin [24] extracts features including APIs, permissions, and app components to characterize and classify apps. SCSDroid [95] identifies malicious apps by extracting subsequences of system calls. ViewDroid [153] detects repackaging apps by analyzing the bytecode of Android apps. However, dynamic code loading, Java reflection-based method invocation, data encryption, and self-verification of signatures are commonly seen in the malware code [52, 147]. These code obfuscation techniques make static analysis based detection challenging. Dynamic analysis, as a complementary to the static analysis, detects the runtime behaviors of the malicious apps. For example, TaintDroid [51] is a solution to monitor and verify the sensitive information flows through the apps, but it is infeasible to learn the origin of sensitive data leaking (e.g., triggered by users or malicious code fetching from a remote host). Besides, as we treat the apps as blackbox, our solution is lightweight and low overhead, as opposed to monitoring the apps as whitebox in TaintDroid.

An important technical enabler of our solution is the ability to discover the triggering relation of pairs of requests. We refer to it as the *pairwise* triggering relation discovery. Pairwise relations are used to construct the *full* triggering relation graph. The graph depicts the causality of network requests and allows one to quickly identify the root triggers of observed events. We utilize learning-based classifiers to predict the pairwise relationship between two network requests. Thereafter, we classify the *root triggers* to identify malicious requests based on their dependency features. The new capability of our solution is to distinguish malicious root triggers from legitimate ones. It allows us to detect the stealthy malware activities that may not be detected by existing traffic profiling methods [48, 137, 146].

Our work is designed to fill the gap between dynamic analysis and network anomaly detection on Android. The dependence model inferring the relation of network requests aims at monitoring the runtime *network behaviors* and reasoning them based on the user's intention.

Our contributions are summarized as follows.

- We present a two-stage learning-based solution to detect the malicious network activities. The discovery of triggering relations on pairwise network events is unique and enables us to construct a triggering relation graph. Our pairing operation is improved with an adaptive window technique. Then we classify the root-triggers to identify the abnormal requests.
- We conduct experiments based on 14GB real-world network and system data. The results show that the triggering relations of network traffic on Android can be inferred at the accuracy of 96.8% to 98.2%.
- Without knowing any *priori* knowledge, our framework enables the detection of malicious requests sent from the newly released apps. Results show that we detect 99.1% malicious requests from all malicious apps. We identify all 12 malicious apps that behave like Trojan, spyware, or bots. In comparison, only one is reported by our organization IDS (`FireEye`).

Our results show that our detection method is useful for identifying unauthorized network activities sent from the malicious apps. The significance of our work is that it provides insights of the network traffic dependency and demonstrates the use of structural and semantic information in reasoning about network behaviors and detecting stealthy anomalies.

5.2 Design Goals

In this chapter, we aim at identifying malicious apps by analyzing their dynamic behaviors, specifically the network traffic. Our goal is to profile the legitimacy of the mobile traffic pattern of benign apps, and thus *to detect the malicious network requests that are sent without user's awareness*.

We utilize the proposed *triggering relation* model (§3) to formalize the causal relations of app-generated network requests in Android. The model depicts the dependency of network events and explains how one triggers the other. The dependencies and semantic information of requests are useful for human experts' cognition, reasoning, and decision making in cyber security. We use the dependency knowledge to determine whether a network request is triggered by legitimate causes.

To detect malicious requests on mobile devices is challenging, as the diverse network traffic from different types of apps (e.g., irregular requests without appropriate referrer in the header) and frequently-generated notification. Existing solutions either reason about the dependency using heuristic algorithms [141] or limit the causality analysis on one specific application (e.g., browsers [107]), which cannot achieve our goal to dynamically analyze network behaviors on Android.

In this section, we discuss the challenges of extracting network traffic dependency on Android devices (§5.2.1) and describe the overview of our analysis approach (§5.2.2).

5.2.1 New Challenges for Triggering Relation Discovery on Mobile Devices

The problem of detecting malicious requests on mobile devices is challenging. We summarize the technical difficulties on the Android platform as follows.

- **Lack of referrer.** In our study, 83% network traffic sent from Android device is generated by non-browser apps. The app developers create diverse apps that communicate with the remote servers via HTTP protocol. However, the HTTP requests sent from non-browser apps rarely contain correct and meaningful referrers. Some apps attach unified referrer information for all outbound requests. Therefore, existing solutions including the referrer-based inference [141] or the instrumented browser [107] do not work for the general apps.

- **Diverse network traffic from apps.** Compared with browser-generated traffic, requests sent from apps have varying patterns. How to use a unified framework to infer the triggering relationship has not been studied yet. Processing massive data also demands scalable solutions.

- **Automatic notifications and updates.** Mobile devices constantly alert users with notifications, resulting in plentiful network requests in the background. The tradition whitelist generated by human experts is no longer adequate. How to distinguish benign updates from stealthy malicious requests is challenging. Properly handling the automatic notifications is crucial in achieving a high accuracy and low false positives.

We infer *triggering relations* by analyzing how the delays are propagated among the network requests of an app. As a result, we discover the dependency without requiring the referrer information. We also design a learning-based approach to classifying the triggering relationship of requests, which is scalable and suitable for all types of Android apps. We build the TRG based on classification results. The graph offers rich structural and context-aware features that are further used to distinguish the automatically generated notifications and malicious requests.

Our model detects the malicious behaviors by discovering the dependency of network requests on Android devices. Shown in Figure 5.1, the behaviors of malware include the unauthorized network activities, e.g., stealthy outbound requests without user's awareness, piggyback requests for updating apps with malicious code, and other types of out-of-order requests. These behaviors existing in a wide range of malware families cause sensitive information leaking and system abusing, e.g., repackage apps, drive-by download apps, and Android bots (see §3.3).

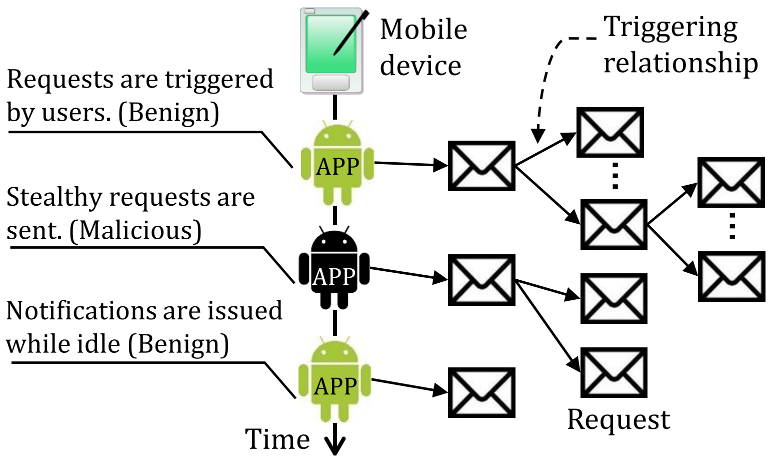

Mobile device
Triggering relationship
Requests are triggered by users. (Benign)
APP
Stealthy requests are sent. (Malicious)
APP
Notifications are issued while idle (Benign)
APP
Time
Request

Figure 5.1: Our TRG discovery aims to detect stealthy outbound network activities from malicious apps in Android.

Our model targets at the stealthy network activities via HTTP, because HTTP is the protocol of choice for most app developers to implement communication with remote servers [146] and is hardly blocked by anti-virus tools. Repackaged apps and drive-by download attacks are the common initial infection vectors [161]. After the malicious apps are installed, the requests sent to remote hosts could leak user's information or conduct bot activities for profits. Our proposed solution can detect these types of activities without knowing any signatures of the malware, and thus applicable to detect the new (zero-day) malicious apps. Trivial engineering on existing malicious apps cannot bypass the detection, as our solution identifies the requests without benign triggers, and is not based on any known malicious code or sensitive APIs.

We target at the malicious apps running on the user-space. Therefore, malware with root access is not in our threat model. We trust the system logs and keylogger apps to infer the triggering relationship and root-triggers.

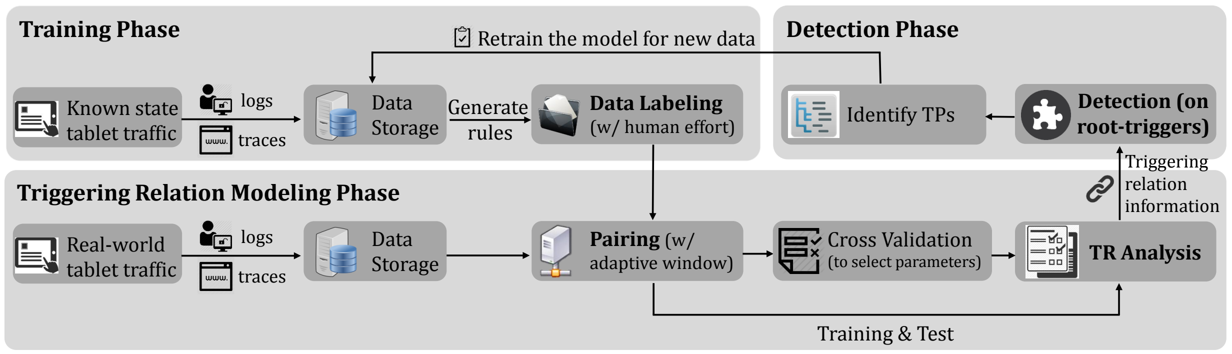
5.2.2 Overview of Our Approach

As explained in §3, the triggering relation graph is composed of network events and the edges that link them. The problem of *triggering relation discovery* on a set of network events can be transformed into inferring the triggering relations of pairs of events, which is defined as the *pairwise triggering relation*. Given a sequence of network requests $\mathbb{R} = \{r_1, r_2, \ldots, r_n\}$, the purpose of *pairwise comparison* is to generate a set of pairs $\mathbb{P} = \{P(r_i, r_j)\}$, where $1 \leq i < j \leq n$, r_i and r_j have a *high* likelihood to have a triggering relation. The pairwise comparison method has been proposed to solve the general relation discovery problem [64, 81, 157]. In this work, we further advance it in the Android context by improving the pairing efficiency and the multinomial classification.

Utilizing predicted results from the classification, we build the TRG $\mathbb{G} = \{T_1, T_2, \ldots, T_m\}$, each T_i is a tree rooted at request r_{t_i}. We classify the root-triggers $\mathbb{R}_{\mathbb{T}} = \{r_{t_i}\}$ ($1 \leq i \leq m$), based on the extracted features from TRG $\mathbb{G}$. The *root-trigger* request r_{t_i} determines the legitimacy of each tree, i.e., if it is generated from a benign app, the entire tree T_i is legitimate. The anomalies are the requests of trees that are not triggered by users, nor belong to the automatic notifications/updates from benign apps.

An overview of workflow is shown in Figure 5.2. Our approach requires the data labeling (in training phase) based on the requests $\mathbb{R}$, details of which are given in §5.4.

The main operations in our analysis are pairing, TR analysis and detection. The first two operations are designed to model the triggering relations and build a TRG. Both TR analysis and detection include training and testing, as the standard operations for the machine learning approach. Specifically, we adopt a multinomial classification in TR analysis and use a binary classifier in detection.



Training Phase
Retrain the model for new data
Detection Phase
Known state tablet traffic
logs
traces
Data Storage
Generate rules
Data Labeling (w/ human effort)
Identify TPs
Detection (on root-triggers)
Triggering relation information
Triggering Relation Modeling Phase
Real-world tablet traffic
logs
traces
Data Storage
Pairing (w/ adaptive window)
Cross Validation (to select parameters)
TR Analysis
Training & Test

Figure 5.2: The workflow of our triggering relation discovery based malware detection.

A) Pairing takes the $\mathbb{R}$ as an input and outputs a set of paired requests $\mathbb{P} = \{P(r_i, r_j)\}$. (§5.3.1)

B) TR analysis takes the predicted pairs P as inputs and outputs a constructed TRG $\mathbb{G}$. (§5.3.2)

C) Detection takes a TRG $\mathbb{G}$ and its root-trigger set $\mathbb{R}_{\mathbb{T}}$ as inputs. It extracts features of $\mathbb{R}_{\mathbb{T}}$ and builds classifications to predict the legitimacy of root-triggers. (§5.3.3)

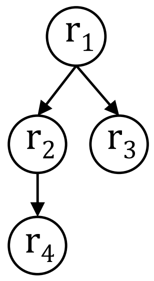
5.3 Triggering Relation Modeling and Detection

We describe our triggering relation modeling phase in this section. The goal of this modeling is to build a complete and accurate TRG for later detection phase.

We introduce the multi-class to describe the structural relations of two requests r_i and r_j: (1) `parent-child`: $r_i \rightarrow r_j$, (2) `sibling`: r_i and r_j have the same parent (i.e., $r_k \rightarrow r_i$ and $r_k \rightarrow r_j$), (3) `same-tree`: r_i and r_j are situated on the same tree ($r_i \in T_i$ and $r_j \in T_i$) without `parent-child` and `sibling` relations, and (4) `no-relation`: r_i and r_j on two different trees. The quad-class offers rich information to identify the relations for requests in TRG. Figure 5.3 illustrates how we generate pairs from a sample tree using the quad-class labeling and how the quad-class relations determine the pattern of a tree in TRG.

5.3.1 Pairwise Feature Extraction

To solve the *pairwise triggering relation* discovery, one needs to make pairs for network requests that may have a triggering relationship. The pairing operation is performed on any two requests r_i and r_j in $\mathbb{R}$ for whose time difference is less than a threshold τ. Let the request r_i have p attributes and each is denoted as $r_i.attr$, $attr \in \{\texttt{time}, \texttt{IP}, \texttt{port}, \texttt{host}, ..., \texttt{URL}\}$. Then, the pairing



(a)

#	Paired items	Relation labels
(1)	$P(r_1, r_2)$	`Parent-child`
(2)	$P(r_1, r_3)$	`Parent-child`
(3)	$P(r_1, r_4)$	`Same-tree`
(4)	$P(r_2, r_3)$	`Sibling`
(5)	$P(r_2, r_4)$	`Parent-child`
(6)	$P(r_3, r_4)$	`Same-tree`

(b)

Figure 5.3: A sample tree (a) and its pairing results (b). In (a), nodes and edges denote the network requests and their triggering relations respectively.

operation is performed on every attribute of the requests. The pairwise comparison can be written as $P(r_i, r_j) = \{f_1(a_i.\texttt{time}, a_j.\texttt{time}), \ldots, f_p(a_i.\texttt{URL}, a_j.\texttt{URL})\}$, where f_k is the pairing function for the k-th attribute. Shown in Table 5.1, different types of attributes require different pairing strategies (f_k). Specifically, string type of attributes usually contains rich semantic information. Therefore, we utilize the normalized edit distance to compute the similarity of two string fields.

Table 5.1: Pairing functions used for different attribute types.

Attribute	Pairwise Function (f_k)	Example(s)
Numerical	Compute the difference.	Timestamp.
Nominal	(Sub)string/equality test.	Protocol, File type.
String	String similarity comparison.	Host, request URL.
Composite	Comparison of sub-attributes.	IP address.

Adaptive Pairing. To avoid the $O(n^2)$ complexity of pairing n requests, we only analyze network requests whose timestamps are within a small time window. In our prototype, we calculate the threshold using the *Chebyshev's inequality*. Let $d_{(i,j)}$ be the time difference between a pair of requests r_i and r_j that have a triggering relationship. During the learning phase, μ and σ^2, the mean and standard deviation (SD) of $d_{(i,j)}$ can be estimated.

$$\Pr(|d_{(i,j)} - \mu| \geq 10 * \sigma) \leq 0.01 \tag{5.1}$$

As shown in Equation 5.1, we obtain that the probability of the difference between $d_{(i,j)}$ and μ exceeds $10 * \sigma$ is no greater than 1%. The Chebyshev's inequality offers an upper bound on the probability of $d_{(i,j)}$, regardless of its distribution. The adaptive pairing window ($\tau = \mu + 10 * \sigma$)

outperforms the predefined and fixed threshold proposed in [156, 157], as it is difficult to find a fixed value on mobile devices due to the diverse network behaviors of apps.

In our solution, we calculate the moving average as the μ and its corresponding σ^2 in Equation 5.1. The moving average smooths the window size, as the network data within a small time interval may fluctuate. The diurnal patterns for mobile users can be adjusted by varying the time windows size (e.g., last day, last week, etc). Later, we confirm that our adaptive pairing using 3-day moving average guarantees the coverage of dependency pairs by 99%.

5.3.2 Quad-class Classification and TRG Construction

In the TR analysis operation, we train classifiers to predict triggering relations on pairs $\mathbb{P} = \{P(i,j)|i < j\}$. The predicted results are in the form of $(r_i, r_j) \rightarrow l_{ij}$, where l_{ij} is one of the relations in the quad-class. Then, machine learning classifiers (e.g. decision tree-based, Bayesian-based) can be applied on the pairwise data $\mathbb{P}$.

The classification results of pairwise comparisons serve as the source for constructing the TRG. Essentially, this operation is the reverse process of Figure 5.3. We analyze three types of relationship: `parent-child`, `sibling`, and `same-tree` and assign weights w_p, w_s, and w_t to them, respectively.[1] The purpose of the weights to construct the TRG based on the classified pairwise triggering relations. Given r_i and r_j, we define $w_{(r_i,r_j)}$ in Equation 5.2 as a score of the pair (r_i, r_j), which is the summation of the weights that indicate their triggering relationship.

$$w_{(r_i,r_j)} = \sum w \begin{cases} w_p, & \text{if } (r_i, r_j) \rightarrow \texttt{parent-child}. \\ w_s, & \text{if } (r_i, r_k) \rightarrow \texttt{parent-child} \\ & \quad \text{and } (r_k, r_j) \rightarrow \texttt{sibling}. \\ w_t, & \text{if } (r_i, r_j) \rightarrow \texttt{same-tree}. \end{cases} \tag{5.2}$$

We show the procedure to calculate the weight for the classified pair (r_i, r_j) in Algorithm 4. The inputs of Algorithm 4 are the predicted results $\mathbb{P}$ and the weights (w_p, w_s, and w_t). In lines 6-9, there may be multiple triggering candidates (parents) for one request. Thus, we update the weights for each node of its parents (denoted by Parent(r_i)). The output of Algorithm 4 is a dictionary of keys $\mathbb{D}$, which contains the calculated weights of pairs that (may) have triggering relationship.

Building a TRG from $\mathbb{D}$ is equivalent to finding the *maximal spanning tree* in a graph. In our approach, we adopt Prim's algorithm, which suggests starting from one vertex and selects the

[1] In our design, the weights are calculated using multivariable linear regression.

Algorithm 4 Triggering Relation Weights Calculation

```
Input: Requests $\mathbb{R} = \{r_1, r_2, \dots, r_n\}$, classified pairs $\mathbb{P} = \{(r_i, r_j) \rightarrow p_{ij}\}$ ($1 \le i < j \le n$), and weights $w_p, w_s, w_t$.
Output: A dictionary of keys $\mathbb{D} = \{(r_i, r_j) = w_{(r_i,r_j)}\}$, where $r_i$ and $r_j$ may have triggering relationship according to the classification, and its weight is $w_{(r_i,r_j)}$.
 1: define $\mathbb{D}$ to store the weight $w_{(r_i,r_j)}$ for the edge $r_i \rightarrow r_j$
 2: for each pair $(r_i, r_j) \rightarrow p_{ij} \in \mathbb{P}$ do
 3:   if $p_{ij} = $ parent-child relation then
 4:     weight $w_{(r_i,r_j)}$ += $w_p$, and update $w_{(r_i,r_j)}$ in $\mathbb{D}$
 5:   else if $p_{ij} = $ sibling relation then
 6:     Parent($r_i$) $\leftarrow$ $r_i$'s trigger(s) according to $\mathbb{P}$
 7:     for $r_p$ in Parent($r_i$) do
 8:       weight $w_{(r_p,r_j)}$ += $w_s$, and update $w_{(r_i,r_j)}$ in $\mathbb{D}$
 9:     end for
10:   else if $p_{ij} = $ same-tree relation then
11:     Root($r_i$) $\leftarrow$ $r_i$'s root request(s) according to $\mathbb{P}$
12:     for each $r_r$ in Root($r_i$) do
13:       weight $w(r_r, r_j)$ += $w_t$, and update $w_{(r_i,r_j)}$ in $\mathbb{D}$
14:     end for
15:   end if
16: end for
17: return $\mathbb{D}$
```

highest weight from the adjacent neighbors.[2] Prim's algorithm fits our needs of starting from known root-triggers. The prerequisite in Prim's algorithm is the weight calculation. During the process of selecting the *largest* weight in Prim's algorithm, there may be equal weights in $\mathbb{D}$. We design rules to select the best triggering parent for a given request from a group of candidates whose weights are equal, i.e., $w_{(r_i,r_k)} = w_{(r_j,r_k)}$. The following rules are presented in the order of the precedence.

Rule 1) If $r_i \in T_i$, $r_j \in T_j$, and $r_{t_i}.\texttt{time} > r_{t_j}.\texttt{time}$, then $r_i \rightarrow r_k$. (The rule is in favor of the request on the most recent dependency tree.)

Rule 2) If $r_i, r_j \in T_i$, $r_i.\texttt{level} < r_j.\texttt{level}$, then $r_i \rightarrow r_k$. (The rule is in favor of the request on upper level of the tree.)

Rule 3) If $r_i, r_j \in T_i$, $r_i.\texttt{level} = r_j.\texttt{level}$, $r_i.\texttt{time} > r_j.\texttt{time}$, then $r_i \rightarrow r_k$. (The rule is in favor of the most recent requests on the same level of the tree.)

[2]Prim's algorithm is usually used to find the minimal spanning tree. However in our scenario, the larger weights mean the higher possibility to have triggering relationship. Therefore, we find the maximal spanning tree for $\mathbb{G}$. The same situation applies when using Kruskal's algorithm.

We make these design choices based on the empirical experience and domain knowledge of the Android network traffic. Prim's algorithm outputs the TRG $\mathbb{G}$ by picking the best candidate trigger from the equal-weight candidates. As the classification results may contain redundant or conflicting information, it is indispensable to solve the tie-breaking of weights. Wrong triggering relationship results in the wrong root-trigger, which causes the false positives of malware detection.

Table 5.2: The triggering relations are built from the predicted results. The tie-break column shows the rule to resolve ties.

#	Predicted results	$w_{(r_i,r_j)}$	Tie-break	Built TR
(1)	$(r_1, r_2) \rightarrow$ `p-c`	$w_p + w_s$	—	$r_1 \rightarrow r_2$
(2)	$(r_1, r_3) \rightarrow$ `p-c`	$w_p + w_s$	No tie.	$r_1 \rightarrow r_3$
(3)	$(r_2, r_3) \rightarrow$ `sib`	0		
(4)	$(r_3, r_4) \rightarrow$ `p-c`	w_p	—	$r_3 \rightarrow r_4$
(5)	$(r_3, r_5) \rightarrow$ `p-c`	w_p	r_3 is on upper level than r_4.	$r_3 \rightarrow r_5$
(6)	$(r_4, r_5) \rightarrow$ `p-c`	w_p		

Note: `p-c` and `sib` denote the `parent-child` and `sibling` relations, respectively. $w(r_i, r_j)$ shows the calculated weights using Algorithm 4.

We illustrate our TRG construction using an example in Table 5.2. First, we calculate the weights by parsing the predicted results. According to formulas (1-3), we observe that r_1 has parent-child relation with r_2 and r_3. Furthermore, the knowledge from formula (3) is redundant, only formulas (1-2) are enough to deduce the triggering relations. In the latter formulas (4-6), the parent of request r_5 has two candidates r_3 and r_4 based on the calculated weights. We resolve this conflict by using our tie-breaking rules.

The complexity of Algorithms 4 is O($|\mathbb{P}|^2$). Due to the sparse triggering relations in a TRG (i.e., $|node| \approx |edge|$), the complexity of Prim's algorithm is O($n * log(n)$), where n is the number of requests in $\mathbb{R}$.

5.3.3 Detection on Root-triggers

In the detection phase, we identify the root triggers $\mathbb{R}_\mathbb{T}$ for each network request based on the constructed TRG $\mathbb{G} = \{T_1, T_2, \dots, T_m\}$. Most requests are directly or indirectly triggered by the legitimate user's inputs ($\mathbb{U}_\mathbb{T}$). However, there may exist requests that cannot be linked with any user's activities, e.g., notification requests from benign apps ($\mathbb{A}_\mathbb{T}$), malicious requests from the malware ($\mathbb{M}_\mathbb{T}$). Therefore, $\mathbb{R}_\mathbb{T}$ can be formalized as the disjoint sets of the three.

$$\mathbb{R}_{\mathbb{T}} = \{\mathbb{U}_{\mathbb{T}} \cup \mathbb{A}_{\mathbb{T}} \cup \mathbb{M}_{\mathbb{T}}\} \tag{5.3}$$

In our design, we use binary classifiers to detect the malicious requests ($\mathbb{M}_{\mathbb{T}}$) in two steps: (1) In a clean/labeled dataset, we distinguish automatic notifications or update requests ($\mathbb{A}_{\mathbb{T}}$) from other benign ones ($\mathbb{U}_{\mathbb{T}}$). Based on the $\mathbb{A}_{\mathbb{T}}$ set, we can generate a whitelist for facilitating the future detection. (2) In a real-world dataset, we filter out the requests that belong to the whitelist, and then differentiate the malicious requests ($\mathbb{M}_{\mathbb{T}}$) from the benign ones ($\mathbb{U}_{\mathbb{T}}$).

For a given root-trigger $r_t \in \mathbb{R}_{\mathbb{T}}$, we extract features as described in Table 5.3. We train and classify the data using common supervised machine learning classifiers (e.g., random forest, logistic regression). The output of the classification in detection is the partition of $\mathbb{R}_{\mathbb{T}}$.

Table 5.3: Features extracted for the detection operation.

Type	Feature Definition / Description
$\mathcal{F}_1$	$\mathcal{F}_1$ describes the number of similar requests in the previous user-triggered trees. $f(r_t) = \|r\|$, where $r \in \mathbb{U}_{\mathbb{T}}$, $r_t.\texttt{time} - r.\texttt{time} \leq \tau^{*\,\dagger}$, $f_u(a_r.\texttt{attr}, a_{r_t}.\texttt{attr}) \leq \tau_u{}^{\ddagger}$
$\mathcal{F}_2$	$\mathcal{F}_2$ describes the number of similar requests in the previous non-user-triggered trees. $f(r_t) = \|r\|$, where $r \in \{\mathbb{A}_{\mathbb{T}} \cup \mathbb{M}_{\mathbb{T}}\}$, $r_t.\texttt{time} - r.\texttt{time} \leq \tau^{*\,\dagger}$ $f_u(a_r.\texttt{attr}, a_{r_t}.\texttt{attr}) \leq \tau_u{}^{\ddagger}$
$\mathcal{F}_3$	$\mathcal{F}_3$ describes the *statistics* from the previous user-triggered trees in $\mathbb{G}$. $f(r_t) = \|r\|$, where $r \in \mathbb{U}_{\mathbb{T}}$, $r_t.\texttt{time} - r.\texttt{time} \leq \tau^{*\,\dagger}$
$\mathcal{F}_4$	$\mathcal{F}_4$ describes the *temporal* relation of r_t and previous events. E.g., u_p is the most recent user event prior to r_t, then $f(r_t) = \|r_t.\texttt{time} - u_p.\texttt{time}\|$.

† τ^* is a time threshold used in feature extraction (not the τ for pairing). τ^* could be 1 second, 1 minute, 1 hour, 12 hours and etc.

‡ f_u is the function to compute the similarity between attributes. $a_r.\texttt{attr}$ denotes a request r's (semantic) features, e.g., `Host`, `Referrer`, `request URL` and `destination IP`.

In this operation, the dependency-based feature set $\mathcal{F}_{\mathcal{D}} = \{\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4\}$ contain the rich semantic and structural information of $\mathbb{G}$. The design of $\mathcal{F}_{\mathcal{D}}$ features is based on our empirical observation on the malicious and benign requests. Mobile users frequently visit websites and some benign requests are cached to save the bandwidth. Therefore, the newly generated requests may miss legitimate referrers due to caching and $\mathcal{F}_1$ features are extracted to characterize them as benign. Because Android apps may pull notifications from the remote servers periodically, the IP, host and referrer information of the notifications have usually seen before. $\mathcal{F}_2$ features are used to characterize the automatically-generated benign requests. $\mathcal{F}_3$ and $\mathcal{F}_4$ features are designed to quantify the pattern of $\mathbb{U}_{\mathbb{T}}$ in $\mathbb{G}$. The heuristics behind these features is that: *1)* requests are usually sent during the highly interactive periods; *2)* user-trigger requests are inclined to be sent immediately after the user's actions, while the idle between two root-trigger requests is a noticeably long interval.

Retraining. To better use the model, one can generate a whitelist based on the classified results (e.g., $\mathbb{A}_{\mathbb{T}}$). In the later detection, our approach saves the processing power and improves the accuracy by removing the confirmed benign requests. This stage may require human efforts to verify the whitelist entries before retraining, as shown in Figure 5.2.

5.3.4 Security Analysis and Limitations

Our dependence analysis model aims at analyzing the network activities of Android devices, while it has certain limitations and possible evasions. Our discussions are as follows.

- *Non-HTTP protocol.* Our solution targets at the stealthy network activities via HTTP, because HTTP is the protocol of choice for most app developers to implement communication with remote servers [146] and is hardly blocked by anti-virus tools. By combining other security tools and policies, one can set up firewall rules for apps that use other protocols.
- *Data authenticity.* The attacker may forge the user and system events so that the malicious requests have triggers. Advanced techniques (e.g., [20, 104, 113]) can be used to ensure the authenticity of the user inputs and system events.
- *Rooting device.* Our prototype requires rooted mobile devices and the monitored apps run in the user-space. This requirement may reduce the usability for non-tech savvy users.
- *Encrypted traffic.* For encrypted network activities sent via `HTTPS`, one might adopt an authorized proxy to decrypt and analyze the traffic. This method requires the device to install a self-signed CA to encrypt the network packets [117].

5.4 Labeling Triggering Relations on Mobile Data

A technical challenge in our work is the lack of readily available labeled data, i.e., network events with labeled triggering relations. We spend a substantial amount of effort designing methods to label the triggering relation of general apps. We present an approach for inferring the dependencies based on the timing perturbation, which delays one request and see if others would be affected. The rationale behind this approach is that the delay of an individual request will be propagated to the requests that are triggered by it.

In this section, we elaborate the approach to labeling the triggering relations (data labeling operation in Figure 5.2). This process is time-consuming and needs lots of human efforts. Therefore, the time perturbation method is suitable for generating rules on small-scale data, which fits the needs of labeling and training purposes. In comparison, our learning-based approach described in §5.3 can be used for analyzing and testing on large-scale data.

5.4.1 Triggering Relation Discovery with Timing Perturbation

To begin with, we first review the major temporal factors when sending out requests from an app on a mobile device. Given two requests r_i and r_j, where r_i triggers r_j, the time difference $d_{(i,j)}$ can be decomposed to several components: (1) DNS query time to obtain the IP address, (2) TCP connection time and one RTT (round-trip time), (3) network delays (e.g. queuing delay), and (4) processing time at both server and client sides.

Any of the factors may impact the discovery of triggering relation for general Android apps. As we have limited visibility into the network- and server-side factors, we use the time sequence of outbound packets to estimate the causality by creating temporal perturbations. We use the time sequence of outbound packets to estimate the triggering relations by creating temporal perturbations. To distinguish the artificially-generated delays from other factors and random noise, *we take advantage the statistical tests to achieve high levels of confidence*. Our dependence analysis and Rippler [152] operate at different levels of granularity: we detect dependencies at the level of outbound requests, while Rippler works at the level of network services.

Our dependency inference consists of three steps. In step (i), we record $\mathbb{R} = \{r_1, r_2, \ldots, r_n\}$, a series of network traffic generated from one app. The list $\mathbb{R}$ serves as a baseline, where the elements in $\mathbb{R}$ are ordered by their timestamps. In step (ii), we re-act the scenario and delay t milliseconds for each object in $\mathbb{R}$. This step is based on the fact that r_j is a dependent of r_i, if and only if r_j is not loaded until r_i is fetched. For step (i) and each iteration in step (ii), we perform m times to obtain m observations.

In step (iii), we apply statistical tests to the m observations for identifying the triggering relations. We define $X_{(j,i)}$ as the variable for the outbound timestamp of j-th packet when the i-th packet is delayed ($i = 0$, for the baseline in Figure 5.4). We denote $\mu_{(j,i)}$ and $\sigma_{(j,i)}$ as the mean and SD of $X_{(j,i)}$. Because the delay injection is independent of each iteration and the samples are from a normal distribution, we adopt the two-sample pooled t-test [124] to figure out whether the j-th

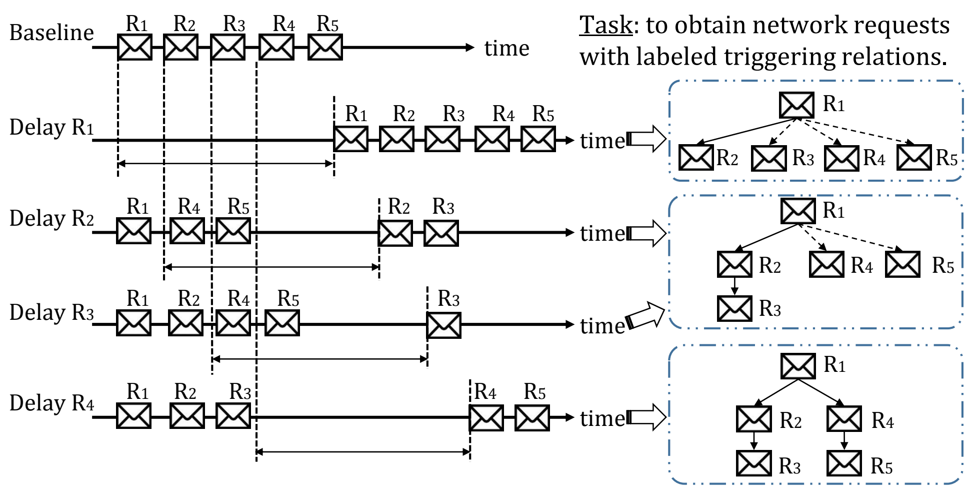

Baseline
Delay R1
Delay R2
Delay R3
Delay R4
R1
R2
R3
R4
R5
time
Task: to obtain network requests with labeled triggering relations.

Figure 5.4: We infer the triggering relation of requests using the timing perturbation and incrementally build the TRG. The dashed lines denote the intermediate results and the solid lines denote the finalized triggering relationship.

packet is affected by the i-th packet. The null hypothesis is $H_0 : \mu_{(j,0)} = \mu_{(j,i)}$. We report the j-th packet is a dependent of the i-th packet, once the statistical test rejects H_0. The TRG can be incrementally built based on the finding of dependency from temporal perturbations, as shown in Figure 5.4.

We assume the requests $\mathbb{R}$ are not changed during our study. However, we do observe some requests from an app may slightly differ between the iterations. Therefore, we compute the edit distance and regard two URLs identical if the similarity is less than a threshold. Furthermore, we confirm such a discrepancy rarely transforms the TRG to a different one.

5.4.2 Root-trigger Identification

To find the root-trigger is an essential task in identifying the normalcy of the requests, as the requests triggered by the root remain the same legitimacy in our model.

In our solution, we take advantage of the Android system debug logs (`Logcat`) and the `KidLogger` app [9] to accurately infer the root-triggers. The `Logcat` serves as a default means to view the system debug outputs, and thus can be used to identify if the request is triggered by user's requests or by system/software updates. We focus on the logging levels `information` and `debug`, as they provide information of the utilization, callers, and callees. The `KidLogger` app is designed to monitor and track on the Android system as a parental control tool. We obtain the following information by processing the `Logcat` and `KidLogger` data.

- Based on the logs from `Logcat`, `ActivityManager` is a major class that interacts with overall activities running on the Android device. It provides the information about when a particular service is awake and invokes network connections. We identify the notification requests by parsing the `ActivityManager` logs.
- When users interact with an app, `debug` logs reveal details on when and how the app responses. Integrated with the information from `Kidlogger`, we determine the foreground apps. Then, we correlate the root-trigger request with the closest user's activity prior to it.

5.5 Evaluation and Results

We implement a prototype to analyze the real-world network traffic on an Android device. Based on the collected data, we conducted extensive tests on our proposed dependence analysis solution. The questions we seek to answer are:

- How accurate is the prediction for triggering relations? (§5.5.4, §5.5.6)
- How accurate is the classification for detecting the notifications and malicious requests? (§5.5.5, §5.5.7)
- Can it detect *new* malicious Android apps? (§5.5.7)
- Is our approach scalable for analyzing real-world data? What is the performance of our approach? (§5.5.8)

5.5.1 Implementation

Our prototype implements all parts of the dependence discovery and detection system. The data are processed on a Linux machine with Intel i5-3320 and 16GB RAM.

Data collection and preprocessing. We root an Android device and install `PythonForAndroid` to run Python code. We use `tcpdump` to collect HTTP requests whose headers contain valid `GET` or `POST` request. To log the user's events, we write scripts to continuously monitor the files from `/data/input/event[0-7]`. We use `ps` and `netstat` commands to collect the process information and associate the network requests with a process. The collection and preprocessing are implemented with 1,400 lines of code in Python, C, and Bash.

Pairing and classification. We implemented the pairing, TR analysis and feature extraction for detection in Java using the Weka library with 1,200 lines of code. To build the TRG, we wrote 800 lines code in Python.

5.5.2 Experimental Dataset

The experimental data are obtained from a Nexus 7 tablet. The device, equipped with our data collecting code, is given to a graduate student of a university for regular daily use. The device comes with 28 bundled apps, and then we installed 36 popular apps via the official Android store [12]. We first collected the network and system logs for a continuous 72-day period. These 64 apps are regarded as benign and do not conduct malicious activities. Thereafter, we installed 24 malicious apps and collected data for 22 days. The malicious apps include repackaged apps that covertly fetch advertisement requests, drive-by download apps that install other apps when running, and spy apps that keep sending out host and user's sensitive information (e.g., bookkeeping). The total dataset is 14 GB.

Experiment setup. We conduct two experiments.

- **Dataset I**: We apply our approach on clean data only. The purpose is to evaluate its ability to recognize the automatically-generated notifications and updates from other benign requests.
- **Dataset II**: With the full dataset, we randomly insert the malicious data (by day) into the benign data and evaluate the effectiveness of our approach.

Dependency labeling of training data. We infer the triggering relation of browser-generated HTTP requests, according to a referrer-based method, which achieves nearly 100% accuracy [157]. To label the dependency of network traffic that comes from non-browser apps, we first installed the apps in the Android emulator. We set up a proxy and route the packets to a Linux host as a single point to control the outbound traffic from the emulator. On the proxy, we use `netem` to delay the traversing packets by adding the queuing delays, according to the scheme mentioned in §5.4. We select 15 popular apps and run the timing perturbation approach. We generate heuristic rules to label all the triggering relations, based on the learned patterns and domain knowledge. Examples are listed in Table 5.4, for given two requests r_i and r_j (r_i proceeds r_j), and a threshold τ.

With our hand-tuned rules, we discover the triggering relation for 86.1% of HTTP requests. The rest of the requests are labeled with the timing perturbation method.

5.5.3 Accuracy and Security Metrics

We apply three classifiers[3], namely random forest (`R-F`), C4.5 algorithm (`C45`), and logistic regression (`LOG`) in both triggering relation modeling and detection phases (in Figure 5.2). The operations are evaluated using the following metrics.

[3] These three classifiers are selected according to the 10-fold cross validation.

Table 5.4: The examples of summarized triggering relation rules using the time injection method. The attributes of request are extracted from packet headers and system events. f_{sim} is used to compare the similarity of two string fields.

No.	Rules (to decide that $r_i \to r_j$, i.e., r_i and r_j have triggering relationship.)	Related Apps
1	r_i.time - r_j.time $\leq \tau \wedge$ (r_i.PID = r_j.PID $\vee$ r_i.program = r_j.program) $\wedge$ r_i.host = r_j.referrer.	Traffic sent from browser apps.
2	r_i.time - r_j.time $\leq \tau \wedge$ r_i.program = r_j.program $\wedge$ r_j.type = `POST` $\wedge$ r_i.host = r_j.host $\wedge$ r_j.referrer = *null*.	Traffic sent from news, social, or education apps.
3	r_i.time - r_j.time $\leq \tau \wedge$ r_i.program = r_j.program $\wedge$ f_{sim}(r_i.host, r_j.host) = true $\wedge$ (r_i.referrer = r_j.referrer $\vee$ r_j.referrer = *null*).	Traffic sent from news, media, or education apps.
4	r_i.time - r_j.time $\leq \tau \wedge$ r_i.program = r_j.program $\wedge$ r_i.destinationIP = r_j.destinationIP $\wedge$ f_{sim}(r_i.host, r_j.host) = true $\wedge$ r_i.program = r_j.XRequestWith.	Traffic sent from news or media apps (using `AJAX`).

- **Pairing coverage** (PC). This metric is to evaluate if the (potential) triggering relation $r_i \to r_j$ ($i < j$) are paired using the adaptive pairwise window (size = τ).

$$PC = \frac{|\forall r_i, r_j \in \mathbb{R}, \text{s.t. } r_i \to r_j, d_{(i,j)} \leq \tau|}{|\forall r_i, r_j \in \mathbb{R}, \text{s.t. } r_i \to r_j|} \tag{5.4}$$

- **TR accuracy** (TRA): For each request $r \in \mathbb{R}$, we calculate the ratio of the number of requests correctly identified its trigger to the total number of requests, based on the ground truth. The metric evaluates the effectiveness of the classifiers.

$$TRA = \frac{|\forall r_j, \exists r_i \in \mathbb{G}, \text{s.t. } r_i \to r_j|}{|\forall r_j, \exists r_i \in \mathbb{R}, \text{s.t. } r_i \to r_j|} \tag{5.5}$$

- **Precision, Recall, F-score**: False negatives (FN) are the malicious requests being detected as benign ones. False positives (FP) are the benign requests being detected as malicious ones. We use the conventional metrics: *i)* false positive rate (FPR) and false negative rate (FNR), *ii)* precision and recall, and *iii)* F-score (the *harmonic mean* of precision and recall).

5.5.4 TR Analysis on Dataset I

Pairing. We calculate the adaptive pairing windows size for dataset I using the 3-day moving average value and standard deviation (SD). The pairing window size for each sampling day is shown in Figure 5.5. We observe that the average of the time difference $d_{(i,j)}$ is stable, though the SD fluctuates a lot. Therefore, we have a fluctuating adaptive pairing window size for each sampling day. We spot that the pairing window size is increased from the 51st to 53rd days. This prolonged window is due to a plural of `AJAX` calls from a browser app (`Chrome`). We confirm that the pairing coverage is nearly 100% and only a few dependent pairs are missed due to their extremely long delays. The high pairing coverage is guaranteed as explained in §5.3.1.

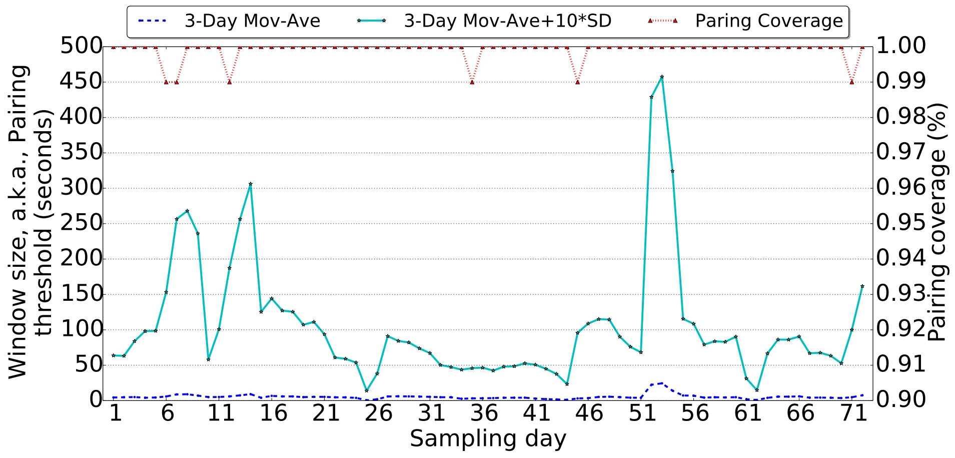


Figure 5.5: The adaptive pairing window size and its coverage rate for each sampling day on dataset I.

We find that `parent-child`, `sibling`, and `same-tree` relations account for 3%, 19% and 15% of the total number of pairs.[4] The rest of the pairs (62%) are labeled as `no-relation`.

Triggering Relation Modeling. We vary the sizes of training and test data using $i \in [1, 5, 10, 15, 20]$, i.e., we train the data for i continuous day(s) and test the data in the following i continuous day(s). Averaged results are reported until the data on the last day of dataset I is tested. We show the TR accuracy rates in Figure 5.6.

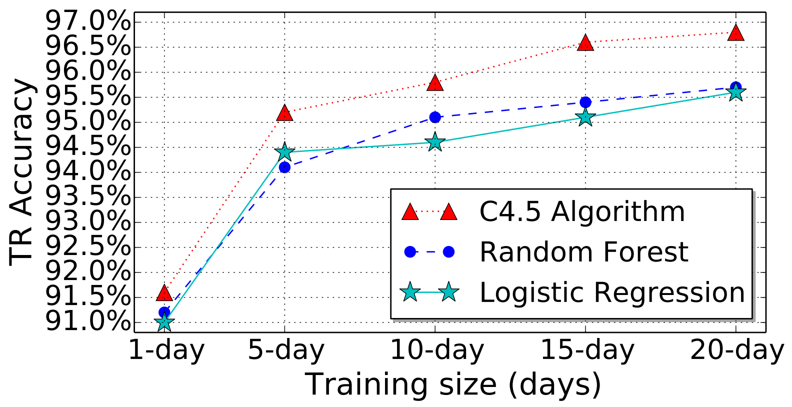


Figure 5.6: The TR accuracy of dataset I.

All three classifiers achieve the higher TR accuracy using larger training datasets. The C4.5 algorithm outperforms the other two in all settings and it achieves the best TR accuracy at 96.8%. For

[4]The number of `sibling` relation is greater than that of `same-tree`, which is due to the shallow tree structure formed by the Android network requests.

the wrongly predicted cases, we found that there are two main reasons that cause misclassification.

- When users view multiple similar webpages (e.g. price comparison on shopping sites), similar requests are sent to the same host from different tabs on browser apps. These requests are likely to be predicted to wrong triggers, because users may switch the tabs frequently and shortly. As this only happens on browser apps, a finer-granularity solution on tab-level events can reduce the wrong predictions.
- Some requests from the `news` apps tend to have a long delay between push notifications. The unconventionally long delay hinders the inference of triggering relations. A possible solution is to preprocess the time differences using advance techniques (e.g., log transformations).

5.5.5 Recognition of Auto-generated Traffic

The purpose of this experiment is to evaluate our ability to recognize the automatically-generated notifications and updates from other benign requests. This experiment helps us further reduce false positive (false alerts). We evaluate the classification based on the results of running the C4.5 algorithm in §5.5.4.

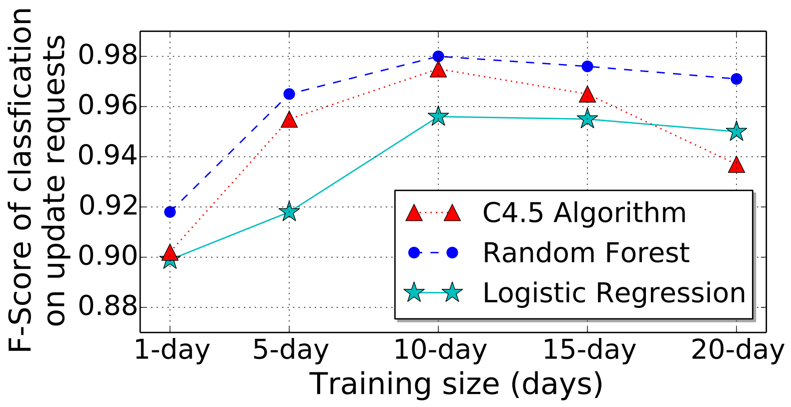


Figure 5.7: The F-scores of classifying root-trigger requests on dataset I.

In Figure 5.7, we plot the F-score of the three classifiers on classifying the update requests. The random forest classifier outperforms the other two classifiers and it maintains a relatively high and steady accuracy. A peak is observed in all classifiers, and the F-score reaches its highest value using 10-day training size. This phenomenon is due to the constant changes of app and user behaviors. Therefore, the size of training windows is a trade-off between prediction accuracy and coverage. The prolonged training windows are not effective in characterizing the root-triggers from recently active apps.

Whitelist generation. We generate a list of HTTP entries that belong to the auto-generated notifications and benign update requests. The entires are presented as a 4-tuple (host, destination IP, request type and program[5]). We label 28 distinct entries in the training data. These entries belong to 6 apps. The random forest classifier identifies 41 new entries from the testing data (52-day data) that can be added to the whitelist. In the later detection, the requests that match the whitelisted entires can be marked as benign to reduce the false positives.

FP and FN analysis. For all classifiers, the false positive rate ranges from 1.4% to 2.9%. False positives are mostly due to that the benign apps occasionally send out statistical data for advertisement. For example, a social app sends `POST` requests to `www.google-analytics.com/collect`.

The update requests sent to a popular third party host may be classified as false negative. For example, a weather app sometimes fetches metadata files (`json` or `xml`) from `s3.amazonaws.com`, which counts as FNs.

5.5.6 TR Analysis on Dataset II

We run the TR Analysis on the dataset II by varying the training sizes and plot the results in Figure 5.8. All TR accuracy results are above 96.0% for training data that are greater than 5 days. The differences of TR accuracy are less than 1.0% for three classifiers when using 15-day and 20-day training data. The accuracy results in Figure 5.8 demonstrate the same pattern as the plots in Figure 5.6. The results converge to a high accuracy, which shows it is feasible to use a small dataset in testing the real-world (mixed benign and malicious) Android network traffic.

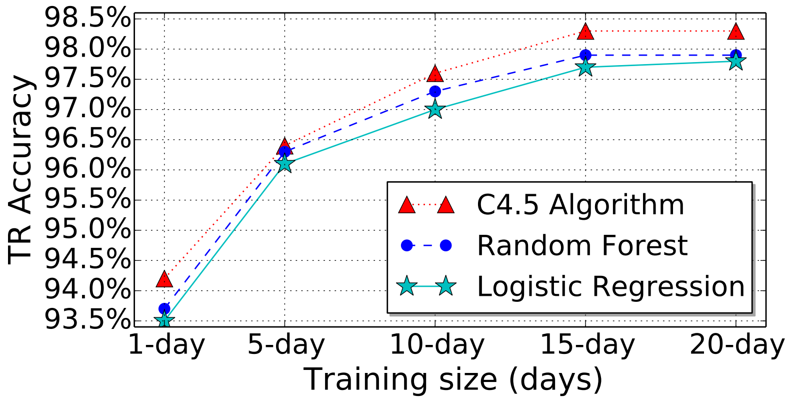


Figure 5.8: The TR accuracy of dataset II.

[5]The request type specifies the `GET` or `POST` requests. The program is identified using the PID and PPID (parent PID) information generated from the `netstat` and `ps` commands.

5.5.7 Detection of Malicious Requests

The purpose of this experiment is to evaluate the effectiveness of the constructed TRG in detecting malware activities. We first remove the notifications and update requests based on the whitelisting created in §5.5.5. Thereafter, we run the binary classifiers on the root-triggers of the constructed TRG (training size is 10-day). Shown in Table 5.5, both tree-based classifiers (`C45` and `R-F`) achieve the better prediction accuracy (F-scores) than the logistic regression (`LOG`).

Table 5.5: Detection accuracy results of three classifiers on Dataset II (sorted by F-scores).

Classifier	FPR	FNR	Precision	Recall	F-score
`C45`	5.2%	0.6%	0.991	0.994	0.993
`R-F`	0.7%	3.6%	0.999	0.964	0.981
`Log`	8.1%	27.8%	0.980	0.722	0.832

We label 12 malicious apps, including adware, Trojan and drive-by download apps. We successfully detect 12 newly installed malicious apps throughout the detection. A summary of the detected malicious apps is listed in Table 5.6. Our solution provides a high accuracy in detecting both existing malware families and new ones of all types. For the known malware variants, e.g. `wbfire.facts` and `com.crazyapps` families, our solution detects them as they send requests to fetch advertisements or conduct bot activities. During the study, our organization IDS (FireEye) only reported the malware activities from the `com.crazyapps.angry.birds` and classified it as `Trojan.Android.Plankton`. FireEye fails to detect other malware in our experiment.

The malicious apps we detected cover a wide range of adware, Trojan, bots, spyware and drive-by download attack. For example, a game app (`com.Punda.Free`) generates a shortcut link icon to `ad.leadboltapps.net/show_app_icon` on the Android desktop once it is installed. It issues `GET` and `POST` requests to `ad.leadboltapps.net`. A bot app (`com.gfgfgg.dsdf`), pretending to offer optimization tools for Android, actively sends out `POST` requests to `send.cxpts.com` every 5 minutes in the background. Results show that our solution successfully identifies all malware by flagging more than 99.1% their outbound requests. Our method does not have any assumptions on the type of apps or what code obfuscation techniques used in malware.

Comparison with existing solutions. Existing solutions on detecting malicious URLs and domains use temporal, lexical and host-based features from the root-trigger requests [32, 99]. We confirm that the same classifiers, solely using these features, fail to identify 3.3%-4.9% malicious requests and cannot detect four malicious apps in Dataset II.

FP and FN analysis. Based on the detection using C4.5 algorithm classifier, we find that the FPs (54 requests) are mostly due to the redirected traffic to some well-known domain servers. E.g.,

Table 5.6: The package name and communication server of malicious apps in our testing set of dataset II.

ID	Package Name	Communication Server
Adware		
1	`com.allen.txtjjsz`	`gw.youmi.net`
2	`com.chenyx.tiltmazs`	`ade.wooboo.com.cn`
3	`com.zxcalendar.chapp`	`api.is.apmob.cn`
4	`wbfire.facts` family	`media.admob.com`
5	`com.ctinfotech.snake`	`media.admob.com`
Trojan/Backdoor/Bots/Spyware		
6	`com.crazyapps` family	`searchwebmobile.com`
7†	`com.gfgfgg.dsdf`	`send.cxpts.com`
8‡	`com.gucdxjdl.batterysaver`	`send.cxpts.com`
9	`com.GoldDream.TingTing06i`	`lebar.gicp.net`
10	`com.qnuou.game`	`lebar.gicp.net`
11	`com.wing.qingshongxry`	`static2.ad.anzhi.com`
Drive-by download/Update attack		
7†	`com.gfgfgg.dsdf`	`subscription.teebik.com`
8‡	`com.gucdxjdl.batterysaver`	`au.umeng.com`
12	`com.Punda.Free`	`ad.leadboltapps.net`

†, ‡: Both apps exhibit malicious behaviors in two categories.

when a malicious app fetches a list of apps in its WebView for luring users to download, some requests are sent to `lh3.ggpht.com` for retrieving images. There requests are FPs, because the domain is benign. Other FPs may be due to the uncommon domain names and unconventionally long request strings. FNs (22 requests) are mainly sent by spyware or Trojan. The false negative rate is relatively low, e.g., among 100 `POST` requests sent to `api.is.apmob.cn`, only 2 are identified as benign.

5.5.8 Performance

We evaluate the performance regarding: *i)* the pairing efficiency for using our pairing algorithm and a baseline, and *ii)* the running time of testing on pairwise comparisons and mining on root-triggers. We obtain the performance results by averaging the running time over 5 rounds.

Pairing efficiency. The baseline pairing operation was proposed in [157], which uses a fixed time threshold as pairing window and an efficient pairing algorithm to pre-screen the data before pairing. It takes 185 and 121 seconds to use the baseline algorithm and ours respectively, for pairing all requests in our dataset. Our algorithm improves at least 30% pairing performance.

TR analysis and detection. In Table 5.7, we report the runtime of each operation in our prototype. The performance of the classifiers is consistent for both datasets in each operation. The training time is a dominant factor of the total running time for all classifiers.

Table 5.7: The performance of each operation is shown. The results are calculated in seconds per 1000 records.

Operation	Dataset	Runtime: training + test (seconds)		
Pairing	I	0.064		
	II	0.087		
		`C45`	`R-F`	`LOG`
TR analysis	I	0.13+1.9e-3	0.16+2.3e-3	1.65+6.8e-3
	II	0.18+2.3e-3	0.19+3.1e-3	1.66+7.2e-3
Detection	I	0.05+8.2e-3	0.07+9.7e-3	0.21+1.2e-2
	II	0.03+1.1e-3	0.03+2.0e-3	0.17+4.8e-3

Note: The runtime includes both training and testing in TR analysis and detection operations.

In both TR analysis and detection operation, C4.5 algorithm takes the least time. The random forest classification is slightly slower than C4.5 algorithm, while logistic regression takes the longest time. Overall, the running time of detection is less than that of TR analysis using the same classifiers. The total runtime of the detection operation is significantly less than that of the TR analysis, as the size of $\mathbb{R}_{\mathbb{T}}$ is much less than that of $|\mathbb{P}|$, i.e., $|\mathbb{R}_{\mathbb{T}}| \ll |\mathbb{P}|$.

Summary. We highlight our findings below.

- The decision tree-based classifiers (`R-F` and `C45`) yield the best TR accuracy and runtime. Our approach to discovering the triggering relation is scalable and does not require training on the entire data.
- The C4.5 algorithm and random forest classifiers give high precision and recall on identifying the benign notifications and malicious requests. The false positive rate is 0.7% using the random forest classifier.
- We successfully reveal 12 *new* malicious apps by flagging 99.1% of their stealthy network activities with remote hosts. We confirm the detection capability of our approach by pinpointing the sparse anomalies out of voluminous traffic data.

5.6 Summary: Triggering Relation Discovery on Android

In this chapter, we describe an Android malware detection technique that analyzes the dependency of network traffic on mobile devices. Our analysis explores the request-level traffic dependence and reasons about the root-triggers for all HTTP requests sent from the device.

Our approach provides the precise and structural information of dependency inference, comparing to the existing binary classification (e.g., email spam detection). We detect the stealth malware activities by using the features from the constructed TRG. Malicious requests are identified due to the lack of dependency with legitimate triggers. The detection is based on the policies that malicious requests are the lack of dependency with legitimate ones and cannot be categorized to the whitelist.

Our prototype is evaluated on 14GB network traffic data and system logs collected from an Android tablet. Experimental results show that our solution achieves a high accuracy (99.1%) in detecting malicious requests sent from new (zero-day) Android malware.

In this work, we make a step to fill the gap between dynamic analysis and network anomaly detection on Android. We show our detection method is scalable and achieves a high detection rate with dependency-based features. Our study makes an important step in addressing the significance of using dependency knowledge in the detection of the malicious requests.

Chapter 6

Traffic Dependence Visualization for Security

In this chapter, we design a tool, ReView, to assist the analysis of host-based network data based on the traffic dependence.[1] Our visual representation improves the sensemaking process for security and can increase the productivity for security analysts.

The chapter is organized as follows. We describe the motivations and specific problems on the security visualization work in §6.1. The visualization design and implementation details are presented in §6.2 and §6.3, respectively. A case study is given to demonstrate how a security analyst uses our tool in §6.4. We conduct a user study to evaluate our tool in §6.5. §6.6 concludes the chapter.

6.1 Motivation and Problem Statement

This chapter addresses the issue of visualizing the network traffic causality. We aim to design a visualization tool to facilitate the process of identifying anomalous network traffic.

The recently proposed detection method advances the analysis of network traffic by inferring the semantic and logical relations [157]. Its unique advantage is the capability of reasoning about the dependency of network data and thus detecting new stealthy malware activities. The analysis provides automatic anomaly detection in the observed network activities through the probabilistic reasoning of the causal relations in traffic. By pinpointing abnormal network events that lack of valid triggers, it can detect malware activities on an infected machine (e.g., making command-

[1]The content of this chapter is mainly based on our paper [155].

and-control communications with its controller). The triggers include legitimate user events and benign network packets. In this work, we design a visualization tool to facilitate the understanding of host-based network traffic and enhance the sensemaking process for security analysts.

Sensemaking is an analysis process including the tasks of investigating the complex data, exploring their connections, and gaining insights [54]. The dependencies and semantic information of requests are useful for human experts' cognition, reasoning, decision making in cyber security [53]. Our visual representation improves the sensemaking process for security and can increase the productivity for analysts.

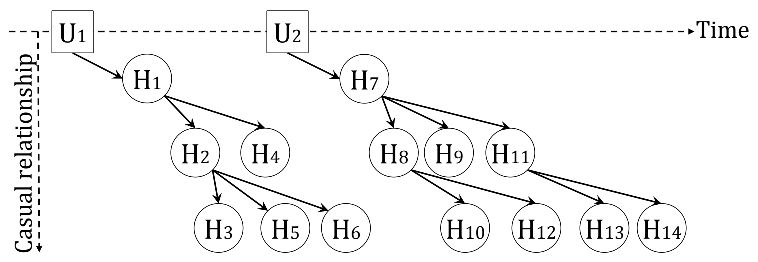
Many existing network security visualization tools provide graphic user interfaces for Intrusion Detection Systems logs (e.g., Snort) [35, 42, 89, 126]. IDS alerts are organized in a log-type structure, where each alert entry indicates a potential intrusion threat. However, very few existing work provides the visualization of the underlying relationship among network events, with one notable exception Portall [58]. Portall visualizes the correlation of host processes and network activities. Our request-level traffic causal relations are much more fine-grained than the process-level correlation in [58]. Thus, new visual representation approaches are needed for displaying the structural information.

Several visual representation techniques [76, 88, 122] have been proposed to visualize the dependency structures. However, these general techniques on visualizing hierarchy or dependence cannot be directly applied to our traffic dependence data. The reason is our unique space efficiency requirement, that is, *how to optimally utilize the screen space for displaying the causal relations of a massive amount of network traffic*.

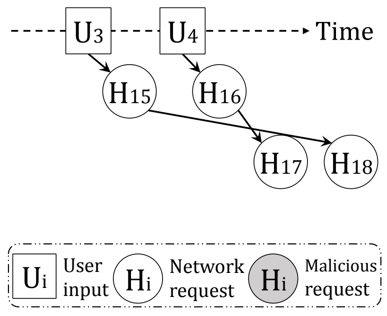
6.1.1 Visual Locality

The triggering relation graph (§3) is built on the application layer packets and used to find all triggering relations to understand how a user interacts with applications and how applications respond to the user by sending out network requests. Therefore, it reveals the logical structure of the requests, which can be used to detect abnormal network activities originated from the host.

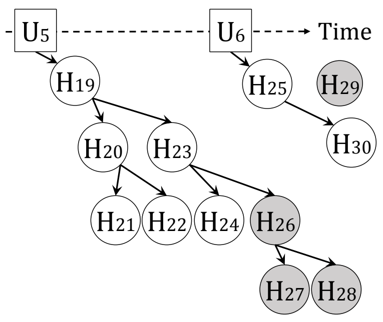
There is no existing tool to display the triggering relation graph. A straightforward approach for displaying host-based traffic dependence is shown in Figure 6.1a. This visual representation arranges network requests using a forest-like layout based on their causal relations; the timeline may be extended horizontally when newer network events are added. Because this layout is intuitive, it has been used for illustrating relations among network events [92, 157]. However, this forest-like layout (Figure 6.1a) does not use the display space efficiently. The length of traffic causality structure grows fast, making it difficult to view related events that are temporally far apart.



(a) A straightforward representation of TRG.



(b) Two crossing edges.



(c) Malicious requests.

Figure 6.1: Schematic drawing of traffic causality for outbound network requests on a computer. Nodes are indexed by their relative occurrence time.

Although rare, we observe that network requests that occur 15 minutes apart may have causal relations. In addition, the requests with causality do not necessarily situate close according to the time, as illustrated in Figure 6.1b (H_{15}-H_{18}). The crossed edges make the TRG messy and hard to analyze.

Definition 6.1.1 (High visual locality). *Items having logical relations are placed close to each other on display.*

In our context, we define *high visual locality* as our primary goal to optimize our visualization designs. This concept requires the arrangement of requests with causality that enables analysts to identify malicious requests with ease. It enables analysts to easily identify related requests. To meet this requirement, our design prioritizes the causality that clusters nodes around their root-triggers and forms trees separately. Within each tree, the nodes are organized by their temporal and other logical information.

In our security model, we consider two types of stealthy malicious network traffic.

- Network requests without valid root triggers are referred to as ***vagabond*** (H_{29} in Figure 6.1c).
- Network requests sent to malicious hosts with valid referrer information are referred to as ***grafted*** (H_{26}-H_{28} in Figure 6.1c).

The vagabond requests, without legitimate root causes, are likely raised due to stealthy malware activities. The grafted requests take place when the servers are misconfigured or compromised, and thus are hard to formalize rules to identify. Blocking the malware network activities effectively isolates the malware, such as spyware exfiltrating sensitive information through outbound traffic. Our TRG model is general and needs no *priori* knowledge about a particular malware class.

6.1.2 Analysis Using All Triggering Relations

One of the design choices for visualizing TRG is whether or not to display *all* the discovered triggering relations. A simple visual representation is only to display vagabond requests, i.e., abnormal network events. However, analysts may neglect some attacks by viewing this type of display, because suspicious requests could be hidden from legitimate ones (i.e., the *grafted* requests). For example, a common attack on web servers is to exploit web vulnerabilities (e.g., SQL injection, cross site scripting, format string injection [59]). After the servers are compromised, attackers can modify the websites by injecting malicious codes, then the clients get infected when they visit the websites. In a recent example, Yahoo's European servers were attacked by malware in January 2014. The infected servers sent out the "exploit kit" to the clients who visited Yahoo's webpages during the infection [10].

These grafted requests are often of the `JavaScript` type with long and obfuscated request strings in URLs, because `JavaScript` requests have handy and advanced functions to be leveraged by attackers. Domain experts have to reason about the legitimacy by integrating with other information (e.g., system logs). It costs security administrators a significant amount of time to figure out the components of a website that got malicious code inserted, let alone finding the origin of infected points [30].

Our TRG model provides a good visual representation for a host-based overview. The *grafted* requests, sent by compromised servers, can be identified by their deeper levels in TRG, late-arriving timestamps, and unusual host domains. Therefore, the analysts need to leverage the inner logic to infer the legitimacy of requests, which makes the displaying of *all* triggering relations more desirable.[2]

6.1.3 Contributions

In this chapter, we introduce ReView, a visual analytic tool, which enhances the sensemaking process for network traffic analysis with human-perceptible visual representations. Our visualization

[2]It is worthy pointing out that we focus on the network request rather than the packet, as the packet might not carry enough semantic information for analyzing its dependency and one request might cross multiple packets.

tool takes as input the network requests and their dependency information. It provides an interactive graphic interface for users. We leverage a unique design that allows the optimal usage of the screen for displaying network events. Our tool can be used by individuals or organizations for protecting their computers. For example, it can be used by security analysts to monitor network traffic, examine traffic anomalies, perform forensic analysis on the network incidents.

Our contribution is twofold.

- We develop a visualization tool for security analysts to efficiently display the network traffic dependency. This tool has a *visual locality* feature that can optimize the displaying of structured data. The visual representation is a *radial* layout based on a curved timeline. Compared with the straightforward axial layout, our design maximizes the use of the screen by bending the timeline into a circle, which achieves high visual locality and extensibility.
- We conduct a user study to evaluate our system with 10 participants and real-world network traffic. Results of the study show that our tool is well suited for security analysts to perform manual inspection and analysis on network events based on their causal relations.

6.2 Visualization Design

Our design for the visualization tool is based on characteristics of traffic triggering relations. We run a pilot study that contains 12MB network data (10-hour HTTP traffic on a host, 45000+ requests) to investigate the characteristics of traffic dependency. We summarize our findings as follows.

Wide-and-shallow trees. The nodes on the top three levels in TRG account for above 90% of the total amount. There are respectively 68% and 21% of requests on the second and third levels, so the trees are extremely wide at their top levels. Besides, 99.7% of trees in TRG have less than 6 levels, while there is only 0.3% of trees whose depth is 7 to 9 levels, which illustrates the shallowness of the trees.

Temporally adjacent events. We further check the time difference between any two requests that have a triggering relationship. Statistics shows that about 93% of HTTP requests trigger their dependencies within 3 minutes. If the time window is enlarged to 15 minutes, then 99.8% of HTTP requests and their dependencies are included. Therefore, the HTTP requests with dependents are temporally close to each other, despite some rare cases.

Sparsity of vagabonds. The vagabond requests are classified into two groups. Malicious and misconfigured packets represent 0.3% and 0.5% respectively of the total number of nodes. *Malicious* requests are sent when a user visits a compromised website or the host is infected by malware. *Misconfigured* requests are not sent to malicious hosts but contain some missing fields in the request header.

Automatic update requests. The updates are the legitimate requests sent to upgrade the system or software periodically, without user interaction. In our design, we maintain a list of known programs and their official domains to reduce the false alarms (e.g., `Java Update Checker` and its domains `javadl-esd.sun.com` and `javadl.oracle.com`).

6.2.1 Design Goals

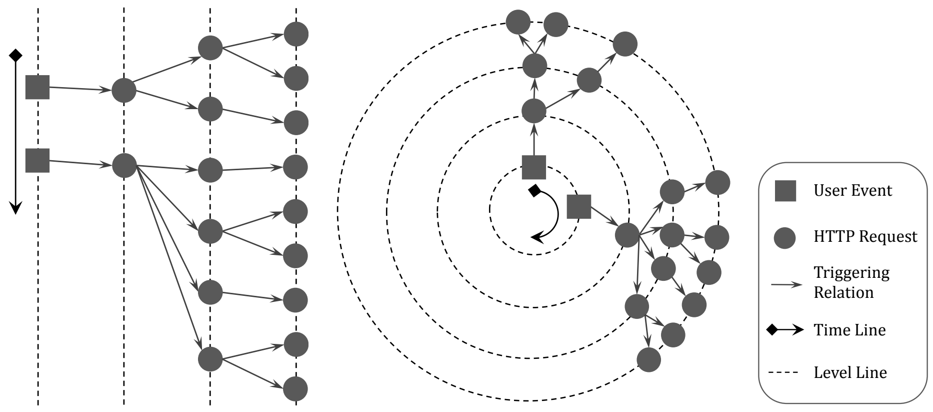
Based on the findings from the pilot study, we highlight our design goals as follows.

- **High visual locality.** The design should meet the *visual locality* requirement, so that the nodes having logical relation should be placed close to each other. In addition, the nodes on each level need to be explicitly displayed and chronologically ordered.
- **Scalability.** The network requests are intensively clustered along the time, so are their dependencies. A condensed view is needed to reduce the redundancy of the benign requests. An adjustable time window should be added to limit the size of rendering voluminous nodes at one time.
- **In-depth & on-demand knowledge display.** The overview representation should aim at presenting the causal relations, meanwhile minimizing the display of the detailed information for each request. Additional options can reveal the details of different levels, as requested by users.

Our design aims to visualize the traffic dependency of network requests by meeting the *high visual locality* principle, so users can identify and analyze the anomalies with ease. We utilize a *radial* design for displaying traffic triggering relations. This design has a curved timeline that is centered on the display, and the radiating branches represent network events and their trigger relations. The advantage of this view design is that it maximizes the utilization of the display screen. It is more convenient for users to interact with this view than a conventional straight timeline view. We further provide a condensed view to simplify the display by merging the trivial nodes. In addition, our design uses a *heatmap* to show the distribution of the requests over time and store original logs in each color-coded tile.

6.2.2 Visual Locality Design for Analysts

We design a radial layout to display the traffic dependency with *high visual locality*. A straightforward visualization of TRG would be an axial layout in Figure 6.2a. However, it is not suitable for visualizing large-scale traffic dependencies. This layout is not efficient at displaying hierarchy structures, as the tree structure spans unilaterally and leads to much unused space. In addition, the length of the forests grows as the data size increases. Users have to scroll up and down for browsing and searching. In the radial layout shown in Figure 6.2b, we arrange the nodes in a clockwise manner. In this design, *timeline* defines the positive direction and the start point is at 12 o'clock position, which is consistent with an analog clock and intuitive to users.



(a) Axial layout. (b) Radial layout.

Figure 6.2: Schematic diagrams of two visualization designs for the structured network data.

Our prototype arranges the nodes and allocates the space of the radial layout as follows.

(1) We sort the root-triggers by their timestamps and plot them in the innermost ring. We cluster the nodes under their root-triggers. Clustering nodes in each tree guarantees no cross edges in the display.

(2) The rendering space is allocated in proportion to the number of nodes on its *second level*, rather than the timespan of the tree in TRG. The time spans of each tree in TRG may overlap. Requests from different trees may arrive simultaneously. Our layout, based on indexing nodes, eliminates the overlapping issue.

(3) We render the nodes by their levels. The *level lines* are used to align the nodes on the same level. The nodes on the same level are lined up on the concentric arcs. The innermost ring is used to place the user inputs, and is divided into sectors whose angles correspond to the sizes of its dependent nodes.

In our design, the trees in TRG are plotted in the sequence of root-triggers' temporal relation, so that none of the tree is overlapped or has crossing edges. Our solution conforms to the *visual locality* principle so that nodes with causalities are placed together without bringing in mess.

Our design maximizes the usage of display space. The radial layout presents hierarchies of events in concentric rings, so it fully utilizes the screen by bending the level lines into circles. Users using the conventional axial layout need to scroll *twice* more than our design, to see the entire graph. Additionally, by concentrating nodes at the center of the screen, the radial layout enables users to easily manipulate the display. The usability is evaluated in §6.5.

6.2.3 Interactive Heatmap

Heatmap is a graphical data presentation approach where each value in the matrix is color-coded. Our design is composed of `LogMap`, an instance of Heatmap (see Figure 6.4). The `LogMap` reveals the density information of the network requests. It provides an overview of the request distribution over the observed period. `LogMap` navigates users to start an analysis by focusing on particular grids (e.g., grids with dark colors). Additionally, security analysts often resort to the original logs for more details. Therefore in our design, the `LogMap` supports users to access the original logs by clicking on the colored tiles in the heatmap.

The `LogMap` divides the timeline into fixed windows (e.g., one minute) and organizes network events of each window into a sub-block. In our design, there are sixty tiles in a sub-block, which represents sixty seconds. The color coding in each tile corresponds to the number of requests. The `LogMap` accommodates different levels of granularity in time, e.g., seconds, minutes, hours, etc.

6.2.4 Condensed View to Distill Information

According to our pilot study, more than 90% of network requests are situated on the top three levels in the TRG. To avoid visual clutter, we provide a condensed view for security analysts. We design a condensing algorithm to merge the nodes that meet *all* the following criteria.

- Legitimate requests that are of the same type;
- Requests that are on the same level in the TRG;
- Requests that are the leaf nodes in the TRG.

The condensing algorithm iterates a list of chronologically sorted requests and outputs a list of condensed nodes. We use an auxiliary dictionary to store each newly generated node and a list of requests being condensed. In the algorithm, we only merge the benign requests, so as to avoid losing any information of abnormal requests. Our condensing algorithm does not merge the nodes on the different (sub)trees, which guarantees that the dependency structure in a TRG is preserved. Therefore, the condensed views are compatible to the original radial and axial layouts.

We categorize the HTTP requests into six types, which are web, CSS, JavaScript, multimedia, data, and others. Around 50% of browser-generated HTTP requests are used to fetch the multimedia objects (e.g., image or streaming data) in our pilot study. Unlike JavaScript objects, these requests to obtain *static* files do not trigger further HTTP traffic.

To assess the effectiveness of the condensing algorithm for reducing the redundancy and emphasizing the anomalies, we test our tool on the pilot study dataset. Shown in Table 6.1, we compare the number of nodes in both original and condensed views. *Compression ratio* is defined to evaluate the effectiveness of the algorithm. The root-triggers are on the first level, and thus cannot be compressed. There are 68% of total requests situated on the second level, and 87.5% of them are merged, which significantly saves space. Overall, the total compression ratio is 82.2%. Multimedia requests mostly serve as leaf nodes and can be compressed as much as 91%. Compared with the original view, our condensed view significantly reduces the redundancy of displaying leaf nodes. Therefore, it helps users identify abnormal nodes due to its visually salient.

Table 6.1: The number of HTTP requests on each level in the Triggering Relation Graph for original and condensed views.

Level in TRG	# of nodes in original view (n)	# of nodes in condensed view (c)	Compression ratio $(1 - \frac{c}{n})$
1	1158	1158	0.0%
2	31242	3913	87.5%
3	9725	2190	77.5%
4	2753	644	76.6%
5	835	225	73.1%
6	201	49	75.6%
7 - 9	74	24	67.6%
Total	45988	8203	82.2%

6.3 Prototype Implementation

We build ReView, a visualization tool for viewing and analyzing the triggering relations for network requests. It is designed based on the three-tier architecture and implemented as a web-based tool. The workflow is illustrated in Figure 6.3. The back-end consumes the raw data and focuses on reasoning. The front-end displays the hierarchy structure for users to analyze.

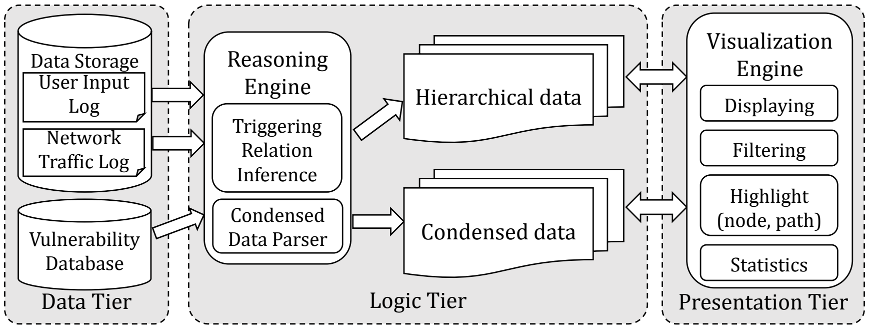


Figure 6.3: The workflow of ReView.

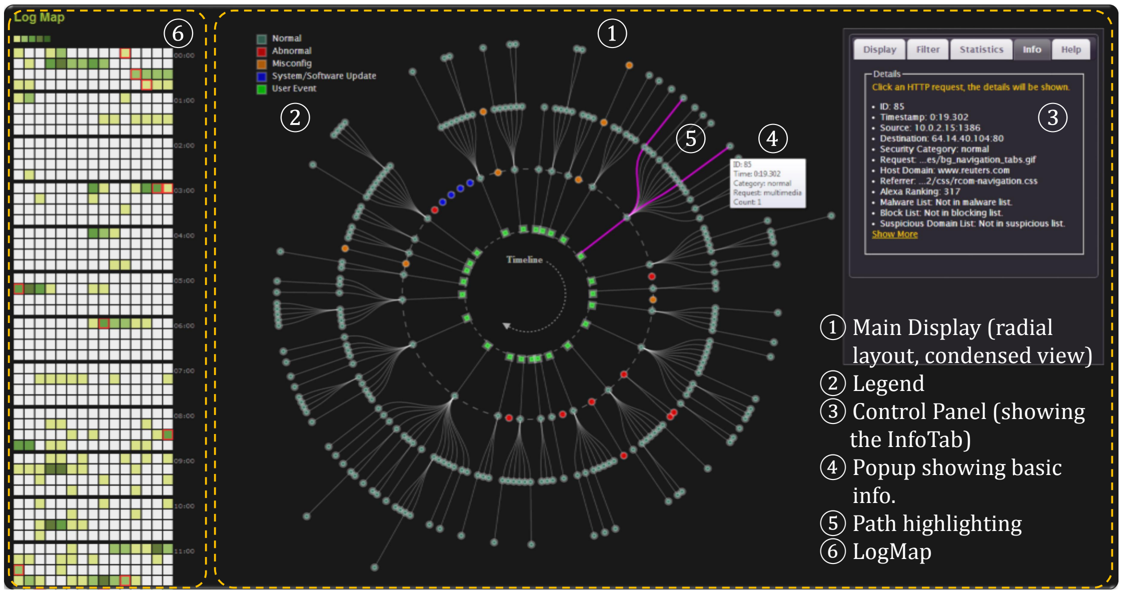
Data tier. ReView takes the user events (e.g., clicking on a hyperlink of a webpage) and the outbound HTTP requests as inputs. The network requests are recorded by leveraging the `libpcap` library. The features of an HTTP request include its timestamp, process ID, source and destination IP address, and request semantic information (e.g., HTTP host domain and referrer). The inputs of the prototype, including the raw data of user inputs and network requests, are saved in a database. In the `data tier`, we add a customized vulnerability database, which is composed of several known blacklists and feeds [11, 15, 18], to facilitate the process of identifying the suspicious requests. However, solely relying on the blacklists is not sufficient because of the ever-changing malware variants and malicious domains.

Logic tier. In the `reasoning engine`, our tool infers the triggering relations by leveraging a machine learning-based solution proposed in [157]. It discoveries vagabond request that are not generated by any user actions. The benign requests that are periodically sent to known websites for upgrading can be easily summarized and filtered out as system/software updates. To further categorize the vagabond requests, we employ both whitelist and blacklist. In ReView, a whitelist is created according to the Alexa top 1000 sites [1], so that the vagabonds that are to these websites can be recognized as misconfigured requests, rather than malicious. Finally, we build a customized blacklist by integrating the source data from several vulnerability databases and feeds [11, 15, 18], based on which we flag the vagabond requests as malicious.

Presentation tier. The rendering in ReView uses `D3` [5], which is a JavaScript library for data visualization. The layout of ReView is shown in Figure 6.4. Our design contains two major components, *main display* and *heatmap* panels.

On the main display panel, we show the optimized visualizations of hierarchical structure using the radial layout. The user inputs are placed in the innermost ring. The HTTP requests are situated on the second and later levels. On the right hand side, a draggable control panel is composed of five tabs that are used to manipulate the layout options, query the source data, highlight the nodes, and show the statistics. Our tool supports path highlighting for exploratory analysis. The path from a selected node to its root-trigger can be highlighted, which helps security analysts identify the *logic chain* of the nodes, understand why the request is triggered and find the common ancestors for multiple nodes. Our tool aims at presenting the causal relations, meanwhile minimizing the display of the detailed information for each request. In ReView, there are three ways to display the HTTP request information: *i)* using the `Popup` when a mouse hovers over a node; *ii)* reading the information at `InfoTab` after clicking a node; and *iii)* loading the complete information in a separate `LogWindow`. These options can reveal the details of different levels, as requested by users.

On the heatmap panel, the `LogMap` reveals the traffic patterns, which is complementary to the main panel. Each tile in `LogMap` represents one second and is colored on a green scale based on the number of requests in this second. The tile with a red frame indicates that at least one vagabond HTTP request is observed during its time window. A separate window that displays the original logs is shown when a colored tile is clicked.



ID: 85
Time: 0:19.302
Category: normal
Request: multimedia
Count: 1

Details
Click an HTTP request, the details will be shown.

- ID: 85
- Timestamp: 0:19.302
- Source: 10.0.2.15:1386
- Destination: 64.14.40.104:80
- Security Category: normal
- Request: ...es/bg_navigation_tabs.gif
- Host Domain: www.reuters.com
- Referrer: ...2/css/rcom-navigation.css
- Alexa Ranking: 317
- Malware List: Not in malware list.
- Block List: Not in blocking list.
- Suspicious Domain List: Not in suspicious list.

Show More

Figure 6.4: A screen layout of ReView.

Data integrity. In this work, we consider application-level malicious requests, so the kernel-level system data (e.g., keyboard and mouse events) are assumed to be trustworthy. To prevent the forgery of user inputs, advanced keystroke integrity solutions such as [71, 128] can be incorporated in our work to further improve system-data assurance.

6.4 Case Study

We describe how a security analyst, Alex, utilizes ReView to detect anomalies in a 4-hour-long traffic with 22,000+ HTTP outbound requests.

(1) **Choose visualization options:** Alex selects the condensed radial layout for viewing the hierarchy structure of network requests. ReView reduces the total number of displaying nodes from 22,000+ to 3,771, which significantly mitigates the performance burden and reduces the unnecessary displaying mess.

(2) **Filter and highlight:** Alex is interested in querying and highlighting the abnormal nodes, so he clicks the `FilterTab` and selects the filtering options (e.g., by legitimacy, request type or host domain). The vagabond nodes are highlighted with colors. Our tool filters out 42 update requests. Also, ReView highlights 225 vagaboud requests and their dependents, which send to 60 distinct domains.

To narrow down to a specific period, Alex displays the traffic data up to 30 minutes for each time by using a sliding window. In the first window of 30 minutes, he finds that there are 17 vagabond requests highlighted by ReView.

(3) **Investigate *vagabond* requests:** Alex clicks on one vagabond node that is highlighted in red. By reading the detailed information from the `InfoTab`, Alice finds that this request is sent to `gighippo.com` and its referrer is null. He then notices that its domain name is on `ZeuS` blocklist [18] and its Alexa global rank is beyond 5 million.

(4) **Investigate suspicious *grafted* requests:** Alex spots a request situated on the 7th level of the TRG. By conducting a further investigation, he finds out that the request is sent to a suspicious host (`ziffdavisglobal.112.2o7.net`) with an irregularly long request URL string (length=1309). After checking the vulnerability database and online resources (e.g. `Web of Trust`), Alex confirms that the request is not sent to a trustworthy host, so it should be blocked.

(5) **Make security decision:** Alex mainly examines four attributes: IP address, host domain, existing blacklists, and the triggering relation (computed by the `reasoning engine`). By repeating the first four steps, he makes a quick-yet-accurate security decision on the legitimacy of the 4-hour-long network traffic.

Alex finds out 94 (out of 225) requests belong to adware, spyware and tracking-site traffic. The rest (131 out of 225) requests are sent to known servers, but they contain missing values in their headers that may be due to misconfiguration of the servers. Last, he identifies two scenarios of grafted requests (12 requests in total).

Our tool provides an interactive visual display for the network requests on a host. In Figure 6.5 (zone A), we demonstrate a scenario of grafted requests using an exemplary case. These requests are usually situated in exceptional deep levels and expanded to deeper levels. In addition, these requests are of `JavaScript` type with long request strings in their URLs. The `JavaScript` requests provide handy and advanced functions to be used by attackers, which is difficult to be detected by rules. With the help of ReView, the trees in TRG that have an exceptional depth and late-arriving `JavaScript` requests are caught more attentions to security analysts.

Vagabond requests, detected by the `reasoning engine`, are shown in Figure 6.5 (zones B and C). We find an interesting pattern that two similar abnormal requests are sent out after a certain website is visited. The website hosts may be compromised by malware or due to misconfiguration.

Besides the triggering relationship of requests, analysts can reason about the legitimacy by reviewing the URL and integrating with other information (e.g., system logs) using our tool. A potential limitation of condensed view is when the redundant leaf nodes are merged, so that malicious grafted nodes could be hidden.

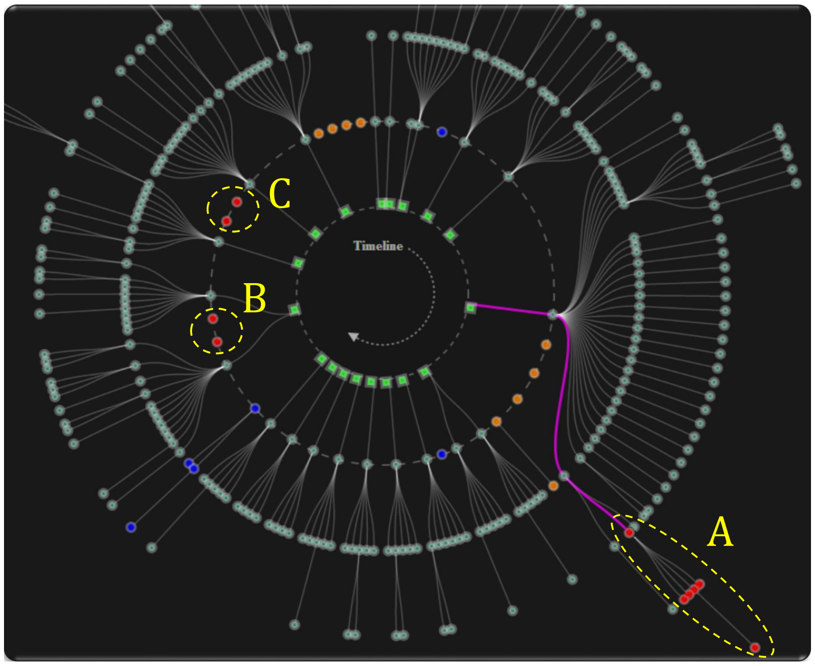


Figure 6.5: A Screenshot to show two types of abnormal requests in ReView. Zone A refers to the grafted requests. Zones B and C refer to the vagabond requests.

In the above case study, we elaborate the process of using ReView to analyze the network logs and make security decisions. It is not only convenient but also accurate by using our tool, comparing to the conventional way that human experts have to write queries and compare logs side by side.

More discussion. We regard ReView as a forensics tool. It has three major uses: *i)* to help analysts understand the network dependency, *ii)* to find new traffic patterns, and *iii)* to recognize/prevent the attacks in the future. In the workflow, security analysts, at the end of the "analysis loop", can treat our tool's output as actionable intelligence. To further accommodate the analysts, one possible future work direction would be to extend our tool with visual analytics capabilities. Our long-term vision is to enable the analysts to steer the reasoning engine, thus bringing the human input back into the loop.

6.5 User Study

We carried out a user study with ten participants, and all of them have at least four years of experience working with computers. Their specialties include system/network security, high performance computing and human computer interaction. As our tool is to visualize the network requests for security purposes, it is not built for the average users. We target users with computer science knowledge and people who are curious about, or care about, the computer security. In this user study, we investigate: *i)* the user's preference, and the trade-off between a neat view of condensed data and a complete view of all triggering relations, and *ii)* how our tool can direct users to analyze the suspicious requests.

We conducted the study in a computer science laboratory at a university. A 15-minute tutorial was given to the participants to introduce the functionalities of ReView. We directed the tutorial using a 7-page slide presentation, so that every participant got the equivalent level of details. In the study, participants were asked to finish 10 questions that include the tasks of analyzing logs and the user preference of different visualization options. Questions 1-5 refer to the tasks of analyzing logs that are usually done by security analysts. Questions 6-10 ask about the user preference of different visualization options. The tasks are generated and selected by the researchers and domain experts, based on whether or not the task would naturally occur in the analysis of the network traffic logs. We list the 10 questions as follows.

Q1 Please list the HTTP header information of 3 root-triggers requests.
Q2 How many levels of requests in this graph? Judging from the graph, what other insights do you find?
Q3 Please specify the number of each type of vagabond requests in the graph.
Q4 Choose one example of each vagabond type and list all relevant information obtained from the tool.
Q5 Please find the first malicious request, could you specify the host domain and infer the reason why it is sent?
Q6 Which way do you use more often to analyze the logs, the log view or the graphic view, and

why?

Q7 Which one you like to use, the axial layout or the radial layout, and why?

Q8 Is the condensed view a good way to trim the original view, and why?

Q9 Which way you like to browse request information, using the popup window, `InfoTab` or `LogWindow`, and why?

Q10 Do you think the `LogMap` is useful in your analysis? What did you find through the `LogMap`?

The data used in the user study was obtained from a graduate student's laptop. We collected both HTTP traces and user's inputs (keyboard and mouse events) for a 30-minute session.[3] The test data contains a total of 3724 HTTP requests, including 24 update requests and 33 vagabonds (12 abnormal and 21 misconfigured requests).

6.5.1 Analysis of User Preference

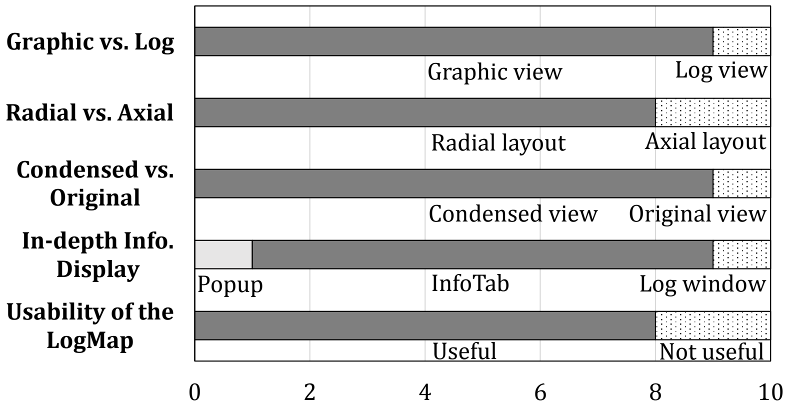


Figure 6.6: The breakdown of results on user preference in the user study.

Figure 6.6 presents a brief summary about user's preferences of major features in ReView. All ten participants correctly identified root-triggers, the first level of network requests, and answered the exact maximum level of requests. Based on results shown in Figure 6.6, nine participants indicated that they preferred visualizations over traditional network logs (for Q6). They agreed on the fact that the provided visualizations in ReView summarize the network data, compared with the overwhelming information in raw logs. Using ReView, users can quickly understand and make use of the structure of network causal relationship for security analysis.

An Easily Perceptible Representation. The proposed visual representations in ReView can be easily perceived and interpreted by users. Participants can relatively quickly understand and make

[3]The average time that global Internet users spend online is about 30 minutes, according to report [8].

use of the structure of network causal relationship for some basic analysis with a 15-minute demonstration, because all ten participants correctly identified root-triggers, the first level of network requests (for Q1), and answered the exact maximum level of requests (for Q2). Additionally, visualizations in ReView provide better representations to efficiently summarize key features of the network traffic data, compared with traditional network logs. Last, eight out of ten participants liked `LogMap` (for Q10). Six of them explained the reason for their preferences as that `LogMap` conveyed the density of packets over time, and the other two thought that `LogMap` presented the request patterns in an easily perceptible manner.

An Efficient Navigator from Three Levels. ReView effectively navigates users among a large amount of network requests through different levels of abstractions: network traffics causality (radial and axial views), aggregated network requests (condensed view), and network traffics density (`LogMap`).

Both radial and axial layouts present the structure of network requests causality, but they implicitly emphasize two different aspects: space usage and time order. Eight participants preferred the radial layout and five of them mentioned that they did not need to scroll the mouse to explore network causal relations (for Q7). Two participants who preferred the axial layout with the reason that the timeline in this view was more salient and easy to understand, which suggests that they explored traffic causal relations by following the time order. In addition, the condensed views more explicitly convey the structure of network traffic causality, compared with the original ones (for Q8). Supported evidence comes from the feedback of nine participants who preferred condensed views. They agreed that condensed views simplified representations by avoiding unnecessary information (e.g., benign requests). One participant who preferred the original views also admitted that condensed views were neat, but he still chose original views in case of missing necessary information. Lastly, `LogMap` enables users to examine network requests by their density with supportive evidence from the feedback of eight participants. With three-level navigations, ReView potentially allows users to perform analysis by their preferences.

6.5.2 Analysis of Suspicious Requests

ReView efficiently directs users to suspicious network requests (for Q3). Nine participants identified the exact number of vagabond requests (33 nodes in total). The other one participant misunderstood the task and counted the vagabonds on the first level. All participants correctly listed detailed information of examples corresponding to each type of the vagabond. They were able to distinguish among different types of vagabond requests by digesting the request information using our tool.

With some domain knowledge, participants can infer reasons for the occurrences of malicious requests (for Q4 and Q5). For example, eight participants answered that the first malicious request was sent to `spi.domainsponsor.com` and the domain was blacklisted as spyware. By reviewing its long URL string, participants found out that the request URL contains some substrings with semantic meanings, such as `migTrackDataExt` and `migAgencyId`. Therefore, they speculated the outbound request is used to leak host information. Users also found that the display of triggering relations for all requests benefits the analysis of malicious ones. For example, participant #9 pointed out that a similar vagabond request is sent out after a website is visited twice, so the website hosts may be compromised by malware or due to misconfiguration.

`LogMap` also helps to direct users to potentially suspicious network traffics with color coding of the frame for each grid. Participants #4, #7 and #10 regarded this as an important factor as to why they preferred `LogMap`. They used the tool to find out that the update requests were often sent out during the idle time, while abnormal ones were sent out along with some legitimate requests.

Summary. ReView benefits network traffic analysis from three aspects. With high visual locality, the proposed visual representations in ReView are used without much cognitive effort for training. Three levels of abstractions about network traffic provide more options for users to perform their analysis. Following network traffic causality and semantic meanings of the color coding, participants used our tool to successfully identify malicious network activities.

6.6 Summary: Visualization for Network Traffic Dependence Analysis

Discovering traffic dependency has been shown to be an effective way to analyze network activities and identify malicious events. We introduced a new concept of *high visual locality* and developed ReView, a visualization tool that maximizes the usage of a screen and helps security analysts better utilize network traffic dependency. In our design, we adopted a radial layout that supports to high visual locality and further optimized the display using a condensing algorithm.

ReView serves as an integrated solution for security analysts to accurately pinpoint anomalous network events and perform further investigation. Our user study and case study confirm that our tool provides easily perceptible representations to help interpret network traffic causality and enhance security analysis with multiple levels of network information.

Chapter 7

Conclusion and Future Work

In this chapter, we conclude this dissertation by summarizing our contributions and discussing directions for future work.

7.1 Conclusion

Correlating events to find dependency is a commonly used approach in reasoning events [21, 22] and detecting attacks [36, 60]. This dissertation studies the triggering relation discovery and its security applications in network security and Android malware detection. The work is motivated by the need of fine-grained analysis and enforcement of dependencies on network events. We introduce the generalization of triggering relation model and give several security applications in detecting the host-based malicious applications, software flaws, and Android malware.

In our dependence analysis, the triggering relation graph is an acyclic, expandable and sparse graph. The inferred dependency in a triggering relation graph illustrates the logic chains of the network requests and can be used to reveal the origin of the malicious activities.

We describe a rule-based approach for building a TRG. The triggering relation discovery algorithm incrementally inserts a new network event with unknown dependency to a well-formed TRG. The algorithm is generated based on empirically derived rules. Additionally, in our prototype, we instrument the browser with the header-authentication capability to ensure the integrity of collected HTTP header information.

A learning-based approach is proposed for processing general types of network traffic. The unique contribution of the machine learning method is to convert the general dependence discovery into the *pairwise* triggering relation problem. Thereafter, we design a novel *pairing* operation that produces pairwise features, so that the discovery problem can be efficiently solved using classification tools. We designed, developed, and compared both rule- and learning-based approaches for triggering relation discovery. We conduct several evaluations on our host-based solution using 10+ GB real-world and DARPA datasets. Results indicate a high accuracy of the triggering relation prediction using the learning-based classification. Additionally, we confirm that our traffic-reasoning technique is effective in detecting browser spyware, DNS bots, and data exfiltrating malware.

We extend our solution on mobile devices for analyzing the dependency of mobile network traffic. Our analysis explores the request-level mobile traffic and provides the precise and structural information of dependency inference. We adopt the two-stage learning-based solution to detect the malicious root-triggers. The dependency-based features, extracted from the constructed TRG, effectively distinguish the malicious root-triggers from the benign ones. The detection is based on the policies that malicious requests are the lack of dependency with legitimate ones and cannot be categorized to the whitelist. We conduct our evaluation on 14GB network traffic data and system logs collected from an Android tablet. Our evaluation results show that our solution achieves a high accuracy (99.1%) in detecting requests sent from the malicious apps.

We design and develop a visualization tool for visually analyzing malware activities. Our tool provides high visual locality for events that are logically related. In our design of the prototype, we adopt a radial layout and further optimized the display using condensing rules. We conduct a user study and results show that participants used our tool can easily identify malicious network activities with the semantic meanings of the color coding.

In this dissertation, we make a step to fill the gap between dynamic analysis and network anomaly detection on Android. Malicious requests are identified due to the lack of dependency with legitimate triggers. Our work focusing on the dynamic interactions between the user and application lends a novel security methodology that is not limited to the ever-changing anomalous patterns. We successfully demonstrated the use of triggering relation discovery for identifying suspicious requests both on hosts and mobile devices.

7.2 Research Directions and Future Work

We envision to extend our work presented in this dissertation in two directions.

Generalization. A generalization of our solution can be done by deriving policies and analyzing complicated network applications (e.g., online games that heavily involve user interactions) for anomaly detection purposes. Our long-term vision is to systematically investigate the design and use of complex policies involving the TRG for network assurance. As part of future work, we also plan to add intelligent filters (e.g., using episode mining or graph mining) to find interesting patterns and novel attacks.

Usable and deployable solution. One future work direction is to deploy our dependence analysis solution for practical use. We plan to extend our solution for real-time triggering relation inference and online detection. Besides, an advanced online supervised learning algorithm for training the triggering relation model is desirable to improve the efficiency and usability. Also, how to adopt our solution to a large-scale network (e.g. LAN) needs more research efforts before the enterprise-wide deployment. Last, a graphic user interface for demonstrating the network stack and complex security policies can significantly help security administrators to investigate network incidents.

Bibliography

[1] Alexa top 1000 sites. http://www.alexa.com/topsites.

[2] An SSL proxying technique to sniff HTTPS packets. http://bit.ly/SSLProxying.

[3] A first: Hacked sites with Android drive-by download malware. http://bit.ly/AndroidDBD.

[4] Botnet Pony 1.9 malware. http://bit.ly/PonyBot.

[5] D3.js: a JavaScript library for visualizing data with HTML, SVG, and CSS. http://d3js.org.

[6] DARPA intrusion detection evaluation. http://bit.ly/DARPAData.

[7] DNScat, a tool to tunnel traffic through dns servers. http://bit.ly/DNSCAT.

[8] How people spend their time online. 2012. http://bit.ly/PeopleOnline.

[9] Kidlogger. http://kidlogger.net/about.html.

[10] Malware attack hits thousands of Yahoo users per hour. 2012. http://bit.ly/Mal2Yahoo.

[11] Malware domain list. http://www.malwaredomainlist.com.

[12] Official Android store. https://play.google.com/store/apps.

[13] Panda security report. 2014. http://bit.ly/PandaReport.

[14] Protecting privacy with referrers. 2010. http://bit.ly/ReferrerPrivacy.

[15] Suspicious domains from sans education. http://bit.ly/SuspiciousDom.

[16] TCPFLOW 1.3. https://github.com/simsong/tcpflow.

[17] Tlogger, an firefox extension. http://dubroy.com/tlogger/.

[18] Zeus tracker domain blocklist. https://zeustracker.abuse.ch/blocklist.php.

[19] Y. Aafer, W. Du, and H. Yin. DroidAPIMiner: Mining API-level features for robust malware detection in Android. In *9th International Conference on Security and Privacy in Communication Networks (SecureComm)*, pages 86–103. 2013.

[20] H. Almohri, D. Yao, and D. Kafura. Process authentication for high system assurance. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 11(2):168–180, 2014.

[21] L. Alvisi, K. Bhatia, and K. Marzullo. Causality tracking in causal message-logging protocols. *Distributed Computing*, 15(1):1–15, 2002.

[22] L. Alvisi and K. Marzullo. Message logging: Pessimistic, optimistic, causal, and optimal. *IEEE Transactions on Software Engineering*, 24(2):149–159, 1998.

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

[24] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. DREBIN: effective and explainable detection of Android malware in your pocket. In *21st Annual Network and Distributed System Security Symposium (NDSS)*, 2014.

[25] H. Asai, K. Fukuda, and H. Esaki. Traffic Causality Graphs: Profiling network applications through temporal and spatial causality of flows. In *23rd International Teletraffic Congress (ITC)*, pages 95–102, 2011.

[26] 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 (WSDM)*, pages 635–644, 2011.

[27] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[28] P. Bahl, R. Chandra, A. G. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proceedings of the ACM SIGCOMM 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 13–24, 2007.

[29] A. Bates, B. Mood, J. Pletcher, H. Pruse, M. Valafar, and K. Butler. Detecting co-residency with active traffic analysis techniques. In *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop (CCSW)*, pages 1–12, 2012.

[30] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell. State of the art: Automated black-box web application vulnerability testing. In *IEEE Symposium on Security and Privacy (SP)*, pages 332–345, 2010.

[31] D. M. Best, S. Bohn, D. Love, A. Wynne, and W. A. Pike. Real-time visualization of network behaviors for situational awareness. In *7th International Symposium on Visualization for Cyber Security (VizSec)*, pages 79–90, 2010.

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

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

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

[35] A. Boschetti, L. Salgarelli, C. Muelder, and K.-L. Ma. TVi: a visual querying system for network monitoring and anomaly detection. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security (VizSec)*, page 1, 2011.

[36] L. Briesemeister, S. Cheung, U. Lindqvist, and A. Valdes. Detection, correlation, and visualization of attacks against critical infrastructure systems. In *Eighth Annual Conference on Privacy, Security and Trust (PST)*, pages 15–22, 2010.

[37] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for Android. In *Proceedings of the 1st ACM Workshop Security and Privacy in Smartphones and Mobile Devices (SPSM)*, pages 15–26, 2011.

[38] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys*, 41, July 2009.

[39] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl. Automating network application dependency discovery: Experiences, limitations, and new solutions. In *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 117–130, 2008. USENIX Association.

[40] H.-K. Choi and J. O. Limb. A behavioral model of web traffic. In *Proceeding of the 7th International Conference on Network Protocols (ICNP)*, pages 327–334, 1999.

[41] M. Christodorescu, S. Jha, and C. Kruegel. Mining specifications of malicious behavior. In *Proceeding of the 1st Annual India Software Engineering Conference (ISEC)*, pages 5–14, 2008.

[42] G. Conti, K. Abdullah, J. Grizzard, J. Stasko, J. A. Copeland, M. Ahamad, H. L. Owen, and C. Lee. Countering security information overload through alert and packet visualization. *IEEE Computer Graphics and Applications*, 26(2):60–70, 2006.

[43] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde. Can't you hear me knocking: Identification of user actions on Android apps via traffic analysis. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 297–304, 2015.

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

[45] 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 (WWW)*, pages 281–290, 2010.

[46] J. Crussell, C. Gibler, and H. Chen. Attack of the clones: Detecting cloned applications on Android markets. In *Proceeding of the 17th European Symposium on Research in Computer Security (ESORICS)*, pages 37–54. Springer, 2012.

[47] W. Cui, Y. H. Katz, and W.-T. Tan. BINDER: An extrusion-based break-in detector for personal computers. In *Proceedings of USENIX Annual Technical Conference (ATC)*, pages 363–366, 2005.

[48] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song. NetworkProfiler: Towards automatic fingerprinting of Android apps. In *Proceedings of the IEEE INFOCOM*, pages 809–817, 2013.

[49] D. E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, (2):222–232, February 1987.

[50] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 973–978, 2001.

[51] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems*, 32(2):5, 2014.

[52] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of Android application security. In *Proceeding of the 20th USENIX Security Symposium*, 2011.

[53] A. Endert, P. Fiaux, and C. North. Semantic interaction for sensemaking: inferring analytical reasoning for model steering. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2879–2888, 2012.

[54] A. Endert, P. Fiaux, and C. North. Semantic interaction for visual text analytics. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI)*, pages 473–482, 2012.

[55] J. Erman, M. Arlitt, and A. Mahanti. Traffic classification using clustering algorithms. In *Proceedings of the 2nd Annual ACM Workshop on Mining Network Data (MineNet)*, pages 281–286, 2006.

[56] J. Erman, A. Mahanti, M. F. Arlitt, I. Cohen, and C. L. Williamson. Semi-supervised network traffic classification. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 369–370, 2007.

[57] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, June 1999.

[58] G. A. Fink, P. Muessig, and C. North. Visual correlation of host processes and network traffic. In *IEEE Workshop on Visualization for Computer Security (VizSec)*, pages 11–19, 2005.

[59] J. Fonseca, M. Vieira, and H. Madeira. Vulnerability & attack injection for web applications. In *Proceedings of the 2009 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 93–102, 2009.

[60] M. Fredrikson, M. Christodorescu, J. Giffin, and S. Jhas. A declarative framework for intrusion analysis. In *Cyber Situational Awareness*, pages 179–200. 2010.

[61] X. Fu, R. Ren, J. Zhan, W. Zhou, Z. Jia, and G. Lu. LogMaster: mining event correlations in logs of large-scale cluster systems. In *IEEE 31st Symposium on Reliable Distributed Systems (SRDS)*, pages 71–80, 2012.

[62] S. Garera, N. Provos, M. Chew, and A. D. Rubin. A framework for detection and measurement of phishing attacks. In *Proceedings of the 2007 ACM Workshop on Recurring Malcode (WORM)*, pages 1–8, 2007.

[63] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck. Structural detection of Android malware using embedded call graphs. In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security (AISec)*, pages 45–54, 2013.

[64] L. Getoor and C. P. Diehl. Link mining: a survey. *SIGKDD Explorations*, 7(2):3–12, December 2005.

[65] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung. Vision: automated security validation of mobile apps at app markets. In *Proceedings of the Second International Workshop on Mobile Cloud Computing and Services (MCS)*, pages 21–26.

[66] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, pages 101–112, 2012.

[67] T. M. Green, W. Ribarsky, and B. Fisher. Visual analytics for complex concepts using a human cognition model. In *IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pages 91–98, 2008.

[68] 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*, pages 139–154, 2008.

[69] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting malware infection through IDS-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium*, 2007.

[70] D. Gugelmann, F. Gasser, B. Ager, and V. Lenders. Hviz: HTTP(S) traffic aggregation and visualization for network forensics. *Digital Investigation*, 12(Supplement-1):1–11, 2015.

[71] 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.

[72] H. Hao, V. Singh, and W. Du. On the effectiveness of API-level access control using bytecode rewriting in Android. In *8th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 25–36, 2013.

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

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

[75] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network monitoring using traffic dispersion graphs (TDGs). In *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 315–320, 2007.

[76] T. Itoh, H. Takakura, A. Sawada, and K. Koyamada. Hierarchical visualization of network intrusion detection data. *IEEE Computer Graphics and Applications*, 26(2):40–47, 2006.

[77] Y. Jang, S. P. Chung, B. D. Payne, and W. Lee. Gyrus: A framework for user-intent monitoring of text-based networked applications. In *21st Annual Network and Distributed System Security Symposium (NDSS)*, 2014.

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

[79] J. Jung, A. Sheth, B. Greenstein, D. Wetherall, G. Maganis, and T. Kohno. Privacy oracle: a system for finding application leaks with black box differential testing. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security (CCS)*, pages 279–288, 2008.

[80] L. Kagal, C. Hanson, and D. J. Weitzner. Using dependency tracking to provide explanations for policy management. In *9th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 54–61, 2008.

[81] 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.

[82] S. Kairam, D. MacLean, M. Savva, and J. Heer. GraphPrism: compact visualization of network structure. In *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI)*, pages 498–505, 2012.

[83] S. Kandula, R. Chandra, and D. Katabi. What's going on? Learning communication rules in edge networks. In *Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, August 2008.

[84] D. A. Keim, F. Mansmann, J. Schneidewind, and T. Schreck. Monitoring network traffic with radial traffic analyzer. In *IEEE Symposium on Visual Analytics Science And Technology (VAST)*, pages 123–128, 2006.

[85] A. Keller, U. Blumenthal, and G. Kar. Classification and computation of dependencies for distributed management. In *Proceedings of the 5th IEEE Symposium on Computers and Communications (ISCC)*, pages 78–83, 2000.

[86] 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.

[87] C. Kintzel, J. Fuchs, and F. Mansmann. Monitoring large ip spaces with clockview. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security (VizSec)*, page 2, 2011.

[88] A. Kobsa. User experiments with tree visualization systems. In *10th IEEE Symposium on Information Visualization (InfoVis)*, pages 9–16, 2004.

[89] H. Koike and K. Ohno. SnortView: Visualization system of snort logs. In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security (VizSec/DMSEC)*, pages 143–147, 2004.

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

[91] Z. Li, Y. Tang, Y. Cao, V. Rastogi, Y. Chen, B. Liu, and C. Sbisa. WebShield: Enabling various web defense techniques without client side modifications. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2011.

[92] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. G. Greenberg, and Y.-M. Wang. WebProphet: Automating performance prediction for web services. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 143–158, 2010.

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

[94] 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 (MALWARE)*, pages 47–54, 2009.

[95] Y.-D. Lin, Y.-C. Lai, C.-H. Chen, and H.-C. Tsai. Identifying Android malicious repackaged applications by thread-grained system call sequences. *Computers & Security*, 39:340–350, 2013.

[96] 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)*, pages 967–974, 2006.

[97] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang. CHEX: statically vetting Android apps for component hijacking vulnerabilities. In *ACM Conference on Computer and Communications Security (CCS)*, pages 229–240, 2012.

[98] L. Lu, V. Yegneswaran, P. Porras, and W. Lee. BLADE: An attack-agnostic approach for preventing drive-by malware infections. In *Proceedings of 17th ACM Conference on Computer and Communications Security (CCS)*, pages 440–450, 2010.

[99] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1245–1254, 2009.

[100] R. Marty. *Applied security visualization*. 2009.

[101] A. W. Moore and D. Zuev. Internet traffic classification using Bayesian analysis techniques. pages 50–60, 2005.

[102] A. Moshchuk, T. Bragin, and D. Deville. SpyProxy: Execution-based detection of malicious web content. In *Proceedings of the 16th USENIX Security Symposium*, 2007.

[103] C. Mulliner. Privacy leaks in mobile phone internet access. In *14th International Conference on Intelligence in Next Generation Networks (ICIN)*, pages 1–6, 2010.

[104] D. Muthukumaran, A. Sawani, J. Schiffman, B. M. Jung, and T. Jaeger. Measuring integrity on mobile phone systems. In *Proceedings of the 13th ACM symposium on Access control models and technologies (SACMAT)*, pages 155–164, 2008.

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

[106] C. Neasbitt, B. Li, R. Perdisci, L. Lu, K. Singh, and K. Li. WebCapsule: Towards a lightweight forensic engine for web browsers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 133–145, 2015.

[107] C. Neasbitt, R. Perdisci, K. Li, and T. Nelms. ClickMiner: Towards forensic reconstruction of user-browser interactions from network traces. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1244–1255, 2014.

[108] T. Nelms, R. Perdisci, M. Antonakakis, and M. Ahamad. WebWitness: investigating, categorizing, and mitigating malware download paths. In *Proceedings of the 24th USENIX Conference on Security Symposium (USENIX Security)*, pages 1025–1040, 2015.

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

[110] D. Patnaik, S. Laxman, and N. Ramakrishnan. Discovering excitatory relationships using dynamic Bayesian networks. *Knowledge and Information Systems*, 29(2):273–303, 2011.

[111] R. Perdisci, I. Corona, D. Dagon, and W. Lee. Detecting malicious flux service networks through passive analysis of recursive DNS traces. In *25th Annual Computer Security Applications Conference (ACSAC)*, pages 311–320, 2009.

[112] C. Qian, X. Luo, Y. Le, and G. Gu. VulHunter: Toward discovering vulnerabilities in Android applications. *IEEE Micro*, 35(1):44–53, 2015.

[113] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan. User-driven access control: Rethinking permission granting in modern operating systems. In *IEEE Symposium on Security and privacy (SP)*, pages 224–238, 2012.

[114] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield. Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification. In *Proceedings of the 4th ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 135–148, 2004.

[115] F. Roveta, G. Caviglia, L. Di Mario, S. Zanero, F. Maggi, and P. Ciuccarelli. BURN: Baring unknown rogue networks. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security (VizSec)*, pages 1–10, 2011.

[116] J. Rutkowska. Introducing stealth malware taxonomy. 2006.

[117] D. Shackleford. Blind as a bat? supporting packet decryption for security scanning (white paper). November 2012.

[118] U. Shankar and C. Karlof. Doppelganger: Better browser privacy without the bother. In *ACM Conference on Computer and Communications Security (CCS)*, pages 154–167, 2006.

[119] S.-P. Shieh and V. D. Gligor. On a pattern-oriented model for intrusion detection. *IEEE Transactions on Knowledge and Data Engineering*, 9(4), July 1997.

[120] H. Shiravi, A. Shiravi, and A. A. Ghorbani. A survey of visualization systems for network security. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1313–1329, 2012.

[121] J. Shirley and D. Evans. The user is not the enemy: Fighting malware by tracking user intentions. In *Proceedings of the 2008 Workshop on New Security Paradigms (NSPW)*, pages 33–45, 2008.

[122] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics (TOG)*, 11(1):92–99, 1992.

[123] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, T. Grance, L. T. Heberlein, C.-L. Ho, K. N. Levitt, B. Mukherjee, D. L. Mansur, K. L. Pon, and S. E. Smaha. A system for distributed intrusion detection. *COMPCOM Spring*, 91:170–176, 1991.

[124] G. Snedecor. *Statistical methods*. Iowa State University Press, 1989.

[125] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy (SP)*, pages 305–316, 2010.

[126] H. Song, C. Muelder, and K.-L. Ma. Crucial nodes centric visual monitoring and analysis of computer networks. In *ASE International Conference on Cyber Security (CyberSecurity)*, pages 16–23, 2012.

[127] A. Srivastava and J. T. Giffin. Automatic discovery of parasitic malware. In *Proceeding of the 13th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 97–117, 2010.

[128] 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.

[129] G. Stringhini, C. Kruegel, and G. Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security (CCS)*, pages 133–144, 2013.

[130] M. Sun and G. Tan. NativeGuard: Protecting Android applications from third-party native libraries. In *Proceedings of the 7th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, pages 165–176, 2014.

[131] H. S. Teng, K. Chen, and S. C.-Y. Lu. Adaptive real-time anomaly detection using inductively generated sequential patterns. In *Proceedings of IEEE Symposium on Security and Privacy (SP)*, pages 278–284, 1990.

[132] M. Ter Louw, J. Lim, and V. Venkatakrishnan. Enhancing web browser security against malware extensions. *Journal in Computer Virology*, 4:179–195, 2008.

[133] T. Vidas, J. Tan, J. Nahata, C. L. Tan, N. Christin, and P. Tague. A5: Automated analysis of adversarial Android applications. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, pages 39–50, 2014.

[134] K. Wang and S. J. Stolfo. Anomalous payload-based network intrusion detection. In *7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 203–222, 2004.

[135] X. Wang and M. K. Reiter. Using web-referral architectures to mitigate denial-of-service threats. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 7(2):203–216, 2010.

[136] Y. Wang, S. Hariharan, C. Zhao, J. Liu, and W. Du. Compac: Enforce component-level access control in Android. In *Fourth ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 25–36, 2014.

[137] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. ProfileDroid: multi-layer profiling of Android applications. In *18th Annual International Conference on Mobile Computing and Networking (Mobicom)*, pages 137–148, 2012.

[138] C. Whittaker, B. Ryner, and M. Nazif. Large-scale automatic classification of phishing pages. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2010.

[139] N. Williams, S. Zander, and G. Armitage. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *Computer Communication Review*, 36(5):5–16, 2006.

[140] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu. DroidMat: Android malware detection through manifest and API calls tracing. In *Seventh Asia Joint Conference on Information Security (Asia JCIS)*, pages 62–69, 2012.

[141] G. Xie, M. Iliofotou, T. Karagiannis, M. Faloutsos, and Y. Jin. ReSurf: Reconstructing web-surfing activity from network traffic. In *IFIP Networking Conference*, pages 1–9, 2013.

[142] H. Xiong, P. Malhotra, D. Stefan, C. Wu, and D. Yao. User-assisted host-based detection of outbound malware traffic. In *Proceedings of International Conference on Information and Communications Security (ICICS)*, pages 293–307, 2009.

[143] K. Xu, P. Butler, S. Saha, and D. Yao. DNS for massive-scale command and control. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 10(3):143–153, 2013.

[144] K. Xu, H. Xiong, C. Wu, D. Stefan, and D. Yao. Data-provenance verification for secure hosts. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 9(2):173–183, 2012.

[145] K. Xu, D. Yao, Q. Ma, and A. Crowell. Detecting infection onset with behavior-based policies. In *Proceedings of the Fifth International Conference on Network and System Security (NSS)*, pages 57–64, 2011.

[146] Q. Xu, Y. Liao, S. Miskovic, M. Baldi, Z. M. Mao, A. Nucci, and T. Andrews. Automatic generation of mobile app signatures from traffic observations. In *Proceedings of the IEEE INFOCOM*, pages 1481–1489, 2015.

[147] L. Yan and H. Yin. DroidScope: Seamlessly reconstructing the OS and dalvik semantic views for dynamic Android malware analysis. In *Proceedings of the 21th USENIX Security Symposium*, pages 569–584, 2012.

[148] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras. DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in Android applications. In *19th European Symposium on Research in Computer Security (ESORICS)*, pages 163–182. 2014.

[149] J. Yang and D. Evans. Dynamically inferring temporal properties. In *Proceedings of the 2004 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis For Software Tools and Engineering (PASTE)*, pages 23–28, 2004.

[150] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck. AppContext: Differentiating malicious and benign mobile app behaviors using context. In *37th IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 303–313, 2015.

[151] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang. AppIntent: Analyzing sensitive data transmission in Android for privacy leakage detection. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security (CCS)*, pages 1043–1054, 2013.

[152] A. Zand, G. Vigna, R. Kemmerer, and C. Kruegel. Rippler: Delay injection for service dependency detection. In *Proceedings of the IEEE INFOCOM*, pages 2157–2165, 2014.

[153] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu. ViewDroid: Towards obfuscation-resilient mobile application repackaging detection. In *7th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, pages 25–36, 2014.

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

[155] H. Zhang, M. Sun, D. Yao, and C. North. Visualizing traffic causality for analyzing network anomalies. In *Proceedings of International Workshop on Security and Privacy Analytics (IWSPA)*, pages 37–42, 2015.

[156] H. Zhang, D. Yao, and N. Ramakrishnan. A semantic-aware approach to reasoning about network traffic relations. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security (CCS)*, pages 1375–1378, 2013.

[157] H. Zhang, D. Yao, and N. Ramakrishnan. 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)*, pages 39–50, 2014.

[158] N. Zhang, K. Yuan, M. Naveed, X. Zhou, and X. Wang. Leave me alone: App-level protection against runtime information gathering on Android. In *IEEE Symposium on Security and Privacy (SP)*, pages 915–930, 2015.

[159] X. Zhang, A. Ahlawat, and W. Du. AFrame: Isolating advertisements from mobile applications in Android. In *Annual Computer Security Applications Conference (ACSAC)*, pages 9–18, 2013.

[160] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou. Smartdroid: an automatic system for revealing UI-based trigger conditions in Android applications. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices (SPSM)*, pages 93–104, 2012.

[161] Y. Zhou and X. Jiang. Dissecting Android malware: Characterization and evolution. In *IEEE Symposium on Security and Privacy (SP)*, pages 95–109, 2012.

[162] L. Zomlot, S. Chandran, D. Caragea, and X. Ou. Aiding intrusion analysis using machine learning. In *12th International Conference on Machine Learning and Applications (ICMLA)*, volume 2, pages 40–47, 2013.