

REPORT DOCUMENTATION PAGE			Form Approved OMB NO. 0704-0188		
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA, 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 23-03-2018		2. REPORT TYPE Final Report		3. DATES COVERED (From - To) 1-Aug-2014 - 31-Jul-2017	
4. TITLE AND SUBTITLE Final Report: Causality-Based Traffic Reasoning for Securing Large-Scale Networks			5a. CONTRACT NUMBER W911NF-14-1-0535		
			5b. GRANT NUMBER		
			5c. PROGRAM ELEMENT NUMBER 611102		
6. AUTHORS			5d. PROJECT NUMBER		
			5e. TASK NUMBER		
			5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAMES AND ADDRESSES Virginia Polytechnic Institute & State Univ North End Center, Suite 4200 300 Turner Street, NW Blacksburg, VA 24061 -0001			8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES) U.S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709-2211			10. SPONSOR/MONITOR'S ACRONYM(S) ARO		
			11. SPONSOR/MONITOR'S REPORT NUMBER(S) 64804-CS-YIP.12		
12. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Danfeng Yao
a. REPORT UU	b. ABSTRACT UU	c. THIS PAGE UU			19b. TELEPHONE NUMBER 540-231-7787

RPPR Final Report

as of 24-Jan-2019

Agency Code:

Proposal Number: 64804CSYIP
INVESTIGATOR(S):

Agreement Number: W911NF-14-1-0535

Name: Danfeng Yao
Email: danfeng@vt.edu
Phone Number: 5402317787
Principal: Y

Organization: **Virginia Polytechnic Institute & State University**

Address: North End Center, Suite 4200, Blacksburg, VA 240610001

Country: USA

DUNS Number: 003137015

EIN: 546001805

Report Date: 31-Oct-2017

Date Received: 23-Mar-2018

Final Report for Period Beginning 01-Aug-2014 and Ending 31-Jul-2017

Title: Causality-Based Traffic Reasoning for Securing Large-Scale Networks

Begin Performance Period: 01-Aug-2014

End Performance Period: 31-Jul-2017

Report Term: 0-Other

Submitted By: Danfeng Yao

Email: danfeng@vt.edu

Phone: (540) 231-7787

Distribution Statement: 1-Approved for public release; distribution is unlimited.

STEM Degrees: 1

STEM Participants: 3

Major Goals: Our project goal is to detect clandestine malicious and unexplained network activities due to stealthy advanced persistent threats (APTs) and spyware activities in organizational networks (supporting both Android and Linux systems). This problem is challenging because there are usually not much difference between stealthy APT activities and normal traffic, in terms of volume, appearance, pattern, etc. We aim to demonstrate a new triggering relation discovery technique to construct the request-level causality structure in network traffic.

Our high-level approach is two-fold:

- 1) to detect anomalies based on the causal relations between network activities and user actions (assuming that user-triggered network activities are legitimate), and
- 2) to design machine learning algorithms to compute causal relations (specifically triggering-relation graphs) by analyzing the attributes of network traffic.

Prior to our work, this approach had not previously been systematically studied in the literature.

We anticipate these findings to impact future army capabilities in multiple ways, including substantially improving the accountability of network activities and improving the resilience against APT, spyware, and other clandestine malicious activities.

Accomplishments: Our main technical accomplishments are summarized as follows.

- a) We formalized the problem of triggering relation graph (TRG) discovery in networks and defined a root-trigger security policy for using TRG to detect malware activities.
- b) We invented two approaches for discovering the triggering-relations among network events. i) We designed the first discovery algorithm based on empirically derived causal rules. ii) We invented another advanced learning-based approach based on an innovative data-pairing operation.
- c) We achieved high accuracy (95-100%) in terms of identifying triggering relations when evaluating with synthetic and real-world network traffic, including HTTP, DNS, and TCP traffic.

RPPR Final Report

as of 24-Jan-2019

d) Our prototype on Android for mobile traffic also achieved 99% accuracy.

e) Our visualization prototype enables the compact display of TRG that maximally utilizes the computer screen of security analysts.

Next, we describe the five (5) technical achievements in more details. The figures and references are shown in the uploaded PDF file.

a) We gave the first formal definition for the triggering relation discovery problem [1]. Intuitively, given a set of events, the triggering relation discovery problem is to identify the order of causal relations among the events, i.e., in what order some events triggered other events to occur. We are particularly interested in network events and user actions. Building a triggering relation graph as depicted in Figure 1 can be used to detect suspicious network or system activities that do not have valid triggering relations, e.g., events not triggered (directly or indirectly) by any valid user inputs.

Figure 2 illustrates that a triggering relation graph nicely organizes and structures user events and network events (HTTP and DNS) in normal legitimate operations. The graph can be applied to detect various security threats, once policies are defined. The policies describe properties of valid events (e.g., with a valid user event as its root trigger).

b) We invented two approaches for discovering the triggering relations among network events [2]. First, we designed a discovery algorithm based on empirically derived rules. In the rule-based approach, we inspected the temporal, semantic, and process-related attributes to reveal the causal relations among network requests. Then, we demonstrated an advanced learning-based approach. A new feature extraction method is introduced as the pairing operation. The pairing operation performs pairwise attribute comparisons, enabling the use of binary classifiers for our triggering relation discovery. We have received a US patent [1] on this ML-based causality discovery approach.

c) We thoroughly conducted experimental evaluations on the accuracy of the detection algorithms (in terms of conventional precision and recall metrics, as well as customized metrics such as root trigger correctness) [2]. In addition, we demonstrated the detection of several categories of real-world threats, including:

- 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).

d) Our causality analysis prototype for Android systems detects malicious app activities as shown in Figure 4 [3].

e) We recognize the importance of human-in-the-loop detection and the need for producing usable tools and GUI interfaces for security analysts to take the full advantage of the causality analysis technology. We designed and developed a visualization tool with a special visual locality property that supports different levels of visual-based querying and reasoning required for the sensemaking process on complex network data. Leveraging the domain knowledge, security analysts can use our tool to identify abnormal network activities and patterns due to attacks or stealthy malware. We conducted a user study that

RPPR Final Report as of 24-Jan-2019

confirms our tool can enhance the readability and perceptibility of the dependency for host-based network traffic [4].

Training Opportunities: This project gave me the opportunity to train multiple graduate and undergraduate students, including Hao Zhang, Ke Tian, and Hannah Roth. It helped the PI to graduate one PhD student (Hao Zhang) from Virginia Tech.

Results Dissemination: The project produced multiple top-tier security publications, which are shown below.

Hao Zhang, Danfeng Yao, Naren Ramakrishnan, and Zhibin Zhang. Causality Reasoning about Network Events for Detecting Stealthy Malware Activities. Computers & Security (C&S). 58: 180-198. Elsevier. 2016.

Hao Zhang, Danfeng Yao, and Naren Ramakrishnan. Causality-based Sensemaking of Network Traffic for Android Application Security. In Proceedings of 9th ACM Workshop on Artificial Intelligence and Security (AISec), co-located with the 23rd ACM Conference on Computer and Communications (CCS). October 28, 2016, Hofburg Palace, Vienna, Austria.

Hao Zhang, Maoyuan Sun, Danfeng Yao, and Chris North. Visualizing Traffic Causality for Analyzing Network Anomalies. In Proceedings of International Workshop on Security and Privacy Analytics (SPA), co-located with ACM CODASPY. San Antonio, TX. Mar. 2015.

Danfeng Yao, Xiaokui Shu, Long Cheng, and Salvatore J. Stolfo. Anomaly Detection as a Service: Challenges, Advances, and Opportunities. In Information Security, Privacy, and Trust Series. Morgan & Claypool. Oct. 2017.

Honors and Awards: This YIP award plays an extremely important role in helping PI Yao establish her technical leadership in the causality-based security field. During this award period, PI Yao was elected the Elizabeth and James E. Turner Jr. '56 Faculty Fellow at Virginia Tech in 2016.

Protocol Activity Status:

Technology Transfer: Our project has produced one approved US patent:

Danfeng Yao and Hao Zhang. Detection of Stealthy Malware Activities with Traffic Causality and Scalable Triggering Relation Discovery. U.S. Patent No.9,888,030. Granted on Feb 6th., 2018. (Virginia Tech is in contact with a Charlottesville-based big data security company on commercialization.)

PARTICIPANTS:

Participant Type: PD/PI

Participant: Danfeng (Daphne) Yao

Person Months Worked: 1.00

Funding Support:

Project Contribution:

International Collaboration:

International Travel:

National Academy Member: N

Other Collaborators:

Participant Type: Graduate Student (research assistant)

Participant: Hao Zhang

Person Months Worked: 9.00

Funding Support:

Project Contribution:

International Collaboration:

International Travel:

RPPR Final Report

as of 24-Jan-2019

National Academy Member: N
Other Collaborators:

Participant Type: Graduate Student (research assistant)

Participant: Ke Tian

Person Months Worked: 3.00

Funding Support:

Project Contribution:

International Collaboration:

International Travel:

National Academy Member: N

Other Collaborators:

Participant Type: Undergraduate Student

Participant: Hannah Roth

Person Months Worked: 1.00

Funding Support:

Project Contribution:

International Collaboration:

International Travel:

National Academy Member: N

Other Collaborators:

ARTICLES:

Publication Type: Journal Article

Peer Reviewed: Y

Publication Status: 1-Published

Journal: 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS).

Publication Identifier Type: Other

Publication Identifier:

Volume:

Issue:

First Page #:

Date Submitted: 8/30/16 12:00AM

Date Published: 8/30/16 4:11PM

Publication Location:

Article Title: Unearthing Stealthy Program Attacks Buried in Extremely Long Execution Paths

Authors: Xiaokui Shu, Danfeng Yao, Naren Ramakrishnan.

Keywords: program anomaly detection, exploit, data mining, outlier detection, clustering, non-control data attack, frequency anomaly, montage anomaly

Abstract: Modern stealthy exploits can achieve attack goals without introducing illegal control flows, e.g., tampering with non-control data and waiting for the modified data to propagate and alter the control flow legally. Existing program anomaly detection systems focusing on legal control flow attestation and short call sequence verification are inadequate to detect such stealthy attacks. We point out the need to analyze program execution paths and discover event correlations in large-scale execution windows among millions of instructions. We propose an anomaly detection approach with two-stage machine learning algorithms to recognize diverse normal call-correlation patterns and detect program attacks at both inter- and intra-cluster levels. We implement a prototype of our approach and demonstrate its effectiveness against three real-world attacks and four synthetic anomalies with less than 0.01% false positive rates and 0.1~1.3 ms analysis overhead per behavior instance.

Distribution Statement: 1-Approved for public release; distribution is unlimited.

Acknowledged Federal Support: Y

RPPR Final Report as of 24-Jan-2019

Publication Type: Journal Article Peer Reviewed: Y **Publication Status:** 1-Published
Journal: Computers & Security
Publication Identifier Type: Other **Publication Identifier:**
Volume: **Issue:** **First Page #:**
Date Submitted: 8/30/16 12:00AM **Date Published:** 8/30/16 8:16PM
Publication Location:

Article Title: Causality reasoning about network events for detecting stealthy malware activities

Authors: Hao Zhang, Danfeng Yao, Naren Ramakrishnan, Zhibin Zhang.

Keywords: Causal analysis, malware traffic, host-based anomaly detection, machine learning, classification, accuracy, pairing, triggering relation, triggering relation discovery

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

Distribution Statement: 1-Approved for public release; distribution is unlimited.

Acknowledged Federal Support: Y

BOOKS:

Publication Type: Book Peer Reviewed: Y **Publication Status:** 1-Published
Publication Identifier Type: ISBN **Publication Identifier:** 1681731096
Book Edition: **Volume:** **Publication Year:** 2017 **Date Received:** 23-Mar-2018
Publication Location:
Publisher: Morgan & Claypool
Book Title: Anomaly Detection as a Service: Challenges, Advances, and Opportunities
Authors: Danfeng Yao, Xiaokui Shu, Long Cheng, Salvatore J. Stolfo
Editor:
Acknowledged Federal Support: Y

CONFERENCE PAPERS:

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: 2015 ACM International Workshop on International Workshop on Security and Privacy Analytics
Date Received: 23-Mar-2018 **Conference Date:** 04-Mar-2015 **Date Published:**
Conference Location: San Antonio, Texas, USA
Paper Title: Visualizing Traffic Causality for Analyzing Network Anomalies
Authors: Hao Zhang, Maoyuan Sun, Danfeng Yao, Chris North
Acknowledged Federal Support: Y

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2014)
Date Received: 23-Mar-2018 **Conference Date:** 20-Oct-2014 **Date Published:**
Conference Location: Miami, FL
Paper Title: Role-playing game for studying user behaviors in security: A case study on email secrecy
Authors: Kui Xu, Danfeng Yao, Manuel A. Perez-Quinones, Casey Link, E. Scott Geller
Acknowledged Federal Support: Y

RPPR Final Report

as of 24-Jan-2019

Publication Type: Conference Paper or Presentation

Publication Status: 0-Other

Conference Name: the 2016 ACM Workshop

Date Received: 23-Mar-2018

Conference Date: 28-Oct-2016

Date Published:

Conference Location: Vienna, Austria, ACM CCS Workshop on AISec

Paper Title: Causality-based Sensemaking of Network Traffic for Android Application Security

Authors: Hao Zhang, Danfeng Yao, Naren Ramakrishnan

Acknowledged Federal Support: **Y**

Publication Type: Conference Paper or Presentation

Publication Status: 1-Published

Conference Name: Proceedings of International Workshop on Security and Privacy Analytics (SPA), co-located with ACM CODASPY

Date Received: 23-Mar-2018

Conference Date: 17-Mar-2015

Date Published: 17-Mar-2015

Conference Location: San Antonio, TX

Paper Title: Visualizing Traffic Causality for Analyzing Network Anomalies

Authors: Hao Zhang, Maoyuan Sun, Danfeng Yao, Chris North

Acknowledged Federal Support: **Y**

DISSERTATIONS:

Publication Type: Thesis or Dissertation

Institution: Virginia Tech

Date Received: 23-Mar-2018

Completion Date: 10/8/15 11:15AM

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

Authors: Hao Zhang

Acknowledged Federal Support: **Y**

PATENTS:

Intellectual Property Type: Patent

Date Received: **23-Mar-2018**

Patent Title: Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery

Patent Abstract: A computer system for distinguishing user-initiated network traffic from malware-initiated network traffic

Patent Number: 9888030

Patent Country: USA

Application Date: 01-May-2014

Application Status: 3

Date Issued: 06-Feb-2018

Final Report of Causality-Based Traffic Reasoning for Securing Large-Scale Networks

Danfeng (Daphne) Yao
Virginia Tech
ARO YIP Project

Our project goal is to detect clandestine malicious and unexplained network activities due to stealthy advanced persistent threats (APTs) and spyware activities in organizational networks (supporting both Android and Linux systems). This problem is challenging because there are usually not much difference between stealthy APT activities and normal traffic, in terms of volume, appearance, pattern, etc.

Our high-level approach is two-fold: 1) to detect anomalies based on the causal relations between network activities and user actions (assuming that user-triggered network activities are legitimate), and 2) to design machine learning algorithms to compute causal relations (specifically triggering-relation graphs) by analyzing the attributes of network traffic.

This approach has not previously been systematically studied in the literature.

Our main technical accomplishments are summarized as follows.

- a) We formalized the problem of triggering relation graph (TRG) discovery in networks and defined a root-trigger security policy for using TRG to detect malware activities.
- b) We invented two approaches for discovering the triggering-relations among network events. i) We designed the first discovery algorithm based on empirically derived causal rules. ii) We invented another advanced learning-based approach based on an innovative data-pairing operation.
- c) We achieved high accuracy (95-100%) in terms of identifying triggering relations) when evaluating with synthetic and real-world network traffic, including HTTP, DNS, and TCP traffic.
- d) Our prototype on Android for mobile traffic also achieved 99% accuracy.
- e) Our visualization prototype enables the compact display of TRG that maximally utilizes the computer screen of security analysts.

We anticipate these findings to impact future army capabilities in multiple ways, including substantially improving the accountability of network activities and improving the resilience against APT, spyware, and other clandestine malicious activities.

Our project has produced one approved US patent [1], as well as multiple top-tier security publications [2,3,4,5]. This YIP award plays an extremely important role in helping PI Yao establish her technical leadership in the causality-based security field. During this award period, PI Yao was elected the Elizabeth and James E. Turner Jr. '56 Faculty Fellow at Virginia Tech in 2016.

Next, we describe the five (5) technical achievements in more details.

a) We gave the first formal definition for the triggering relation discovery problem [1]. Intuitively, given a set of events, the triggering relation discovery problem is to identify the order of causal relations among the events, i.e., in what order some events triggered other events to occur. We are particularly interested in network events and user actions. Building a triggering relation graph as depicted in Figure 1 can be used to detect suspicious network or system activities that do not have valid triggering relations, e.g., events not triggered (directly or indirectly) by any valid user inputs.

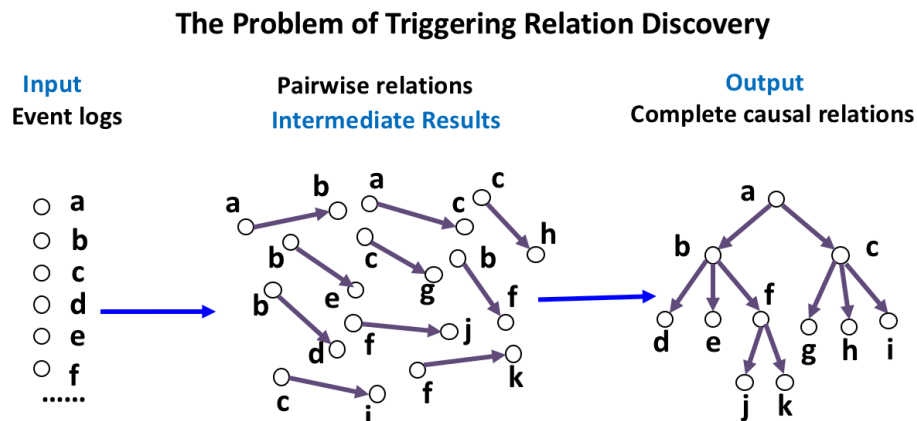


Figure 1. Triggering relation discovery problem.

Figure 2 illustrates that a triggering relation graph nicely organizes and structures user events and network events (HTTP and DNS) in normal legitimate operations. The graph can be applied to detect various security threats, once policies are defined. The policies describe properties of valid events (e.g., with a valid user event as its root trigger).

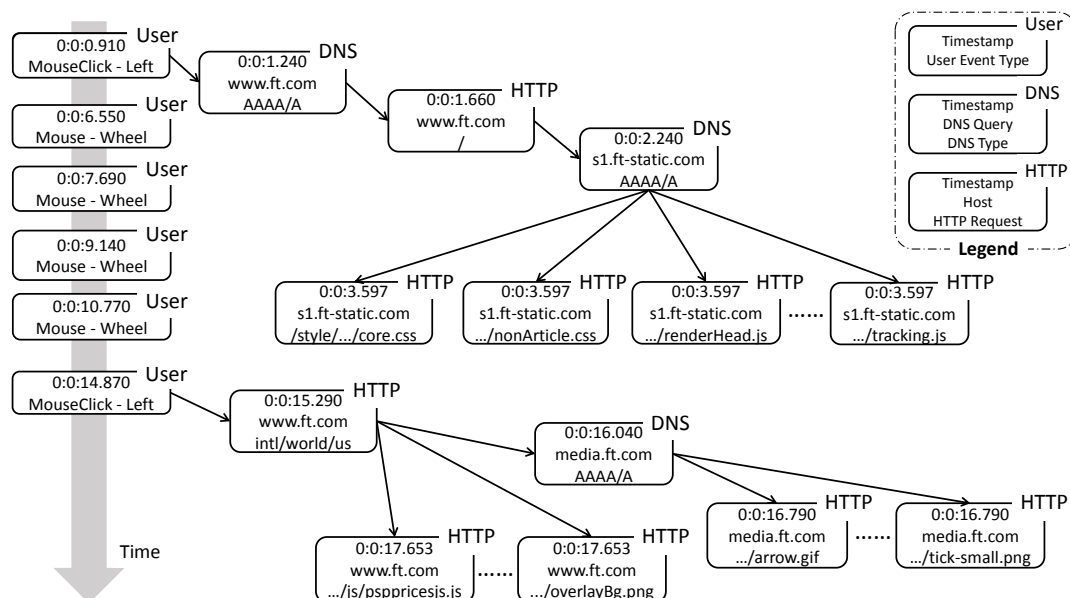


Figure 2. An example of a triggering relation graph.

b) We invented two approaches for discovering the triggering relations among network events [2]. First, we designed a discovery algorithm based on empirically derived rules. In

the rule-based approach, we inspected the temporal, semantic, and process-related attributes to reveal the causal relations among network requests. Then, we demonstrated an advanced learning-based approach. A new feature extraction method is introduced as the **pairing operation**. The pairing operation performs pairwise attribute comparisons, enabling the use of binary classifiers for our triggering relation discovery. We have received a US patent [1] on this ML-based causality discovery approach.

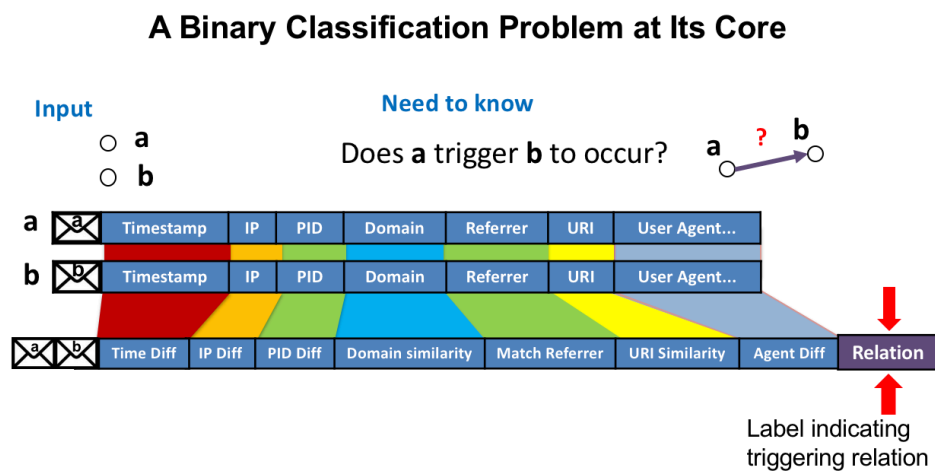


Figure 3. Illustration of our features and pairing operation in ML-based causality discovery approach.

c) We thoroughly conducted experimental evaluations on the accuracy of the detection algorithms (in terms of conventional precision and recall metrics, as well as customized metrics such as root trigger correctness) [2]. In addition, we demonstrated the detection of several categories of real-world threats, including:

- 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).

d) Our causality analysis prototype for Android systems detects malicious app activities as shown in Figure 4 [3].

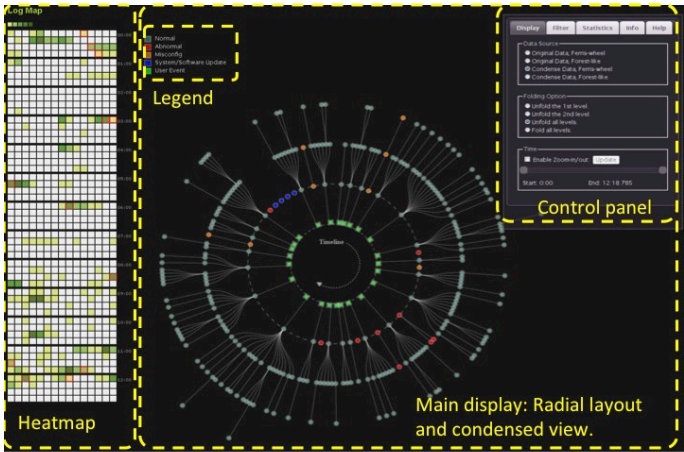
e) We recognize the importance of human-in-the-loop detection and the need for producing usable tools and GUI interfaces for security analysts to take the full advantage of the causality analysis technology. We designed and developed a visualization tool with a special visual locality property that supports different levels of visual-based querying and

reasoning required for the sensemaking process on complex network data. Leveraging the domain knowledge, security analysts can use our tool to identify abnormal network activities and patterns due to attacks or stealthy malware. We conducted a user study that confirms our tool can enhance the readability and perceptibility of the dependency for host-based network traffic [4].

#	Package Name	
1	<u>com.allen.txtiisz (ebook)</u>	Adware
2	<u>com.chenvx.tiltmazs</u>	Adware
3	<u>com.zxcalendar.chapp</u>	Adware
4	<u>wbfire.facts family</u>	Adware
5	<u>com.crazyapps family</u>	Trojan
6	<u>com.gfgfgg.dsdf</u>	Spyware
7	<u>com.gucdxidl.batterysaver</u>	Bot
8	<u>com.GoldDream.TingTing06i</u>	Bot
9	<u>com.wing.qingshongxry</u>	Spyware
10	<u>com.Punda.Free (game)</u>	Drive-by

Android malware activities detected

Figure 4. Example of Android malicious apps detected by causality analysis.



Visualization of TRG

Figure 5. Example of Android malicious apps detected by causality analysis.

In conclusion, this project has ended successfully with the demonstration of the feasibility of the novel causality based security technique and its versatility in detecting various types of network and system threats. A promising future direction is to explore the use of deep learning methods (such as LSTM) in recognizing complex causality patterns.

Our references are listed below. We also append the publications to the end of this report.

References:

- [1] Danfeng Yao and Hao Zhang. Detection of Stealthy Malware Activities with Traffic Causality and Scalable Triggering Relation Discovery. U.S. Patent No.9,888,030. Granted on Feb 6th., 2018. (Virginia Tech is in contact with a Charlottesville-based big data security company on commercialization.)
- [2] Hao Zhang, Danfeng Yao, Naren Ramakrishnan, and Zhibin Zhang. Causality Reasoning about Network Events for Detecting Stealthy Malware Activities. *Computers & Security (C&S)*. 58: 180-198. Elsevier. 2016.
- [3] Hao Zhang, Danfeng Yao, and Naren Ramakrishnan. Causality-based Sensemaking of Network Traffic for Android Application Security. In *Proceedings of 9th ACM Workshop on Artificial Intelligence and Security (AISec)*, co-located with the *23rd ACM Conference on Computer and Communications (CCS)*. October 28, 2016, Hofburg Palace, Vienna, Austria.
- [4] Hao Zhang, Maoyuan Sun, Danfeng Yao, and Chris North. Visualizing Traffic Causality for Analyzing Network Anomalies. In *Proceedings of International Workshop on Security and Privacy Analytics (SPA)*, co-located with *ACM CODASPY*. San Antonio, TX. Mar. 2015.
- [5] Danfeng Yao, Xiaokui Shu, Long Cheng, and Salvatore J. Stolfo. Anomaly Detection as a Service: Challenges, Advances, and Opportunities. In *Information Security, Privacy, and Trust Series*. Morgan & Claypool. Oct. 2017.

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/coseComputers
&
Security

Causality reasoning about network events for detecting stealthy malware activities¹



CrossMark

Hao Zhang^a, Danfeng (Daphne) Yao^{a,*}, Naren Ramakrishnan^a, Zhibin Zhang^b^a Department of Computer Science, Virginia Tech, Blacksburg, VA, USA^b Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

ARTICLE INFO

Article history:

Received 31 July 2015

Received in revised form 20

December 2015

Accepted 11 January 2016

Available online 21 January 2016

Keywords:

Network security

Anomaly detection

Stealthy malware

Traffic analysis

Dependence analysis

Machine learning classification

ABSTRACT

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

© 2016 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

1. Introduction

Recent advancements in information technology have raised concerns on the security risks posed by the prevalence of malicious software. A study showed that a significant portion of computers worldwide is infected with malware conducting clandestine activities (Panda Security Report). Malware may spy on the victim user, cause data exfiltration, and abuse the computer for conducting bot activities (e.g., command-and-control). The initial infection vector of most malware is usually through exploiting vulnerabilities of common networked soft-

ware, e.g., heap overflow vulnerability in a web browser or its plug-ins (Cova et al., 2010). Once the infection is successful (e.g., zero-day exploits), network requests from advanced malware may not exhibit distinct communication patterns. Because of this lack of signatures, pattern-based scanning is ineffective.

Compared to inspecting individual network requests, a more effective network security approach is to discover characteristic behavioral patterns in network event attributes. For example, BINDER (Cui et al., 2005) detects anomalous network activities on personal computers through analyzing the correlation in network events by the temporal and process information. BotMiner (Gu et al., 2008) performs a correlation

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

* Corresponding author. Tel.: +1 540 231 7787.

E-mail address: danfeng@vt.edu (D.(D.) Yao).

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

0167-4048/© 2016 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

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

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

We introduce the problem of *triggering relation discovery* in network traffic and describe its application in solving challenging network security problems, such as stealthy malware detection. Triggering relations of events provide contextual interpretations for the behaviors of systems and networks, illustrating why sequences of events occur and how they relate to each other. We develop rule- and learning-based methods that detect network activities of stealthy malware activities through reasoning their causal relationship. We design a new method *pairing* that produces special pairwise features in the learning-based approach, so that the discovery problem can be efficiently solved with existing binary classification methods (e.g., SVM). By enforcing a *root-trigger* policy, we can identify the suspicious events that lack of valid triggers, and thus ensure an application's correct responses to user activities.

The higher-level information such as the underlying relations or semantics of events is useful for human experts' cognition, reasoning, and decision-making in cyber security (Green et al., 2008; Zhang et al., 2015). Thus, analyzing relations among network events provides important insights for identifying general stealthy malicious activities. The causality offers the logical interpretation to the vast amount of otherwise structureless and contextless network events. Our work demonstrates that triggering relations among cyberspace events enables the network assurance with structural evidence of the hosts.

Our contributions are summarized as follows.

1. We formalize the problem of triggering relation discovery in network requests, the data structure of triggering relation graph, and their applications for detecting stealthy malware activities.
2. We present two approaches for discovering the triggering relations among network events. First, we design a discovery algorithm based on empirically derived rules. In the rule-based approach, we inspect the temporal, semantic, and process-related attributes to reveal the causal relations among network requests. Then, we propose an advanced learning-based approach. A new feature extraction method is introduced as the *pairing* operation. It performs pairwise

attribute comparisons, enabling the use of binary classifiers for our triggering relation discovery.

3. We adopt a new *root-trigger security policy* on discovered triggering relations for malware detection. This policy allows one to identify *vagabond events*, i.e., network events that do not have proper causes to justify their occurrences.
4. We extensively evaluate the classification and detection accuracy rates of our solution with DARPA dataset and real-world network traffic, including HTTP, DNS, and TCP traffic. Both rule- and learning-based methods are compared in identifying the dependencies and the learning-based one yields better prediction accuracy rates than rule-based one. We show that our dependence analysis is effective in detecting stealthy malware activities (e.g., HTTP-based malware and DNS bots).

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

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

1.1. Organization

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

2. Model and overview

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

2.1. TRG definitions and properties

Triggering relationship between event e_i and event e_j describes the temporal relation and causal relation between them, specifically e_i precedes e_j and e_i is the reason that directly causes e_j to occur. The specific semantics of triggering relation depend on the type of events and environment. An event may be defined

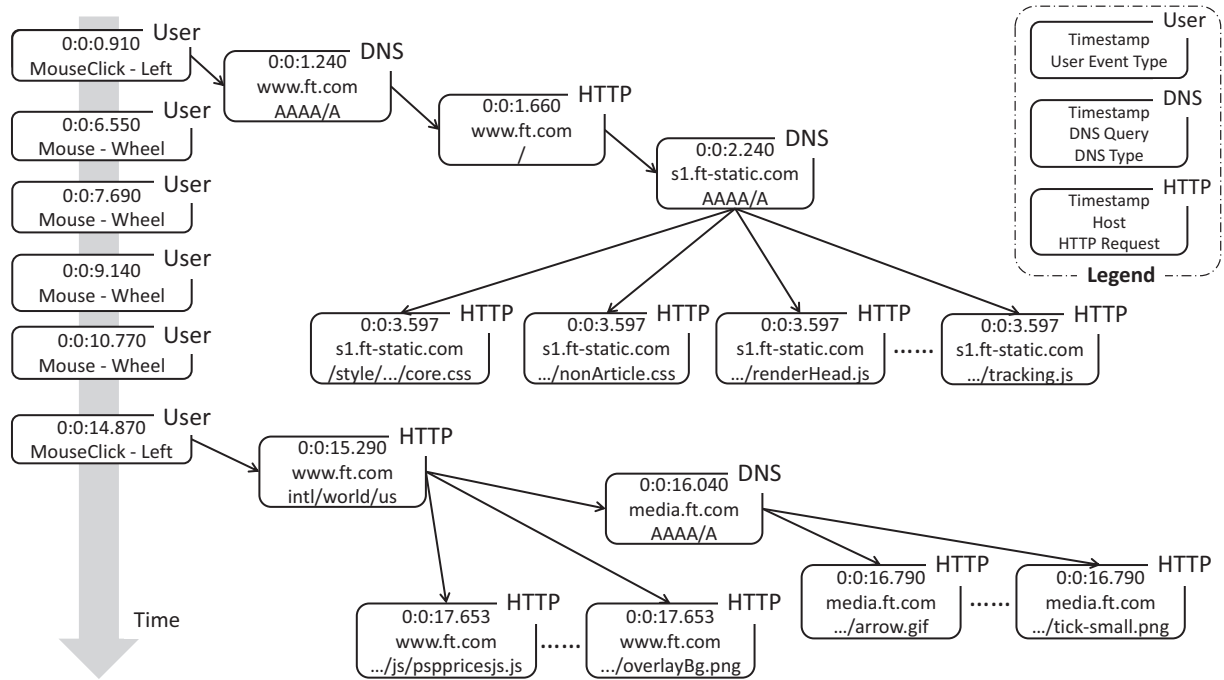


Fig. 1 – An example of triggering relation graph for outbound DNS and HTTP traffic on a host. The network and user events are denoted as rounded squares. The arrows represent the triggering relations between events.

at any relevant type or granularity, including user actions (e.g., keyboard stroke, mouse click), machine behaviors (e.g., network request, function call, system call, file system access), and higher-level operations and missions (e.g., database access, obtaining Kerberos authorization, distributing video to select users).

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

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

We summarize the properties of TRG as follows.

- **Attribute-based:** Each event in TRG has multiple attributes, describing its network and kernel information.
- **Expandable:** TRG is a forest of trees in arbitrary height. The degree of a node in a tree can be expanded as it grows. The legitimacy of the tree is determined by its root, i.e., the dependent nodes are benign as long as the root is legitimate.
- **Acyclic:** The edge in each tree connected between two events is unidirectional. Because of the temporal property of events, triggering relation graphs are free of cycles.

- **Sparsity:** A TRG is usually sparse, i.e., the number of a node's neighbors compared to the total numbers of nodes is small. The triggering relations occur in a short time range. Two events rarely form the triggering relationship, if they are far apart in terms of time.

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

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

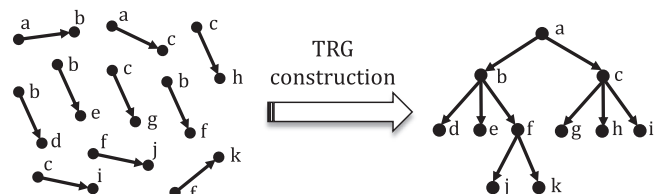


Fig. 2 – The triggering relation graph (TRG) on the right can be constructed from the pairwise triggering relations on the left.

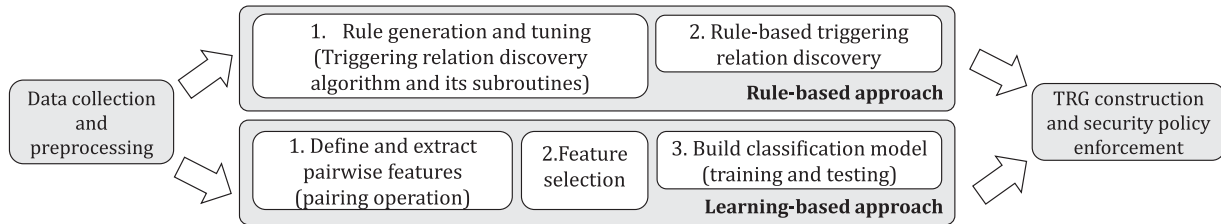


Fig. 3 – The system workflow containing major operations in the rule- and learning-based approaches.

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

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

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

2.2. Security applications of TRG

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

The definition of root triggers may vary. In a user-intention based security model, the user-input actions serve as the root triggers. The analysis can pinpoint network activities that are not intended by users. Blocking these outbound malware network activities effectively isolates the malware, including

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

We describe a scenario for using our tool to detect stealthy outbound malware activities on a host. DNS tunneling has been abused by botnets for command and control communications (Xu et al., 2013). These abnormal outbound DNS queries are automatically generated by malware on the host, typically with the botnet-related payload. These surreptitious DNS activities are difficult to detect, because of their format resemblance to regular DNS queries. Our analysis tool reasons about the legitimacy of observed DNS traffic on a possibly infected host. Legitimate DNS queries are usually issued by an application (e.g., browser) upon receiving certain user inputs (e.g., entering a URL into the address bar). The application then issues additional DNS or other requests (e.g., HTTP, FTP). Botnet DNS queries lack of any matching user triggers.

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

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

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

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

3. Rule-based discovery

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

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

3.1. Overview of rule-based discovery

Given a new request, the triggering relation discovery (TRD) algorithm aims at identifying its dependence with respect to

the known requests. We construct a forest structure to store the network requests and organize them according to the definition of TRG. The requests with known dependencies are chronologically organized into trees rooted by user events in the existing TRG. The root-triggers, thus, are also chronologically ordered.

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

Algorithm 1 Rule-based Triggering Relation Discovery (TRD)

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

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

```

1: if IsRoot( $p$ ) then
2:   create a new tree  $T_{m+1}$  with root  $p$ 
3:    $\mathbb{T} \leftarrow \mathbb{T} \cup \{T_{m+1}\}$ 
4:   return True
5: else
6:   for  $i \leftarrow m$  to 1 do
7:     continue if  $p.time - r_i.upTime > \tau$  or  $r_i.pid \neq p.pid$ 
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  $\{n.parent \rightarrow p\}$  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

```

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

3.2. Details of classification rules

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

The IsRoot procedure is used to test whether or not a request is the first one triggered by a user event. In the context of the browser, traffic-inducing user events may include typing into the address bar of the browser, clicking on a hyperlink or a bookmark, opening a new window or tab, and reloading a webpage. Therefore, the corresponding root request is the first immediate outgoing network request that has the identical process ID and with correlating content. The content may be the URL of the hyperlink for a mouse click, which needs to match the URL in the root request.

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

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

The IsSibling procedure is used for the nodes whose parent nodes cannot be directly determined. Therefore, identifying the sibling relation of a request helps establish a triggering relation by the transitivity. We are given two outbound HTTP requests p_a and p_b , where p_a 's parent node is known, p_b 's parent is unknown, and p_a proceeds p_b . To determine whether p_b is a sibling node of p_a , we define the rules as follows.

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

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

4. Learning-based discovery

We describe a machine learning approach to inferring triggering relations on network requests in this section. As compared in Table 1, the learning-based solution provides a more adaptive solution than the heuristic rules. We introduce a scalable feature extraction method referred to as *pairing*. This operation converts individual network events into event pairs with comparable pairwise attributes. We then show how binary classification algorithms can be used for the triggering relation discovery.

4.1. Overview of learning approach

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

- Pairing is a new operation that we design for extracting pairwise comparison results (i.e., features) of events' attributes. Its inputs are two events $e = (A_1, \dots, A_m)$ and $e' = (A'_1, \dots, A'_m)$. The output is the event pair (e, e') with m pairwise attribute values $(B_1, \dots, B_m)$, where a pairwise attribute $B_i (i \in [1, m])$ represents the comparison result of attributes A_i and A'_i . That is, $B_i = f_i(A_i, A'_i)$, where $f_i()$ is a comparison function for the type of the i -th attribute in the events. The comparison function $f_i()$ (e.g., IsEqual, IsGreaterThan, WithinThreshold, IsSubstring, etc.) is chosen based on the type of attribute. The feature construction can be extended to comparing different traffic types. Pairing is performed on every two events that may have the triggering relation. Moreover, we demonstrate an efficient pairing algorithm and advanced strategies to reduce the cost of pairing without compromising the analysis accuracy in Sections 4.3 and 4.4. The pairwise features are used as inputs to the subsequent learning algorithms.
- Data Labeling is the operation that produces the correct triggering relations for the event pairs in a (small) training dataset. A binary label (1 or 0) indicates the existence or non-existence of any triggering relation in an event pair, e.g., $\langle (e, e'), 1 \rangle$ represents that event e triggers e' . Data labeling is based on pairwise attributes (e.g., $B_1, \dots, B_m$) and may require manual efforts.
- Training is the operation that produces a machine learning model with labeled training data.
- Classification is the operation to use the trained machine learning model to predict triggering relations on new event pairs $\mathbb{P} = \{(e_i, e_j)\}$. E.g., the outputs of binary prediction results are in the form of $\{\langle (e_i, e_j), l_{ij} \rangle\}$, where the binary classification result $l_{ij} \in \{0, 1\}$ represents whether event e_i triggers e_j in $\mathbb{P}$.
- TRG Construction is the operation to build the complete triggering relation graph based on pairwise classification results. If event e_i triggers e_j in the event pairs $\mathbb{P}$, then e_i and e_j are connected by a directed edge in the TRG.

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

4.2. Pairing operation for feature extraction

The pairwise attributes are formed by aligning the same event features and comparing the relevant ones (e.g., the request type and the referrer type). Without loss of generality, we illustrate a basic pairing procedure with outbound HTTP network events as an example. The approach can be generalized to other event types. The pairing details are illustrated as follows.

- Numeric attributes (e.g., timestamps) are compared by computing their difference, e.g., the interval between the timestamps of two network events. That is, $B_i = A_i - A'_i$.
- A nominal attribute (e.g., file type, protocol type) categorizes the property of an event. Comparing nominal attributes usually involves string comparison, e.g., substring or equality tests.
- For the string type of attributes, we compute the similarity of the attribute values as the pairing attribute value. That is, $B_i = f_s(A_i, A'_i)$, where function f_s is a similarity measure, e.g., normalized edit distance. Take HTTP request as an example, the similarity index can be computed between two host domains, two referrer fields, and two request URLs.
- A composite attribute can be converted to primitive types, e.g., a destination address containing four octets for the IP

address and an integer for the port. Therefore, the comparison of two composite attribute values is made by comparing the sub-attribute values separately.

4.3. Efficient pairing algorithm

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

We describe the efficient pairing 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. It is confirmed from our experiments that the efficient pairing algorithm significantly reduces the size of pairwise data, without scarifying the detection accuracy.

Algorithm 2 Efficient Pairing Algorithm (EPA)

Input: A list of chronological sorted events, $\mathbb{L} = \{e_i\}$.

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

```

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:         $\mathbb{P} \leftarrow \mathbb{P} \cup \text{Pairing}(e_i, e_j)$ 
14:      end if
15:    end for
16:    calculate the expire time and update  $D[d]$ 
17:    add  $e_j$  in  $D[d]$ 
18:  else
19:    add new entry  $(d, \{e_j\})$  in  $D$ 
20:  end if
21: end for
22: return  $\mathbb{P}$ 

```

Table 2 – Semantics of values in a cost matrix.

		Classified as	
		W/O TR	With TR
Ground truth	W/O TR	TN: no penalty.	FP: penalty for finding triggering relations in non-related pairs. TP: no penalty.
	With TR	FN: penalty for failure to discover triggering relations.	
Note: TR stands for triggering relation.			

4.4. Efficiency pairing using parallel computing

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

4.4.1. Divide-and-conquer

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

4.4.2. MapReduce

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

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

4.5. Feature selection and cost matrix

We use two different feature selection algorithms, namely *Information Gain* and *Gain Ratio*. Once a set of features is chosen, we train and classify the data using three common supervised machine-learning classifiers – Naive Bayes, a Bayesian network (John and Langley, 1995), and a support vector machine (SVM) (Cortes and Vapnik, 1995).

Due to the sparsity of triggering relations existing in network traffic, we leverage the customized cost matrices (Elkan, 2001) to penalize missed relations during the training. In cost-sensitive classifiers, the cost matrix can be defined to weigh the false positive (FP) and false negative (FN) differently. A false negative refers to the failure to discover a triggering relation. A false positive means finding triggering relation in a non-related pair.

Shown in Table 2, the cost matrix used in our model is labeled by two categories: *with triggering relation* and *without*

triggering relation. The values in the matrix are the weights for penalizing classification mistakes. We set positive values in the cells for FNs and FPs. The cost-sensitive classification takes a cost matrix as an input. The trained model aims at minimizing the total penalty in imbalanced datasets. For simplicity, we show the values and omit the labels of the cost matrix. For example, $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ is a cost matrix that has no bias on FPs and FNs; $\begin{bmatrix} 0 & 1 \\ 10 & 0 \end{bmatrix}$ penalizes the FNs 10 times more than FPs for a classifier. In Section 6, we thoroughly evaluate how cost matrices improve our analysis accuracy.

5. Root-trigger security policy

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

The security policy defines the legitimate and abnormal events, which can be used to analyze the TRG and make security decisions. A specific root-trigger security policy is based on the *user intention* (Zhang et al., 2012), where a valid root trigger should be related to a user activity (e.g., a function call to retrieve user inputs, mouse clicks, or keyboard inputs). Other definitions for valid root triggers may be made according to the specific applications.

Under the *root-trigger* security policy, one determines the legitimacy of a network event e based on the legitimacy of e 's root trigger, i.e., whether or not e has a legitimate root trigger. Anomalous events are those without a valid root trigger, namely the *vagabonds*. These events may be due to malware activities or server misconfiguration.

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

The inputs of the root finding algorithm are an event e_k and a set $\mathbb{P}^*$ containing all the pairwise triggering relations $\{(e_i \rightarrow e_j)\}$. The output is a set $\mathbb{R}$ that includes all the roots of e_k . To compute the transitive reduction, we opt for a queue Q to perform the breadth-first traversal of TRG. In each iteration, we obtain the parent(s) $\mathbb{T}$ of a dequeued event n . For each event e in the set $\mathbb{T}$, we add e to the return set $\mathbb{R}$ if it is a root-type event. Otherwise (i.e., e is an intermediate node on the path from e_k 's root to e_k), the algorithm enqueues e onto Q for further iteration. This analysis returns root triggers for the network requests. Network requests without valid root triggers, namely the *vagabonds*, are flagged and alerted to the administrator for further inspection.

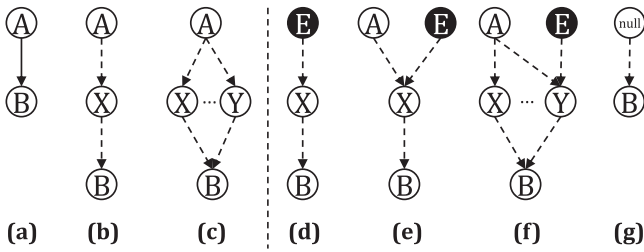


Fig. 4 – The illustration of various cases where B's predicted root trigger is correct (a–c) or wrong (d–g) on the triggering relation graph constructed from pairwise triggering relations. Let the ground truth of B's root trigger be A. Case (a) is where B's parent is also B's root. Cases (b) and (c)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

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

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

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

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

6. Evaluations and results

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

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

- How efficient is the learning-based approach? What is the most time-consuming operation in learning-based approach? (Section 6.4)

6.1. Experimental overview

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

6.1.1. Datasets

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

- Dataset I (HTTP). We collected the user events and outbound HTTP traffic in a user study with 20 participants. Each participant was asked actively surf the web for 30 minutes on a computer equipped with our data collection program.
- Dataset II (DNS and HTTP). We used tcpdump to continuously collect the outbound DNS queries and HTTP requests on an active user's workstation for 19 days. We collected types A/AAAA DNS queries and the outbound HTTP requests that contain valid GET, HEAD, or POST information in the headers.
- Dataset III (UDP and TCP). The dataset includes a DDoS attack scenario. The attackers first performed IP sweep and probed the hosts that run the Sadmin service. Then, three hosts got infected and were installed malicious scripts. Last, attackers launched a DDoS attack from the victim machines. In this experiment, we focus on our detection to the individual host and verify whether our method could be used for identifying the break-ins on each host.
- Dataset IV (server TCP traffic). We collected TCP packets on an active Linux server in a research lab. The inbound and outbound TCP packet headers were collected for 42 days using tcpdump.

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

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

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

¹Dataset III contains 33 different protocols, including TCP, UDP, DNS, and etc.

² η is the reduction percentage after using our efficient pairing algorithm (EPA).

³TR 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).

⁴RT labeling describes the methods to label root-triggers (RT), e.g., Tlogger and the heuristic rules mentioned in Section 5.

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

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

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

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

6.1.1.2. Pairwise feature extraction on network data

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

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

For application layer protocols, besides the aforementioned ones, we extract the pairwise features that have semantic meanings, e.g., the similarity between two request URLs, two referrers, and a domain and a DNS query. This type of features includes human-readable languages and can be analyzed using string comparison and advanced natural language processing techniques. In addition, the aggregate information over time is particularly useful to decide the triggering relation-

ship, when there are some missing or incomplete values in the packet. All pairwise features are finally chosen by two feature ranking algorithms (InfoGain and GainRatio). In our evaluation, we select the most contributive features with the cut-off value 0.01.

6.1.1.3. Classification setup of learning-based method

Three common classification techniques are compared: naive Bayes classifier, a Bayesian network, and a support vector machine (SVM).¹ Due to the sparsity of triggering relations in network traffic, we define a cost matrix that penalizes classifying false negatives more than classifying the false positives. Classification and TRG construction operations are implemented in Java using the Weka library. We perform both 10-fold cross validation and train-n-test types of evaluation. The two evaluation methodologies yield similar classification results. We report the train-n-test results, unless otherwise specified.

6.1.1.4. Accuracy and security metrics

- The conventional *precision and recall* measures (Baeza-Yates et al., 1999) evaluate the classification accuracy of the positives (i.e., the existence of triggering relations). In the equations below, TP, FP, and FN stand for true positives, false positives, and false negatives, respectively.

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

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

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

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

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

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

6.2. Evaluation of our learning-based approach

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

6.2.1. Evaluation on dataset I

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

We vary the cost matrices and compute the pairwise accuracy rates of the three classifiers. The results are shown in Fig. 5a. The accuracy rate is consistently high for naive Bayes classifier, despite the changes of cost matrices. Bayesian network and SVM respond differently to the changes of penalty values in cost matrices. In Table 5, we report the accuracy results under the cost matrix of $\begin{bmatrix} 0, & 1 \\ 10, & 0 \end{bmatrix}$. This matrix gives 10 units of penalty to a false negative and 1 unit of penalty to a false positive for the pairwise classification.

6.2.1.2. Correctness of root triggers. The purpose of this analysis is to identify the reasons for wrong predictions of root triggers. Running the root finding algorithm (in Section 5) on the predicted triggering relations, we obtain the root-triggers of all events and compare them with the ground truth. Fig. 5b shows the correctness of root-trigger analysis under different settings (classifier and cost matrices). The naive Bayes and Bayesian network classifiers yield nearly 100% accuracy of finding the root triggers, both of which are not very sensitive to the cost matrices. In contrast, the accuracy of SVM increases significantly with increased false negative penalty in

the cost matrix. In Table 6, we summarize the results of root trigger correctness for dataset I. Our prediction of events' root triggers is accurate. It has a very small error rate, as low as 0.06%. These errors in finding root triggers generate false alerts. Wrong root triggers are mostly because of missing attributes in the original data or late-arriving requests. We further analyze false alerts later.

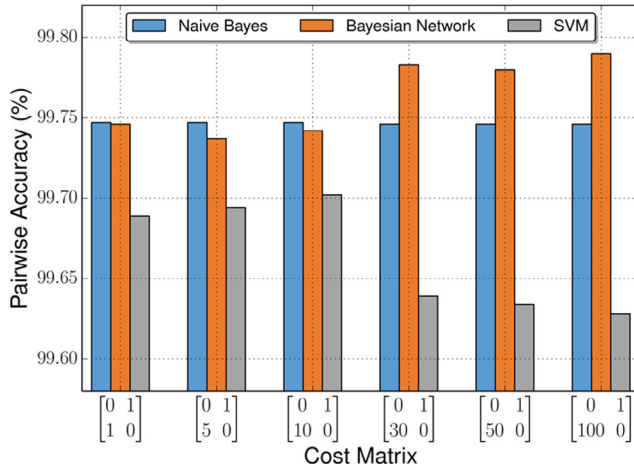
6.2.2. Detection of malicious traffic in dataset I

As defined in Section 5, vagabond events are those that do not have any valid user events as their root triggers. There are total 1.2% vagabond HTTP requests in dataset I. Some of them are malicious traffic to known blacklisted websites. Our analysis finds in dataset I that among these vagabond events, there are 169 suspicious requests sent to 36 distinct domains. Manual inspection reveals that these requests are to tracking sites, malware-hosting or blacklisted sites, and aggressive adware. They are partly due to users visiting compromised websites. For example, some requests track the user's cookies and send back to remote hosts with known blacklisted sites (e.g., 2o7.net, imrworldwide.com). We analyze the geographic locations of the malicious servers based on their IP addresses. All of them are located in the US, except one IP located in Netherlands.

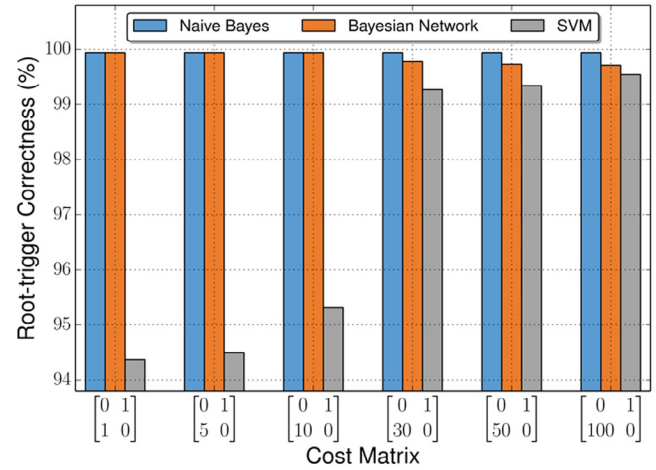
In our model, some vagabond requests are false alerts (i.e., requests without proper triggers), but are legitimate/benign. False alerts in dataset I are due to four main reasons:

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

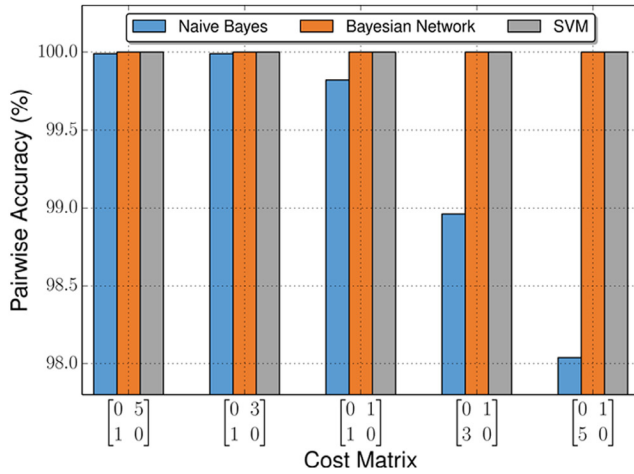
Reducing false alerts can be achieved through more sophisticated inference methods under incomplete information, which will be investigated in our future work.



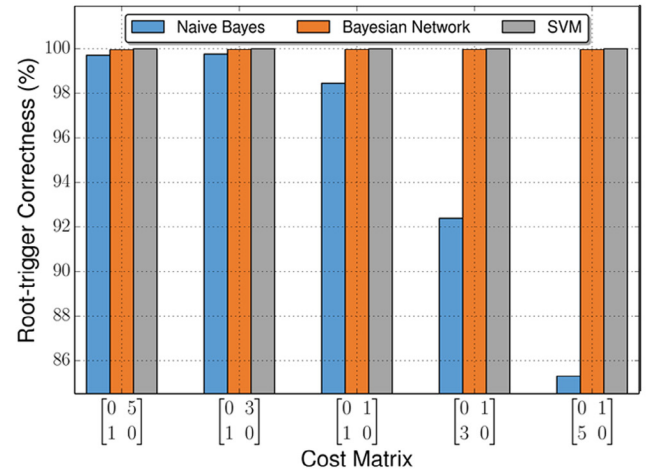
(a)



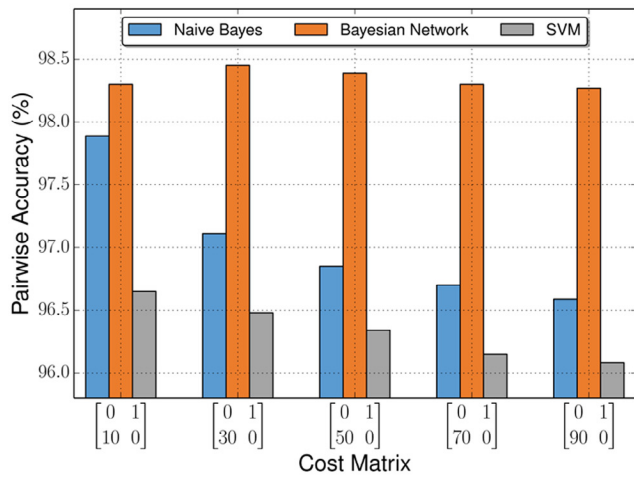
(b)



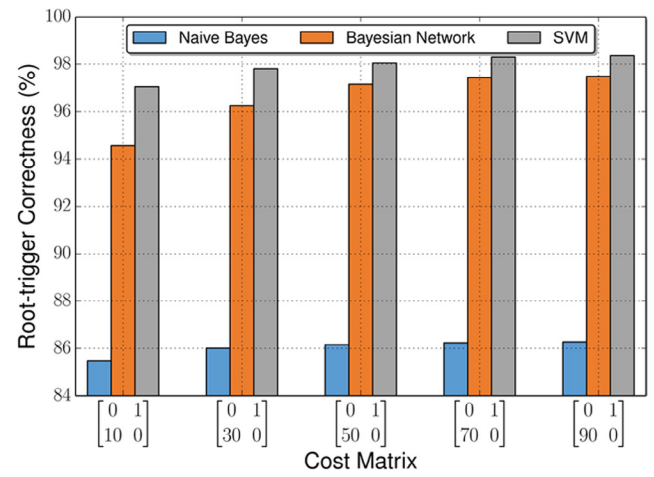
(c)



(d)



(e)



(f)

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

Table 6 – Correctness of root triggers in Dataset I.

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

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

6.2.3. Evaluation on dataset II

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

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

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

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

6.2.4. Evaluation on dataset III

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

6.2.4.1. Pairwise classification accuracy. Results in Table 5 show a high prediction accuracy rate for pairwise triggering relations. The precision results are all above 0.99, while the recalls

vary. By investigating the dataset, we found that the major reason of false positives is due to the change of protocols in one TCP session. Packets that have triggering relations with others usually share the same network protocol. However, in some rare cases, a packet using protocol A may trigger the another one using protocol B, especially when protocol A is a general one (e.g., TCP) and protocol B is a specific one (e.g., TELNET, SMTP and RSH protocols). A possible solution is to use features (in a vector) to characterize the change of protocols in a finer granularity, so that the classifiers can learn the patterns.

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

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

Table 7 – Correctness of root triggers on Dataset II.

	Naive Bayes	Bayesian network	SVM
Cost matrix	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Correct (case a–c)	98.44%	100.00%	100.00%
Wrong (case d–f)	1.37%	0.00%	0.00%
Wrong (case g)	0.19%	0.00%	0.00%

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

We further introduce a new policy for detecting the malicious root-triggers, after finding the root-triggers using Algorithm 3. The policy is a heuristic one and based on our observations on dataset III. We highlight the high-risk protocols based on their frequencies and patterns of occurrence. In this policy, we found that network packets via Portmap protocol (port = 111) immediately followed (<0.1 s) by the Sadmin protocol (port = 32773) are malicious, which can be regarded as the

period of infection. Besides, we discovered that the Portmap and Sadmin protocols occurred three times in a short period (<10 s) and the packets from an external IP (202.77.162.213) after each period are related to malicious activities (e.g., remote buffer overflow for rooting shell). This experiment suggests that data-specific security policies are more effective than general purpose ones. We show our model could be integrated with refined policies to detect malicious activities in the early age.

Algorithm 3 Root Finding Algorithm (RFA)

Input: An event e_k and $\mathbb{P}^* = \{(e_i \rightarrow e_j)\}$.

Output: A set $\mathbb{R}$, where each in $\mathbb{R}$ is a root of e_k .

```

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

```

6.2.5. Evaluation on dataset IV

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

6.3. Evaluation of stealthy malware activities

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

6.3.1. Malicious browser extension

We write a proof-of-concept malicious Firefox extension, which is a piece of password-stealing spyware. The malware sends the username and password when a user clicks on the Submit button in the browser. This spyware is similar to the existing spyware such as FormSpy and FFsniff. A victim user clicks the Submit to log on to various email services and Internet forums. The spyware requests, which contain the username and password in the HTTP request (/query?id=user_id&ps=password), are sent to its destination host. With our triggering relation model and root-trigger policy, all malicious HTTP requests are detected. However, the default Windows Firewall does not alert the data leaks.

6.3.2. Data exfiltrating malware

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

6.3.3. Detection of real-world malware

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

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

² Most malware samples are found at packetstormsecurity.com.

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

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

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

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

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

6.3.4. DNS bot detection

Botnet command and control channel using DNS tunneling (DNScat) is extremely stealthy and difficult to detect (Xu et al., 2013). We write a proof-of-concept bot that communicates with its bot master by tunneling command and control messages in DNS traffic. The bot generates carefully crafted outbound DNS queries whose payload contains encoded data, e.g., d1js21szq85hyn.cloudfront.net. We overlay the bot queries with a 2-hour DNS-HTTP traffic data (from dataset II), and then analyzed using our learning-based solution. Our evaluation confirms that our method successfully recognizes all the bot DNS queries as anomalies. These DNS queries do not have valid user-event root triggers.

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

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

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

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

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

6.4.1. Performance of rule- and learning-based approaches

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

6.4.2. Performance breakdown of learning-based approaches

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

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

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

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

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

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

6.5. Summary

We summarize our findings below.

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

7. Related work

There exist solutions for discovering application or service dependencies for management and reliability purposes (Bahl et al.,

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

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

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

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

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

Our triggering relation discovery problem may bear superficial similarities to the link prediction problem in the context of mining social network data (Backstrom and Leskovec, 2011; Getoor and Diehl, 2005; Kahanda and Neville, 2009; Liben-Nowell and Kleinberg, 2007). Getoor and Diehl (2005) surveyed the link mining problem and pointed out the sparsity in linked data. Liben-Nowell and Kleinberg (2007) formalized the link prediction problem and surveyed an array of methods on measuring the proximity of nodes in a network. Follow-up research applied advanced machine learning methods to social network data. These advanced methods include logistic regression, decision tree, and naive Bayesian (Kahanda and Neville, 2009) as well as supervised random walks (Backstrom and Leskovec, 2011). Besides the obvious semantic differences in the two problems, our work differs from social network link prediction.

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

8. Conclusions

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

REFERENCES

- Babić D, Reynaud D, Song D. Malware analysis with tree automata inference. In *Computer Aided Verification*. 2011. p. 116–31, 20.
- Backstrom L, Leskovec J. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*. 2011. p. 635–44, 20, 21.
- Baeza-Yates R, Ribeiro-Neto B, et al. *Modern information retrieval*, vol. 463. New York: ACM press; 1999. p. 13.
- Bahl PV, Chandra R, Greenberg A, Kandula S, Maltz D, Zhang M. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proceedings of ACM SIGCOMM*, August 2007. 19.
- Bilge L, Kirda E, Kruegel C, Balduzzi M. EXPOSURE: finding malicious domains using passive DNS analysis. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*. February 2011. 20.
- Borders K, Prakash A. Web Tap: detecting covert web traffic. In *Proceedings of the 11th ACM Conference on Computer and Communication Security*. 2004. p. 110–20, 20.
- Chen X, Zhang M, Mao ZM, Bahl P. Automating network application dependency discovery: experiences, limitations, and new solutions. In *Proceedings of OSDI*. 2008. p. 117–30, USENIX Association. 19.
- Cortes C, Vapnik V. Support-vector networks. *Mach Learn* 1995;20(3):273–97. 10.
- Cova M, Kruegel C, Vigna G. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In *Proceedings of 19th International World Wide Web Conference*, 2010. 2.
- Cui W, Katz YH, Tan WT. BINDER: an extrusion-based break-in detector for personal computers. In *Proceedings: USENIX Annual Technical Conference*. 2005. p. 4, 2, 20.
- DNScat. A tool to tunnel traffic through DNS servers. <http://tadek.pietraszek.org/projects/DNScat/>. 2004. 18.
- Elkan C. The foundations of cost-sensitive learning. In *International Joint Conference on Artificial Intelligence*, volume 17. 2001. p. 973–8, 10.
- Erman J, Mahanti A, Aritt MF, Cohen I, Williamson CL. Semi-supervised network traffic classification. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS 2007, San Diego, California, USA, June 12–16, 2007. 2007. p. 369–70, 20.
- Getoor L, Diehl CP. Link mining: a survey. *SIGKDD Explor News* 2005;7(2):3–12. 20.

- Green TM, Ribarsky W, Fisher B. Visual analytics for complex concepts using a human cognition model. In *Visual Analytics Science and Technology*, 2008. VAST'08. IEEE Symposium on. 2008. p. 91–8, 2.
- Gu G, Perdisci R, Zhang J, Lee W. BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium*. 2008. 2.
- Gummadi R, Balakrishnan H, Maniatis P, Ratnasamy S. Not-a-bot: improving service availability in the face of botnet attacks. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NDSI)*. 2009. 20.
- John G, Langley P. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*. 1995. p. 338–45, 10.
- Kahanda I, Neville J. Using transactional information to predict link strength in online social networks. In *Proceedings of the Third International Conference on Weblogs and Social Media (ICWSM)*. 2009. 20, 21.
- Kandula S, Chandra R, Katabi D. What's going on? Learning communication rules in edge networks. In *Proceedings of ACM SIGCOMM*. August 2008. 19.
- Keller A, Blumenthal U, Kar G. Classification and computation of dependencies for distributed management. In *Proceedings of International Symposium on Computers and Communications*. 2000. p. 78–83, 19.
- King ST, Mao ZM, Lucchetti DG, Chen PM. Enriching intrusion alerts through multi-host causality. In *Proceedings of Network and Distributed System Security (NDSS)*. 2005. 2, 19.
- Kolbitsch C, Comporetti PM, Kruegel C, Kirda E, Zhou X, Wang X. Effective and efficient malware detection at the end host. In *USENIX Security Symposium*. 2009. p. 351–66, 20.
- Lee W, Stolfo SJ, Mok KW. A data mining framework for building intrusion detection models. In *Proceedings of IEEE Symposium on Security and Privacy*. 1999. p. 120–32, 20.
- Li Z, Zhang M, Zhu Z, Chen Y, Greenberg AG, Wang Y-M. WebProphet: automating performance prediction for web services. In *NSDI*, volume 10. 2010. 20.
- Liben-Nowell D, Kleinberg J. The link-prediction problem for social networks. *J Am Soc Inf Sci Tec* 2007;58(7):1019–31. 20, 21.
- MIT Lincoln Laboratory. Darpa intrusion detection evaluation. https://www.ll.mit.edu/ideval/data/2000/LLS_DDOS_1.0.html. 2000. 12.
- Moore AW, Zuev D. Internet traffic classification using Bayesian analysis techniques. In *Proceedings of the International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS 2005*, June 6–10, 2005, Banff, Alberta, Canada. 2005. p. 50–60, 20.
- Moshchuk A, Bragin T, Deville D. SpyProxy: execution-based detection of malicious web content. In *Proceedings of the 16th USENIX Security Symposium*. 2007. 20.
- Natarajan A, Ning P, Liu Y, Jajodia S, Hutchinson SE. NSDMiner: automated discovery of network service dependencies. In *INFOCOM*. 2012. p. 2507–15, 2, 5, 19.
- Neasbitt C, Perdisci R, Li K, Nelms T. ClickMiner: towards forensic reconstruction of user-browser interactions from network traces. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 2014. p. 1244–55, 20.
- Nguyen TTT, Armitage GJ. A survey of techniques for Internet traffic classification using machine learning. *IEEE Commun Surv Tut* 2008;10(1–4):56–76. 20.
- Tlogger. An Firefox extension. <http://dubroy.com/tlogger/>. 2009. 11, 24.
- Williams N, Zander S, Armitage G. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *SIGCOMM Comput Commun Rev* 2006;36(5):5–16. 20.
- Xie G, Iliofotou M, Karagiannis T, Faloutsos M, Jin Y. ReSurf: reconstructing web-surfing activity from network traffic. In *IFIP Networking Conference*, 2013, Brooklyn, New York, USA, 22–24 May, 2013. 2013. p. 1–9, 19.
- Xie P, Li JH, Ou X, Liu P, Levy R. Using Bayesian networks for cyber security analysis. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*. 2010. p. 211–20, 20.
- Xu K, Butler P, Saha S, Yao D. DNS for massive-scale command and control. *IEEE Trans Dependable Sec Comput* 2013;10(3):143–53. 5, 12, 18.
- Zand A, Vigna G, Kemmerer R., Kruegel C. Rippler: delay injection for service dependency detection. In *INFOCOM'14*. 2014. p. 2157–65. 2, 5, 19.
- Zhang H, Banick W, Yao D, Ramakrishnan N. User intention-based traffic dependence analysis for anomaly detection. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*. 2012. p. 104–12, 1, 3, 6, 10.
- Zhang H, Yao D, Ramakrishnan N. Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery. In *Proceedings of the 9th ACM Symposium on Information, Computer, and Communication Security (ASIACCS'14)*, June 2014. 2.
- Zhang H, Sun M, Yao D, North C. Visualizing traffic causality for analyzing network anomalies. In *Proceedings of International Workshop on Security and Privacy Analytics (IWSPA)*. 2015. p. 37–42, 1, 3.
- Zomlot L, Chandran S, Caragea D, Ou X. Aiding intrusion analysis using machine learning. In *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*, volume 2. 2013. p. 40–7, 20.
- TCPFLOW 1.3. <https://github.com/simong/tcpflow>. 24.
- Panda Security Report. <http://press.pandasecurity.com/press-room/reports/>. 2015. 2.
- Pony botnet. Botnet pony 1.9 malware. <http://laboratoriomalware.blogspot.com/2013/01/botnet-pony-19-malware.html>. 2013. 17.
- Hao Zhang received his Ph.D. degree in Computer Science from Virginia Tech in 2015. He was a member of the Human-Centric Security Laboratory directed by Professor Danfeng Yao. He received his M.S. degree in Computer Science from Villanova University, PA in 2010. He holds a U.S. patent on his network anomaly detection technology. His current research interest is on designing machine learning methods for network and mobile security.
- Danfeng (Daphne) Yao is an associate professor and L-3 Faculty Fellow in the Department of Computer Science at Virginia Tech, Blacksburg. She received her Computer Science Ph.D. degree from Brown University in 2007. She received the NSF CAREER Award in 2010 for her work on human-behavior driven malware detection, and most recently ARO Young Investigator Award for her semantic reasoning for mission-oriented security work in 2014. She received the Outstanding New Assistant Professor Award from Virginia Tech College of Engineering in 2012. Dr. Yao has several Best Paper Awards (e.g., ICICS '06, CollaborateCom '09, and ICNP '12) and Best Poster Awards (e.g., ACM CODASPY '15). She was given the Award for Technological Innovation from Brown University in 2006. She held a U.S. patent for her anomaly detection technologies. Dr. Yao is an associate editor of *IEEE Transactions on Dependable and Secure Computing (TDSC)*. She serves as PC members in numerous computer security conferences, including ACM CCS. She has over 65 peer-reviewed publications in major security and privacy conferences and journals.

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

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

Causality-based Sensemaking of Network Traffic for Android Application Security*

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

ABSTRACT

Malicious Android applications pose serious threats to mobile security. They threaten the data confidentiality and system integrity on Android devices. Monitoring runtime activities serves as an important technique for analyzing dynamic app behaviors. We design a triggering relation model for dynamically analyzing network traffic on Android devices. Our model enables one to infer the dependency of outbound network requests from the device. We describe a new machine learning approach for discovering the dependency of network requests. These request-level dependence relations are used to detect stealthy malware activities. Malicious requests are identified due to the lack of dependency with legitimate triggers. 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 malicious apps.

1. INTRODUCTION

Similar to PC malware, researchers found that malicious mobile applications usually fetch and run code on-the-fly without the user's knowledge [49]. Their purposes are often to stealthily collect and exfiltrate sensitive information.

Static analysis solutions typically inspect the source code, binaries or call sequences for detecting anomalies. For example, Drebin [4] extracts features including APIs, permissions, and app components to characterize and classify apps. SCSDroid [20] identifies malicious apps by extracting subsequences of system calls. However, dynamic code loading, Java reflection-based method invocation, data encryption, and self-verification of signatures are commonly seen in the malware code [11, 38]. These types of code obfuscation 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,

*This work has been supported in part by NSF grant CAREER CNS-0953638 and ARO YIP W911NF-14-1-0535.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

AISeC'16, October 28 2016, Vienna, Austria

© 2016 ACM. ISBN 978-1-4503-4573-6/16/10...\$15.00

DOI: <http://dx.doi.org/10.1145/2996758.2996760>

TaintDroid [10] 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). Our solution provides the logic insights between the user's interaction and malicious traffic. 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.

In this paper, we aim at identifying malicious apps by analyzing their dynamic behaviors, specifically the network traffic generated by apps. We profile the traffic pattern of benign apps to detect malicious network requests and enhance the sensemaking process.

Sensemaking is an analysis process including the tasks of investigating the complex data, exploring their connections, and gaining insights [13]. In this paper, we propose a learning-based approach to discover *dependencies* of network requests and thus detect malicious ones. The problem of dependence discovery on network events has been introduced in [44]. The dependence of network requests is defined as the *triggering relationship* (TR). The triggering relation between two network requests r_i and r_j exists if r_j cannot be issued unless r_i is sent out first (denoted by $r_i \rightarrow r_j$). The triggering relations on app-generated requests capture how one network event is related to other events. They model the underlying logical and semantic relations among network events, which is useful for human experts' cognition, reasoning, and decision making in cyber security [12, 43].

The temporal and triggering relationship can be represented in a directed graph that is referred to as *triggering relation graph* (TRG). We refer the first request that triggers others as the *root-trigger* request. It may be caused by legitimate user activities or generated by benign apps. 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., web-browsing, downloading a song). The legitimacy of the requests relies on the legitimacy of their root-triggers. Therefore, dependence structure of network requests offers important insights for identifying network anomalies, and thus supporting the sensemaking process.

Our Android traffic causal analysis targets at the stealthy network activities via HTTP. HTTP is the protocol chosen by most app developers to implement communication with remote servers [37] and is hardly blocked by anti-virus tools. Repackaged apps and drive-by download attacks are the common initial infection vectors [49]. 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.

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 (TRG). The graph depicts the causality of requests and allows one to quickly identify the root triggers of observed events. Our solution classifies 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 in Android network traffic. It allows us to detect the stealthy malware activities that may not be detected by existing traffic analysis methods (e.g., [9, 37]).

Dependence analysis was proposed for detecting anomalies on hosts in [44]. Our work systematically differs from theirs in the problem scope and implementations. First, we extend the triggering relation discovery problem for all types of apps on mobiles. The definition of triggering relationship is studied using the delay injection approach. In comparison, theirs only handles the browser-generated requests by using the referrer-based heuristic. Second, their solution requires a predefined whitelist to filter out the non-user triggered benign requests, while we leverage the two-stage learning-based approach to classify the benign requests.

Our contributions are summarized as follows.

- We utilize the triggering relation model to formalize the dependency of mobile network requests and traffic-generating user inputs. The discovery of triggering relations on pairwise network events enables us to construct a triggering relation graph. We present a two-stage learning-based solution and use it to classify the abnormal requests.
- We conduct our experiments on 14GB network and system data. Our results show that the triggering relations of network traffic on Android can be inferred at the accuracy of 98.2%. Without knowing any *priori* knowledge, our solution enables the detection of malicious requests sent from the newly released apps. Results confirm that we can detect 99.1% malicious requests from all malicious apps.

The significance of our work is to provide insights of the traffic dependency for Android security and demonstrate the use of structural and semantic information in reasoning about network behaviors and detecting stealthy anomalies.

Dependency analysis is a promising security approach that is capable of describing how events are related to each other and further identify the anomalies isolating from others. The pairwise relationship is designed to compare the features of two events, which are also applicable to system calls, services in a network, and events in Internet of Things (IoT). The key challenges of obtaining pairwise relationship are *i)* the features need to sufficiently characterize the logic relation between events, and *ii)* the classification needs to be scalable for voluminous data. In this paper, we present our prototype to address the problem in the context of Android network traffic. We show its feasibility in detecting zero-day network anomalies based on the dependencies of network requests.

2. MOTIVATION AND OVERVIEW

We discuss the challenges of extracting network traffic dependency in Android and the security applications of our analysis model. We also give an overview of our solution.

2.1 New Challenges of 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 [36, 45] or the instrumented browser [25, 26] 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. Properly handling the notifications is crucial in achieving a high accuracy and low false positives. The tradition whitelisting created by human experts is no longer adequate. Hence, to automatically generate whitelisting using learning-based methods is desirable.

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 classify 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 which are further used to identify malicious requests and aid the sensemaking process. The whitelisting generated from machine learning outcomes saves human efforts and improves the detection accuracy.

2.2 Security Applications of Our Detection

Our model detects the malicious network behaviors by discovering the dependency of network packets on Android devices. The behaviors of malware include the unauthorized network activities, e.g., stealthy outbound requests without user’s awareness, piggyback requests containing 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. Malicious apps may not immediately trigger its malicious behaviors, because the timing or logic bombs (e.g., debugger-checks, rooting-checks) could delay the triggering. We list some examples of our security applications as follows.

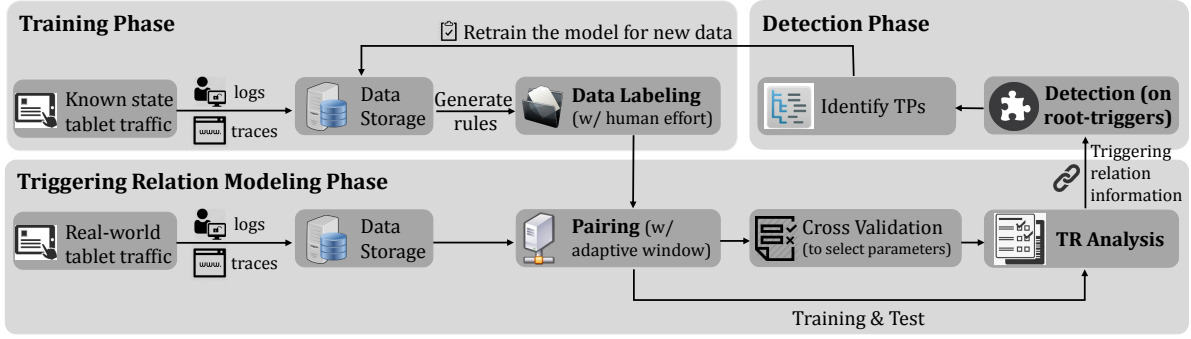


Figure 1: The workflow of our triggering relation discovery based malware detection.

- **Repackaged apps** refer to the malicious apps created by repackaging the existing benign ones [49]. 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. In our context, drive-by downloading refers to the unintentional malicious download. For example, it may lure users and stealthily install a new malicious app or create a shortcut icon to some malicious or advertisement sites, e.g., *com.Punda.Free*.
- **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.

The benign apps sometimes send out outbound requests without user’s awareness. As an open problem, these “gray” requests are difficult to be classified, because they may be sent with user’s consents. Our model labels these requests as *suspicious*.

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.

2.3 Overview of Our Approach

A 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 solved by inferring the triggering relations of pairs of events, which is defined as the *pairwise triggering relation*. Given a sequence of network requests $\mathbb{R} = \{r_1, r_2, \dots, r_n\}$, 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 [16, 17, 44]. In this work, we further advance it in the Android context by improving the pairing efficiency and using the multinomial classification.

Utilizing predicted results from the classification, we build the TRG $\mathbb{G} = \{T_1, T_2, \dots, T_m\}$, each T_i is a tree rooted at request r_{t_i} . We classify the root-triggers $\mathbb{R}_T = \{r_{t_i} \mid 1 \leq i \leq m\}$, 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 auto-generated notifications/updates from benign apps.

We show an overview of the workflow in Figure 1. Our approach requires the data labeling (in training phase) based on the requests $\mathbb{R}$, details of which are given in §5.

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.

- Pairing takes the $\mathbb{R}$ as an input and outputs a set of paired requests $\mathbb{P} = \{P(r_i, r_j)\}$. (§3.1)
- TR analysis takes the predicted pairs $\mathbb{P}$ as inputs and outputs a constructed TRG $\mathbb{G}$. (§3.2)
- Detection takes a TRG $\mathbb{G}$ and its root-trigger set $\mathbb{R}_T$ as inputs. It extracts features of $\mathbb{R}_T$ and builds classifications to predict the legitimacy of root-triggers. (§4)

3. TRIGGERING RELATION MODELING

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

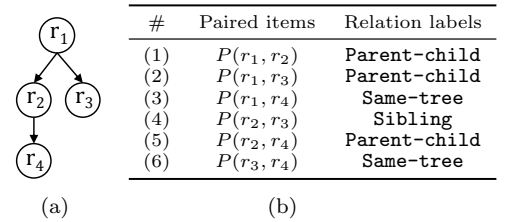


Figure 2: A sample tree (a) and its pairing results (b). In (a), nodes and edges denote the network requests and their triggering relations respectively.

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 a TRG. Figure 2 illustrates how we generate pairs from a sample tree based on quad-class labeling and how these relations determine the pattern of a tree in a TRG.

3.1 Pairwise Feature Extraction

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 $r_i.attr$, $attr \in \{\text{time}, \text{IP}, \text{port}, \text{host}, \dots, \text{URL}\}$. Then, the pairing operation is performed on every attribute of the requests. The pairwise comparison can be written as $P(r_i, r_j) = \{f_1(a_i.time, a_j.time), \dots, f_p(a_i.URL, a_j.URL)\}$, where f_k is the pairing function for the k -th attribute. Shown in Table 1, different types of attributes require different pairing strategies (f_k).

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

Table 1: Pairing functions used for different attribute types.

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 (threshold). Let $d_{(i,j)}$ be the time difference between a pair of requests r_i and r_j that have triggering relation. We obtain the 3-day moving average of $d_{(i,j)}$ as the μ and its corresponding σ^2 in the learning phase, and calculate the threshold using the *Chebyshev's inequality* ($\tau = \mu + 10 * \sigma$). 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.

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.

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 is 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 1 as a score of the pair (r_i, r_j) , which is the summation of the weights that indicate their triggering relationship.

$$w_{(r_i, r_j)} = \sum w \begin{cases} w_p, & \text{if } (r_i, r_j) \rightarrow \text{parent-child.} \\ w_s, & \text{if } (r_i, r_k) \rightarrow \text{parent-child} \\ & \text{and } (r_k, r_j) \rightarrow \text{sibling.} \\ w_t, & \text{if } (r_i, r_j) \rightarrow \text{same-tree.} \end{cases} \quad (1)$$

We show the procedure to calculate the weight for the classified pair (r_i, r_j) in Algorithm 1. The inputs of Algorithm 1 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 $\text{Parent}(r_i)$). The output of Algorithm 1 is a dictionary of keys $\mathbb{D}$, which contains the calculated weights of pairs that (may) have triggering relationship.

Algorithm 1 Triggering Relation Weights Calculation

Input: Requests $\mathbb{R} = \{r_1, r_2, \dots, r_n\}$, classified pairs $\mathbb{P} = \{(r_i, r_j) \rightarrow p_{ij}\} (1 \leq i < j \leq n)$, 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  $p_{ij} = \text{parent-child}$  relation then
4:     weight  $w_{(r_i, r_j)} += w_p$ , and update  $w_{(r_i, r_j)}$  in  $\mathbb{D}$ 
5:   else if  $p_{ij} = \text{sibling}$  relation then
6:      $\text{Parent}(r_i) \leftarrow r_i$ 's trigger(s) according to  $\mathbb{P}$ 
7:     for  $r_p$  in  $\text{Parent}(r_i)$  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  $p_{ij} = \text{same-tree}$  relation then
11:     $\text{Root}(r_i) \leftarrow r_i$ 's root request(s) according to  $\mathbb{P}$ 
12:    for each  $r_r$  in  $\text{Root}(r_i)$  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  $\mathbb{D}$ 
```

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 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 $r_{t_i}.time > r_{t_j}.time$, then $r_i \rightarrow r_k$. (The rule is in favor of the request on the most recent dependency tree.)
- Rule 2)* If $r_i, r_j \in T_i$, $r_i.level < r_j.level$, then $r_i \rightarrow r_k$. (The rule is in favor of the request on the upper level of the tree.)
- Rule 3)* If $r_i, r_j \in T_i$, $r_i.level = r_j.level$, $r_i.time > r_j.time$, then $r_i \rightarrow r_k$. (The rule is in favor of the most recent requests on the same level of the tree.)

We make these design choices based on the empirical study 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.

²Prim'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.

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

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

Table 2: The triggering relations are built from the predicted results. In predicted results, **p-c** and **sib** denote the **parent-child** and **sibling** relations respectively. $w_{(r_i, r_j)}$ shows the calculated weights using Algorithm 1. The tie-break column shows the rule to resolve ties. Note that the weight $w_p + w_s$ in (1) and (2) is due to the predicted result from (3).

We illustrate our TRG construction using an example in Table 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 of 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 1 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 $\mathbb{R}$.

Security discussion. The threshold τ in §3.1 determines the efficiency and accuracy in pairing operation. A large τ causes unnecessary pairs between two irrelevant requests, resulting in unbalanced datasets, which negatively impacts the inference accuracy. A small τ leads to insufficient pairs, resulting in a fragmented TRG. For example, synchronous requests (e.g., **AJAX**), retrieved much later than normal requests, could be regarded as *vagabond* requests, and causes false positives in detection. Therefore, the adaptive window size can improve the pairing accuracy by adjusting the τ according to the recent user’s behaviors.

The TRG construction relies on the weights calculation algorithm and tie-breaking rules. Malicious requests tend to hide themselves by building dependencies to benign requests. Wrong triggering relationship results in the wrong root-trigger, which causes the false positives of malware detection. Our algorithm and rules build correct TRs by exhausting all predicted weights and *reinforce* the sensemaking based on the parent-child and slinging relationships.

4. DETECTION ON ROOT-TRIGGERS

In the detection phase, we identify the root triggers $\mathbb{R}_T$ 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 ($\mathbb{U}_T$). However, there may exist requests that cannot be linked with any user’s activities, e.g., notification requests from benign apps ($\mathbb{A}_T$), malicious requests from the malware ($\mathbb{M}_T$). Therefore, $\mathbb{R}_T$ can be formalized as the disjoint sets of the three.

$$\mathbb{R}_T = \{\mathbb{U}_T \cup \mathbb{A}_T \cup \mathbb{M}_T\} \quad (2)$$

In our design, we use binary classifiers to detect the malicious requests ($\mathbb{M}_T$) in two steps: (1) In a clean/labeled

dataset, we distinguish the notifications or update requests ($\mathbb{A}_T$) from other benign ones ($\mathbb{U}_T$). A whitelisting can be generated from the learning outcomes ($\mathbb{A}_T$). (2) In a real-world dataset, we filter out the requests that belong to the whitelist, and then differentiate the malicious requests ($\mathbb{M}_T$) from the benign ones ($\mathbb{U}_T$).

We describe the features extraction for root-triggers in §4.1 and the root-trigger identification/labeling in §4.2.

4.1 Feature Extraction for Detection

For a given root-trigger $r_t \in \mathbb{R}_T$, we extract features as described in Table 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 $\mathbb{R}_T$.

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 $r \in \mathbb{U}_T$, $r_t.\text{time} - r.\text{time} \leq \tau^* \uparrow$, $f_u(a_r.\text{attr}, a_{r_t}.\text{attr}) \leq \tau_u \uparrow$
$\mathcal{F}_2$	$\mathcal{F}_2$ describes the number of similar requests in the previous non-user-triggered trees. $f(r_t) = r $, where $r \in \{\mathbb{A}_T \cup \mathbb{M}_T\}$, $r_t.\text{time} - r.\text{time} \leq \tau^* \uparrow$, $f_u(a_r.\text{attr}, a_{r_t}.\text{attr}) \leq \tau_u \uparrow$
$\mathcal{F}_3$	$\mathcal{F}_3$ describes the <i>statistics</i> from the previous user-triggered trees in $\mathbb{G}$. $f(r_t) = r $, where $r \in \mathbb{U}_T$, $r_t.\text{time} - r.\text{time} \leq \tau^* \uparrow$
$\mathcal{F}_4$	$\mathcal{F}_4$ describes the <i>temporal</i> 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.\text{time} - u_p.\text{time} $.

$\uparrow \tau^*$ 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.
 $\uparrow f_u$ is the function to compute the similarity between attributes. $a_r.\text{attr}$ denotes a request r ’s (semantic) features, e.g., **Host**, **Referrer**, **request URL** and **destination IP**.

Table 3: Features extracted for the detection operation.

In this operation, the dependency-based feature set $\mathcal{F}_D = \{\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4\}$ contain the rich semantic and structural information of $\mathbb{G}$ and aid the sensemaking in detecting anomalies. The design of $\mathcal{F}_D$ features is based on our empirical observations on malicious and benign requests. $\mathcal{F}_1$ features are extracted to characterize the frequently visit websites and some benign requests. Because these websites can be cached by apps to save the bandwidth, therefore, the newly generated requests may not contain referrers. $\mathcal{F}_2$ features are used to characterize the auto-generated benign requests. Because Android apps may pull notifications from the remote servers periodically, the IP, host and referrer information of the notifications have usually seen before. $\mathcal{F}_3$ and $\mathcal{F}_4$ features are designed to quantify the pattern of $\mathbb{U}_T$ 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 noticeable long interval.

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 [18] 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 the caller app name and its status information. 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.

- We identify the notification requests ($\mathbb{A}_T$) by parsing the **ActivityManager** logs from Logcat. The **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.
- 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 ($\mathbb{U}_T$) with the closest user's activity prior to it.

4.3 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 [37] 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.* As an anti-analysis technique, attackers may forge the user and system events so that the malicious requests have triggers. Advanced solutions (e.g., [2, 24, 28]) 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 the encrypted network activities 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 [29].

5. 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. Our approach is based on the timing perturbation. It 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.

We elaborate the approach to label the triggering relations (data labeling operation in Figure 1). This process is time-consuming and needs 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 §3 can be used for analyzing and testing on large-scale data.

Given two requests r_i and r_j , where r_i triggers r_j , the time difference $d_{(i,j)}$ can be decomposed into 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 causality for non-browser apps. 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 [42] operate at different levels of granularity: we detect dependencies at the level of outbound requests, while Rippler does so at the level of network services.

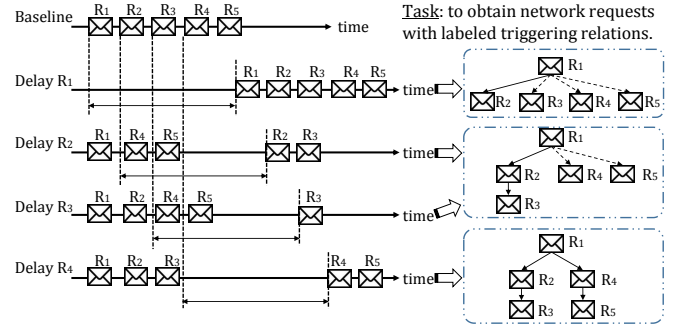


Figure 3: We label training data using the timing perturbation and incrementally build the TRG. Dashed lines represent the intermediate results and solid lines represent the finalized triggering relationship.

Our dependency inference consists of three steps. In step (i), we record $\mathbb{R} = \{r_1, r_2, \dots, r_n\}$, 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 3). 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 to figure out whether the j -th packet is affected by the i -th packet [30]. 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 dependency graph can be incrementally built based on the finding of dependency from temporal perturbations, as shown in Figure 3.

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.

6. 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 TRs? (§6.4, §6.6)
- How accurate is the classification for detecting the notifications and malicious requests? (§6.5, §6.7)
- Can it detect *new* malicious Android apps? (§6.7)
- Is our approach scalable for analyzing real-world data? What is the performance of our approach? (§6.8)

6.1 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 the device and install `PythonForAndroid` to run Python code. We use `tcpdump` to sniff HTTP requests whose headers contain valid GET or POST request. To log the user's events, we continuously monitor the files from `/data/input/`. 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 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.

6.2 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 [3]. 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. The selected popular apps cover a wide range of categories, including news, social network, shopping, games, entertainment, etc.

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, including 1.4 million user input records, 1.8 million system logs, and 565 thousand network events after consolidation.

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 auto-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 to detect malicious requests.

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 [44]. To label the dependency of network traffic that comes from non-browser apps, we first installed the apps in the Android emulator.³ 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. We run the timing perturbation approach and generate heuristic rules to label all the triggering relations. Examples of rules and learned patterns are listed in Table 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.

6.3 Accuracy and Security Metrics

We apply three classifiers⁴, namely random forest (R-F), C4.5 algorithm (C45), and logistic regression (LOG) in both triggering relation modeling and detection phases. The operations are evaluated using the following metrics.

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

$$PC = \frac{|\forall r_i, r_j \in \mathbb{R}, \text{s.t. } r_i \rightarrow r_j, d_{(i,j)} \leq \tau|}{|\forall r_i, r_j \in \mathbb{R}, \text{s.t. } r_i \rightarrow r_j|} \quad (3)$$

- **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.

$$TRA = \frac{|\forall r_j, \exists r_i \in \mathbb{G}, \text{s.t. } r_i \rightarrow r_j|}{|\forall r_j, \exists r_i \in \mathbb{R}, \text{s.t. } r_i \rightarrow r_j|} \quad (4)$$

- **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).

6.4 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 4. We observe that the average of the time difference $d_{(i,j)}$ is a stable value, though the SD fluctuates a lot. 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 §3.1.

³The emulator is used for labeling the triggering relations. All other experiments are based on the real device.

⁴Classifiers are selected based on the 10-fold cross validation.

No.	Rules (to decide that $r_i \rightarrow r_j$, 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	$r_i.time - r_j.time \leq \tau \wedge r_i.program = r_j.program \wedge func_sim(r_i.host, r_j.host) = true \wedge (r_i.referrer = r_j.referrer \vee r_j.referrer = null).$	Traffic sent from news, media, or education apps.
4	$r_i.time - r_j.time \leq \tau \wedge r_i.program = r_j.program \wedge r_i.destinationIP = r_j.destinationIP \wedge func_sim(r_i.host, r_j.host) = true \wedge r_i.program = r_j.XRequestWith.$	Traffic sent from news or media apps (using AJAX).

Table 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. `func_sim` is used to compare the similarity of two string fields.

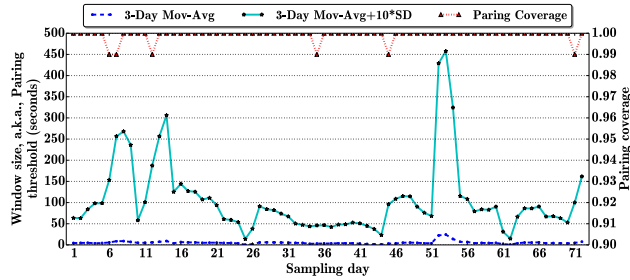


Figure 4: The adaptive pairing window size and its coverage rate for each sampling day on dataset I.

We find that parent-child, sibling, and same-tree relations account for 3%, 19% and 15% of the total number of pairs.⁵ The rest of the pairs (62%) are labeled as no-relation. 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).

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.

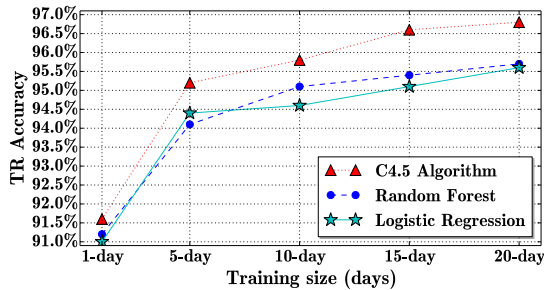


Figure 5: The TR accuracy of dataset I.

All classifiers predict better with the increasing of the training sizes. The C4.5 algorithm outperforms the other two in all settings and it achieves the best TR accuracy at 96.8%.

The wrongly predicted cases mainly come from the browser apps. When users view multiple similar webpages (e.g. price

⁵The 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.

comparison on shopping sites), similar requests are sent to the same host from different tabs. 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.

6.5 Recognition of Auto-generated Traffic

The purpose of this experiment is to evaluate our ability to recognize the auto-generated notifications and updates sent from benign apps. 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 §6.4.

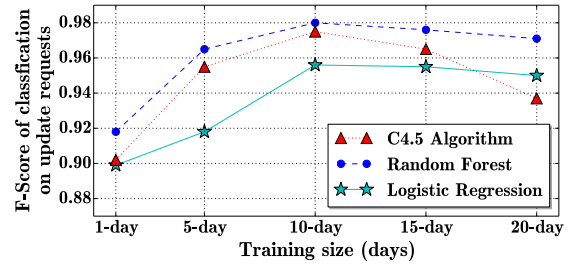


Figure 6: The F-scores of classifying root-trigger requests on dataset I.

In Figure 6, we plot the F-score of the three classifiers on classifying the update requests. 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 predict accuracy and coverage. The prolonged training windows are not effective in characterizing the root-triggers from recently active apps.

Show in Figure 6, random forest classifier outperforms the other two classifiers and it maintains a relatively high and steady accuracy.

Whitelist generation. We generate a list of HTTP entries that belong to the auto-generated notifications and benign update requests. The entries are presented as a 4-tuple (host, destination IP, request type and app name⁶). We label 28 distinct entries in the training data. These entries belong to 6 apps. 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 re-

⁶The request type specifies the GET or POST requests. The app name is identified using the PID and PPID (parent PID) information obtained from `netstat` and `ps` commands.

quests 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.

6.6 TR Analysis on Dataset II

We run the TR Analysis on the dataset II. All TR accuracy results demonstrate the same pattern as shown in Figure 5. 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 results converge to a high accuracy, which shows it is feasible to test real-world (mixed benign and malicious) Android network traffic using a comparable small training dataset.

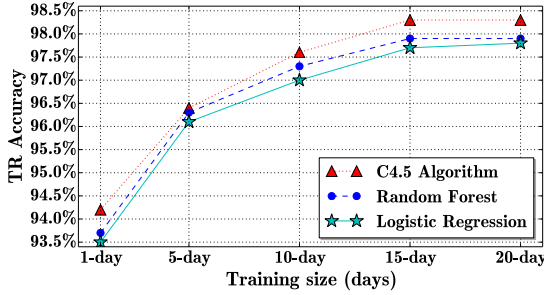


Figure 7: The TR accuracy of dataset II.

6.7 Detection of Malicious Requests

The purpose of this experiment is to evaluate the effectiveness of constructed triggering relation graph in detecting malware network activities. We first remove the notifications and update requests based on the whitelist in §6.5. Thereafter, we run the binary classifiers on the root-triggers of the constructed TRG (training size is 10-day). Shown in Table 5, both tree-based classifiers (C45 and R-F) achieve the better predict accuracy (F-scores) than the log regression.

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

Table 5: Detection accuracy rates on Dataset II.

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 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.

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

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

Table 6: The package name and the communication server of malicious apps in our testing set of dataset II.

The malicious apps we detect cover a wide range of adware, Trojan, bots, spyware and drive-by download apps. 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. We note that the Android bot is not included in the training set, while we can detect it in the testing. We identify the bot’s malicious activities, as their requests are the lack of valid triggers. The detection does not rely on its signatures or C&C servers information.

We regard these malicious apps as *new*, as none of them is in the training set. Their traffic patterns may never be seen before. Results show that our solution successfully identifies all malware by flagging more than 99.1% their requests (out of 33,000+ HTTP requests). Our solution can detect them because the triggering relation tells the causality of requests and the root-triggers reveal the requests’ legitimacy. 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 for detecting malicious URLs and domains use temporal, lexical and host-based features from the root-trigger requests [5, 22]. These features are effective in detecting malware that communicates to the hosts containing abnormal and meaningless URLs. However, their solutions do not consider human inputs and event triggers.

In our experiments, we confirm that the same classifiers, solely using lexical and host-based features, fail to identify 3.3%-4.9% malicious requests and cannot detect 4 malicious apps in Dataset II. By checking the network traffic of these malicious apps, we found that their URLs of communication hosts contain the common English vocabulary seen in benign websites.

FP and FN analysis. Based on the detection using C4.5 algorithm classifier, we find that the FPs are mostly due to

the redirected traffic to some well-known domain servers. E.g., 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. These requests are FPs, because the domain is benign. Other FPs may be due to the uncommon domain names and unconventionally long request strings. FNs are mainly sent by spyware or Trojan.

6.8 Performance

We evaluate the performance in terms of *i*) the pairing efficiency for using our pairing algorithm and a baseline, and *ii*) the running time of testing on pairwise comparisons and detecting 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 [44], 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 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.

Operation	Dataset	Runtime: training + test (seconds)		
Pairing	I	0.064		
	II	0.087		
TR analysis	I	C45	R-F	LOG
		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
		0.05+8.2e-3	0.07+9.7e-3	0.21+1.2e-2
Detection	I	0.03+1.1e-3	0.03+2.0e-3	0.17+4.8e-3
	II	0.03+1.1e-3	0.03+2.0e-3	0.17+4.8e-3

Table 7: The performance of each operation is shown. The results are calculated in seconds per 1000 records. The running time includes both training and testing in TR analysis and detection operations.

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

Summary. We highlight our findings below.

- The C4.5 algorithm and random forest classifiers give high precision and recall on finding the triggering relations and identifying the malicious requests. The false positive rate is 0.7% using the random forest classifier. Our approach to detecting malicious requests is scalable and does not require training on the entire data.
- Our evaluation involving 33,000+ Android app requests achieves a high detection accuracy rate. Results show that 99.1% of stealthy network activities with remote hosts can be identified. We confirm the detection capability of our approach by pinpointing the sparse anomalies out of voluminous traffic data.

7. RELATED WORK

Existing work on protecting Android system and detecting malicious apps is studied in the form of static analysis [21, 40], taint analysis [10, 38, 41], and privilege control [14, 32, 47]. Most static analysis solutions for detecting malicious apps leverage the features from API calls [1, 35, 39], system calls [20], function calls [15].

Dynamic analysis solutions (e.g., [6, 10, 41]) focus on analyzing the application behaviors at runtime. Crowdroid [6] classifies the benign and malicious apps based on the commonly seen system calls, which are obtained from real traces via crowdsourcing. TaintDroid [10] 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 [41] considers the user-intended data transmission on Android and builds the missing leak between the data leaking and user’s interactions, while it needs tremendous efforts in static taint analysis as it treats the apps as whitebox. SmartDroid [48] proposes a hybrid analysis method to identify UI-based event trigger conditions using the sensitive APIs. Our solution systematically differs from the aforementioned dynamic analysis, as ours provides a new aspect by connecting the knowledge from user’s interaction and network traffic for analysis the app’s behavior, while treating the app as blackbox. App Guardian [46] 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. Recent work on policy analysis [31] uses semi-supervised learning to model and analyze the policy refinement process. In our work, we applied the learning-based approach on modeling and enforcing the dependency of network requests.

Dependence analysis plays an important role in detection Android malware. In our work, we utilize the dependence analysis on Android-generated network traffic to detect malicious requests and apps. Our approach extends the technologies on dependence analysis of network traffic [19, 42, 44, 45] and outperforms rule-based methods [8, 36]. Webprophet [19] and Rippler [42] present an approach to actively inject delays to infer the packet or service dependency. Their solutions are designed for the purposes of performance optimization and service dependency inference, as opposed to malware detection. WebWitness [27] is proposed to trace back 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. A most related work is introduced in [44]. Yet, our solution differs from theirs in three aspects.

1. Their work depends on a referrer-based heuristic, which limits its applications. In comparison, we use the delay injection approach for discovering triggering relations.
2. Their work is browser specific, as it requires a browser extension to log users’ inputs. Our solution supports all types of apps on Android and is more general.
3. Their solution requires a straightforward manually generated whitelist to filter out non-user triggered benign requests. We designed a novel two-stage learning approach to automatically recognize benign requests.

Android network traffic classification has been studied in the literature [7,9,33,37]. Authors in [37] proposed to distinguish the mobile traffic by investigating the traffic identify from HTTP headers. NetworkProfiler introduces a technique to generate network profiles for identifying Android apps based on their HTTP traffic [9]. Recent studies [7,23] show that an attacker can recognize user's actions 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, which is beyond the traffic classification and profiling problems.

Also related are studies that proposed classification methods for identifying the suspicious/phishing URLs [5,22,34]. For example, Ma *et al.* proposed to examine the lexical features of the URLs and features of the domain information to identify malicious URLs [22]. Authors in [34] utilized page content-related features to detect phishing pages. Most of their features can be categorized into: *i)* time-based, *ii)* lexical and *iii)* host-based features. However, they do not include the dependency features as ours. Our study makes an important step towards addressing the significance of dependency knowledge in the detection of malicious requests on Android.

8. CONCLUSION AND FUTURE WORK

We described an Android malware detection technique that analyzes the dependency of mobile network traffic. Our analysis explores the request-level traffic dependence and reasons about the root-triggers for all HTTP requests sent from the device. We successfully demonstrated the use of triggering relation discovery for enhancing the sensemaking for security and identifying suspicious requests. Our evaluation confirms that our solution can identify 12 new malicious Android apps that are not previously seen in the training set. For the future work, we plan to deploy our tool to collect Android network traffic and detect more malicious apps via crowdsourcing. Also, we plan to extend our solution for real-time triggering relation inference and online detection.

9. REFERENCES

- [1] Y. Aafer, W. Du, and H. Yin. DroidAPIMiner: Mining API-level features for robust malware detection in Android. In *SecureComm'13*. Springer.
- [2] H. M. Almohri, D. Yao, and D. Kafura. Process authentication for high system assurance. *IEEE TDSC*, 11(2):168–180, 2014.
- [3] Official Android store. <https://play.google.com/store/apps>.
- [4] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. DREBIN: effective and explainable detection of Android malware in your pocket. In *NDSS'14*, 2014.
- [5] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding malicious domains using passive DNS analysis. In *NDSS'11*.
- [6] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for Android. In *SPSM'11*, pages 15–26.
- [7] 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 *CODASPY'15*, pages 297–304, 2015.
- [8] W. Cui, R. H. Katz, and W.-t. Tan. BINDER: an extrusion-based break-in detector for personal computers. In *ACSAC'05*, pages 361–370.
- [9] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song. NetworkProfiler: Towards automatic fingerprinting of Android apps. In *INFOCOM'13*, pages 809–817, 2013.
- [10] 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 TOCS*, 32(2):5, 2014.
- [11] W. Enck, D. Ocateau, P. McDaniel, and S. Chaudhuri. A study of Android application security. In *USENIX'11*, 2011.
- [12] A. Endert, P. Fiaux, and C. North. Semantic interaction for sensemaking: inferring analytical reasoning for model steering. *IEEE VCG*, 18(12):2879–2888, 2012.
- [13] A. Endert, P. Fiaux, and C. North. Semantic interaction for visual text analytics. In *CHI'12*, pages 473–482, 2012.
- [14] Y. Fratantonio, A. Bianchi, W. K. Robertson, M. Egele, C. Kruegel, E. Kirda, and G. Vigna. On the security and engineering implications of finer-grained access controls for android developers and users. In *DIMVA'15*, pages 282–303, 2015.
- [15] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck. Structural detection of Android malware using embedded call graphs. In *AISec'13*, pages 45–54.
- [16] L. Getoor and C. P. Diehl. Link mining: a survey. *SIGKDD Explor. Newsl.*, 7(2):3–12, December 2005.
- [17] I. Kahanda and J. Neville. Using transactional information to predict link strength in online social networks. In *ICWSM'09*.
- [18] Kidlogger. <http://kidlogger.net/about.html>.
- [19] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. G. Greenberg, and Y.-M. Wang. WebProphet: Automating performance prediction for web services. In *NSDI'10*.
- [20] 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.
- [21] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang. CHEX: statically vetting Android apps for component hijacking vulnerabilities. In *CCS'12*, pages 229–240, 2012.
- [22] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In *KDD'09*, pages 1245–1254, 2009.
- [23] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar. I know why you went to the clinic: risks and realization of https traffic analysis. In *PETS'14*, pages 143–163. Springer, 2014.
- [24] D. Muthukumaran, A. Sawani, J. Schiffman, B. M. Jung, and T. Jaeger. Measuring integrity on mobile phone systems. In *SACMAT'08*, pages 155–164. ACM, 2008.
- [25] C. Neasbitt, B. Li, R. Perdisci, L. Lu, K. Singh, and K. Li. WebCapsule: Towards a lightweight forensic engine for web browsers. In *CCS'15*, pages 133–145, 2015.
- [26] C. Neasbitt, R. Perdisci, K. Li, and T. Nelms. ClickMiner: Towards forensic reconstruction of user-browser interactions from network traces. In *CCS'14*, pages 1244–1255.
- [27] T. Nelms, R. Perdisci, M. Antonakakis, and M. Ahamad. WebWitness: investigating, categorizing, and mitigating malware download paths. In *USENIX Security'15*, pages 1025–1040, 2015.
- [28] 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 *SP'12*, pages 224–238. IEEE, 2012.
- [29] D. Shackleford. Blind as a bat? supporting packet decryption for security scanning (white paper). November 2012.
- [30] G. Snedecor. *Statistical methods*. Iowa State University Press, 1989.
- [31] R. Wang, W. Enck, D. Reeves, X. Zhang, P. Ning, D. Xu, W. Zhou, and A. M. Azab. EASEAndroid: Automatic policy analysis and refinement for security enhanced Android via large-scale semi-supervised learning. In *USENIX Security'15*, pages 351–366, 2015.

- [32] Y. Wang, S. Hariharan, C. Zhao, J. Liu, and W. Du. Compac: Enforce component-level access control in Android. In *CODASPY'14*, pages 25–36.
- [33] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. ProfileDroid: multi-layer profiling of Android applications. In *Mobicom'12*, pages 137–148, 2012.
- [34] C. Whittaker, B. Ryner, and M. Nazif. Large-scale automatic classification of phishing pages. In *NDSS'10*.
- [35] 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 *Asia JCIS'12*, pages 62–69.
- [36] G. Xie, M. Iliofotou, T. Karagiannis, M. Faloutsos, and Y. Jin. ReSurf: Reconstructing web-surfing activity from network traffic. In *IFIP Networking Conference, 2013*, pages 1–9.
- [37] 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 *INFOCOM'15*.
- [38] L. Yan and H. Yin. DroidScope: Seamlessly reconstructing the OS and dalvik semantic views for dynamic Android malware analysis. In *USENIX'12*, pages 569–584, 2012.
- [39] 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 *ESORICS'14*, pages 163–182. Springer, 2014.
- [40] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck. AppContext: Differentiating malicious and benign mobile app behaviors using context. In *ICSE'15*, 2015.
- [41] 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 *CCS'13*, pages 1043–1054.
- [42] A. Zand, G. Vigna, R. Kemmerer, and C. Kruegel. Rippler: Delay injection for service dependency detection. In *INFOCOM'14*.
- [43] H. Zhang, M. Sun, D. Yao, and C. North. Visualizing traffic causality for analyzing network anomalies. In *IWSPA'15*, pages 37–42, 2015.
- [44] H. Zhang, D. Yao, and N. Ramakrishnan. Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery. In *ASIACCS'14*, pages 39–50.
- [45] H. Zhang, D. Yao, N. Ramakrishnan, and Z. Zhang. Causality reasoning about network events for detecting stealthy malware activities. *Computers & Security*, 58:180–198, 2016.
- [46] N. Zhang, K. Yuan, M. Naveed, X. Zhou, and X. Wang. Leave me alone: App-level protection against runtime information gathering on Android. In *SP'15*, pages 915–930, 2015.
- [47] X. Zhang, A. Ahlawat, and W. Du. AFrame: Isolating advertisements from mobile applications in Android. In *ACSAC'13*, pages 9–18.
- [48] 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 *SPSM'12*, pages 93–104. ACM, 2012.
- [49] Y. Zhou and X. Jiang. Dissecting Android malware: Characterization and evolution. In *SP'12*, pages 95–109. IEEE, 2012.

Visualizing Traffic Causality for Analyzing Network Anomalies*

Hao Zhang, Maoyuan Sun, Danfeng (Daphne) Yao, and Chris North
Department of Computer Science
Virginia Tech, Blacksburg, VA
{haozhang, smaoyuan, danfeng, north}@cs.vt.edu

ABSTRACT

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 detect attacks and pinpoint their causes. We design and develop a tool to visually represent the causal relations for network requests. The traffic causality information enables one to reason about the legitimacy and normalcy of observed network events. Our tool with a special *visual locality* property supports different levels of visual-based querying and reasoning required for the sensemaking process on complex network data. Leveraging the domain knowledge, security analysts can use our tool to identify abnormal network activities and patterns due to attacks or stealthy malware. We conduct a user study that confirms our tool can enhance the readability and perceptibility of the dependency for host-based network traffic.

Keywords

Anomaly Detection, Network Traffic Analysis, Information Visualization, Usable Security, Visual Locality

1. INTRODUCTION

This paper 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 [25]. Its unique advantage is the capability of reasoning the causality or dependency of network data and thus detecting new stealthy malware activities. The analysis provides automatic anomaly detection in the observed network activities through probabilistic reasoning of the causal relations in traffic. By pinpointing abnormal network events

*This work has been supported in part by NSF grant CAREER CNS-0953638, ARO YIP W911NF-14-1-0535, and L-3 communications.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
IWSPA'15, March 4, 2015, San Antonio, Texas, USA.
Copyright © 2015 ACM 978-1-4503-3341-2/15/03 ...\$15.00.
<http://dx.doi.org/10.1145/2713579.2713583>.

that lack of valid triggers, it can detect malware activities on an infected machine (e.g., making command-and-control communications with its controller). The triggers include legitimate user events and benign network packets.

In this work, we design a tool to assist the analysis of host-based network data based on traffic causality. 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) [3, 5, 13, 18]. 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 underlying relationship among network events, with one notable exception Portall [7]. 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 [7]. Thus, new visual representation approaches are needed. Our solution aims to satisfy a 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*.

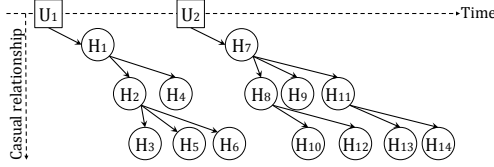
A straightforward approach for displaying host-based traffic is shown in Figure 1a. This visual representation arranges network requests using a forest layout based on their causal relations; the timeline may be extended horizontally when newer network events are added. Because the forest-based layout is intuitive, it has been used for illustrating relations among network events [14, 25]. However, it 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. Statistics shows that more than 90% of the request causal relation falls within a 30-second interval [25]. Although rare, we observe that network requests that occur 15 minutes apart may have causal relations. Therefore, our visualization design takes these unique traffic characteristics into considerations. To this end, we focus on displaying items that have causal relations in visually adjacent space. Causal relations determine their locations on the display. This layout provides two advantages: *i)* enhancing the navigation of traffic causality, and *ii)* improving the identification of anomalous activities.

Our contribution in this work is twofold.

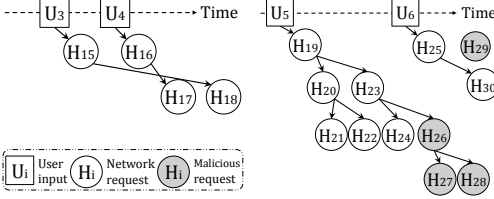
- We develop a visualization tool for security analysts to efficiently display the network traffic dependency. This tool has a *visual locality* feature that can optimize the displaying of structured data. The visual representation is a *radial* layout based on curved timeline display.

Compared to the straightforward forest 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.



(a) A straightforward representation of TRG.



(b) Two crossing edges. (c) Malicious requests.

Figure 1: Schematic drawing of traffic causality for outbound network requests on a computer. Nodes are indexed by their relative occurrence time.

2. TRG AND SECURITY MODEL

DEFINITION 1 (Triggering Relation Graph [25]).

TRG is composed of two types of nodes, user input and network request. The edges in TRG refer to the triggering relations that describe the causal relationship between nodes. The dependencies are defined by two types of edges: i) root-trigger dependency defines the relation between a legitimate user's input (e.g., mouse clicks on hyperlinks) and its generated first network request and ii) inter-request dependency is the relation between two network requests where one directly triggers the other.

Figure 1a shows an example of a TRG, where $U_1 \rightarrow H_1$ is an example of a root-trigger dependency edge and $H_1 \rightarrow H_2$ is an example of an inter-request dependency edge.

Triggering relations can be computed by using rule-based methods [24] or machine learning techniques [25]. TRG 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. In this model, the events having triggering relations to a legitimate cause are benign ones.

However, there is no existing tool to display the triggering relation graph. Straightforward attempts to visualize the TRG (e.g., using conventional straight timeline layout) may not satisfy the high visual locality requirement. In addition, the requests with causality do not necessarily situate close

according to the time, as illustrated in Figure 1b (H_{15} - H_{18}). The crossed edges make the TRG messy and hard to analyze.

DEFINITION 2 (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. 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 1c).
- Network requests sent to malicious hosts with valid referrer information are referred to as *grafted* (H_{26} - H_{28} in Figure 1c).

The vagabond requests are events without legitimate causal relations and likely 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.

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 to only 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 [8]). After the servers are compromised, attackers can modify the websites by injecting malicious codes, then the clients get infected when they visit the websites. These grafted requests are often of the Javascript type with long and obfuscated request strings in URLs, because Javascript requests have diversified functions to be leveraged by attackers. Domain experts have to reason the legitimacy by integrating with other information (e.g., system logs).

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.

3. 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 on 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 upgrade domains to reduce the false alarms (e.g., `Java Update Checker` and its update domains `javadl-esd.sun.com` and `javadl.oracle.com`).

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 at the display, and the radiating branches represent network traffic 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.

3.1 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 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 2b, we arrange the nodes in a clockwise manner. In this design, *time line* 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.

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 of each tree guarantees there is 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

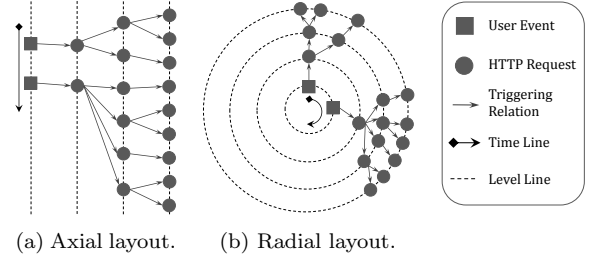


Figure 2: Schematic diagrams of two visualization designs for the structured network data.

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.

Our design maximizes the usage of display space. The radial layout presents hierarchies of events in concentric rings. Users using the conventional forest-like layout need to scroll *twice* more than our design. Additionally, by concentrating nodes at the center of the screen, the radial layout enables users to easily manipulate the display.

3.2 Interactive Heatmap for Accessing Original Traffic Logs

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 4). The **LogMap** reveals the density information of the network requests. It provides an overview of the request distribution over the observed period. The **LogMap** accommodates different levels of time-period granularities, e.g., seconds, minutes, hours, etc. Last, security analysts often resort to the original logs for more details. 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 and organizes network events occurred in each time 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 granularity of events displayed on the **LogMap** can be adjusted according to analysts' needs.

3.3 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 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 condensing 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 evaluate 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 1, we compare the number of nodes in both original and condensed views. *Compression ratio* is defined to qualify 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.

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%

Table 1: The number of HTTP requests on each level in the Triggering Relation Graph for original and condensed views.

4. 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 3.

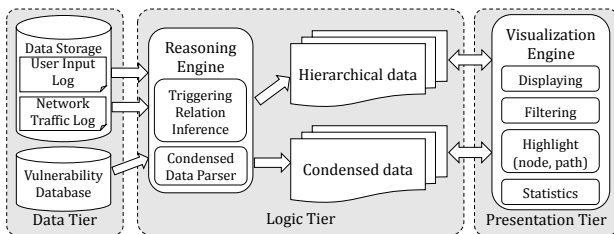


Figure 3: The workflow of ReView.

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). In the *data tier*, we add a customized vulnerability database, which is composed of several known blacklists and feeds [15, 16, 23], 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.

In the *reasoning engine*, our tool infers the triggering relations by leveraging a machine learning-based solution proposed in [25]. It discovers vagabond request that are not generated by any user actions. ReView screens out the benign update traffic using a whitelist. It also filters the traffic according to our customized vulnerability database. Other advanced solutions to rank domains and detect malicious URLs (e.g., [21]) can be also integrated into our tool. In addition, we leverage the condensing algorithm to preprocess the raw traffic data in the *logic tier*.

The rendering in ReView uses D3 [6], which is a Javascript library for data visualization. The layout of ReView is shown in Figure 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 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 complimentary 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.

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 with condensed data

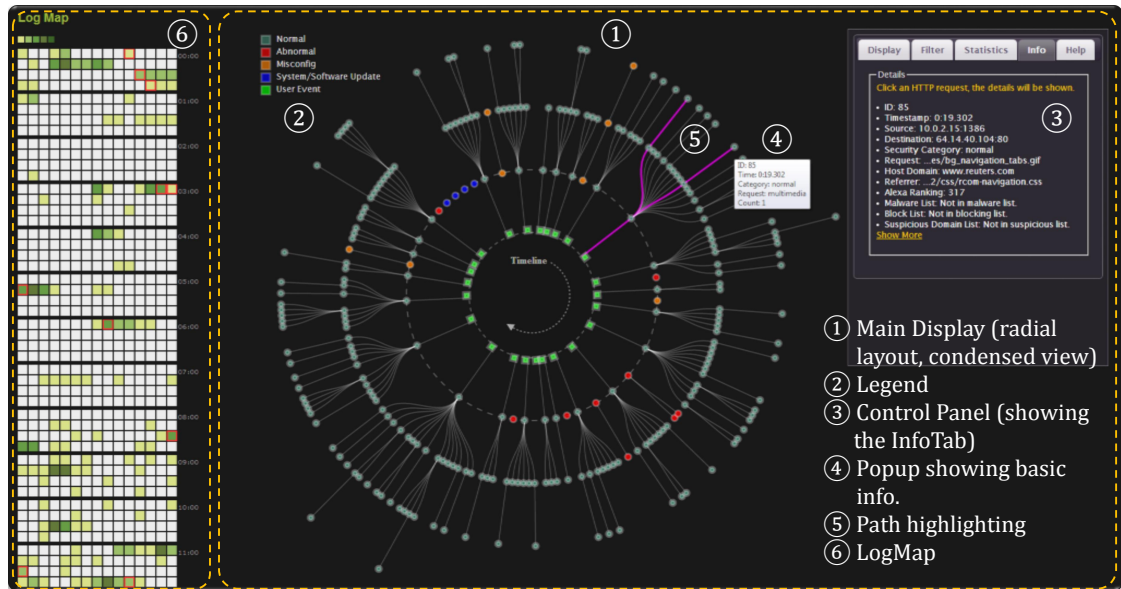


Figure 4: A screen layout of ReView.

and a complete view with all triggering relations; *ii*) how our tool can direct users to analyze the suspicious requests.

We conducted the study in a computer science laboratory in 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 routinely done by analysts and the user preference of different visualization options.

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. The test data contains a total of 3724 HTTP requests, including 24 update requests and 33 vagabonds (12 abnormal and 21 misconfigured requests).

5.1 Analysis of User Preference

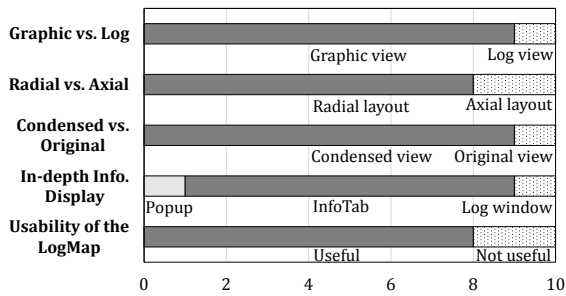


Figure 5: The breakdown of results on user preference in the user study.

Figure 5 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 5, nine participants indicated that they preferred visualizations over traditional network logs.

They agreed on the fact that the provided visualizations in ReView summarize the network data, compared with the overwhelming information in raw logs. With ReView, users can quickly understand and make use of the structure of network causal relationship for security analysis.

ReView effectively navigates users among a large amount of network requests through different levels of abstractions. Eight participants preferred the radial layout over the axial one, because it enables the efficient utilization of the screen space. In addition, participants choosing the radial layout mentioned that they can navigate the display with minimal mouse scrolling needed. Other participants prefer the axial layout because they think the forest-like design is more salient and intuitive.

Nine participants preferred the condensed view over the original one. They agreed that the condensed view simplified representations by avoiding unnecessary leaf nodes. One participant who was in favor of the original view also admitted that condensed view is neat, but he still chose original view in case of missing necessary information. Last, eight participants found LogMap useful for showing the network requests by their distributions over time.

5.2 Analysis of Suspicious Requests

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 example, 8 participants answered that the first malicious request was sent to `spi.domainsponsor.com` and the domain was black-listed as spyware. By reviewing its long URL string, participants found out that the request URL contains some substrings, 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.

6. RELATED WORK

The dependency analysis provides an effective way to pinpoint the origin of the vulnerability and reason the complex structures (e.g., finding the dependency of network services [4] and intrusion detection logs [12], detecting drive-by download attacks [20]). However, the dependency analysis on network requests has not been well studied, with a few exceptions [14, 19]. WebProphet [14] extracts the dependencies between web objects in the light of their delays. ReSurf [19] reconstructs web surfing activities from network traces based on a referrer-based inference.

A comparative study from Goodall [9] confirmed that visualizations, compared with traditional logs, can help users easily perceive patterns and anomalies for cyber security related analysis. Many visualization designs have been proposed as intuitive and efficient means of displaying complex structures that cannot be explicitly identified in the raw data [5, 17]. Radial Traffic Analyzer [11] and Traffic Circle [2] are two network visualization tools that adopt the radial layout. Both solutions are used for monitoring the network (e.g., identifying communication partners and traffic types) by displaying the raw packet or flow information. In ReView, we use concentric arcs to represent different dependency levels, in such way that our tool provides logical relation of the network traffic in an information-rich fashion.

In recent years, many research efforts have been dedicated to visualize the network traffic, as visualization tools help researchers understand the structure of data [17, 22] and identify the anomalies [1, 10]. Traffic Causality Graph (TCG) [1] 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 [1] is based on the protocol, IP and port information of flows, while ours is based on higher level application layer information, which is a fine-grained model and has better descriptive power. Traffic Dispersion Graph (TDG) [10] is used to present the network-wide flows by aggregating the packets. TDG captures the interaction among hosts in a network, while our TRG is on the host-based network traffic. Last, our model differs from TCG and TDG, as their solutions aim at classifying the traffic or detecting port-related threats (e.g., port scanning). However, our TRG is for identifying the abnormal requests through revealing the triggering relations of outbound packets.

7. CONCLUSIONS

Discovering traffic dependency has been shown to be an effective way to analyze network activities and identify ma-

licious events. We introduced a new concept of high visual locality and developed ReView, a visualization tool that maximizes the usage of screen and helps security analysts better utilize network traffic dependency. In our design, we adopted a radial layout that supports to high visual locality. ReView serves as an integrated solution for analysts to accurately pinpoint anomalous network events and perform further investigation. User study results suggest that our tool provides usable visualization and interaction to help interpret network traffic causality and enhance security analysis.

8. REFERENCES

- [1] H. Asai, K. Fukuda, and H. Esaki. Traffic Causality Graphs: Profiling network applications through temporal and spatial causality of flows. In *ITC'11*, pages 95–102, 2011.
- [2] D. M. Best, S. Bohn, D. Love, A. Wynne, and W. A. Pike. Real-time visualization of network behaviors for situational awareness. In *VizSec'10*, pages 79–90, 2010.
- [3] A. Boschetti, L. Salgarelli, C. Muelder, and K.-L. Ma. TVi: a visual querying system for network monitoring and anomaly detection. In *VizSec'11*, page 1, 2011.
- [4] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl. Automating network application dependency discovery: Experiences, limitations, and new solutions. In *OSDI'08*, pages 117–130.
- [5] 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. *Computer Graphics & Applications, IEEE*, 26(2):60–70, 2006.
- [6] D3.js: a JavaScript library to display given digital data into graphic, dynamic forms. <http://d3js.org>.
- [7] G. A. Fink, P. Muessig, and C. North. Visual correlation of host processes and network traffic. In *VizSec'05*, page 2, 2005.
- [8] J. Fonseca, M. Vieira, and H. Madeira. Vulnerability & attack injection for web applications. In *DSN'09*, pages 93–102, 2009.
- [9] J. R. Goodall. Visualization is better! A comparative evaluation. In *VizSec'09*, pages 57–68, 2009.
- [10] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network monitoring using traffic dispersion graphs (TDGs). In *IMC'07*, pages 315–320, 2007.
- [11] D. A. Keim, F. Mansmann, J. Schneidewind, and T. Schreck. Monitoring network traffic with radial traffic analyzer. In *VAST'06*, pages 123–128, 2006.
- [12] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen. Enriching intrusion alerts through multi-host causality. In *NDSS'05*, 2005.
- [13] H. Koike and K. Ohno. SnortView: Visualization system of Snort logs. In *VizSEC/DMSEC'04*, pages 143–147, 2004.
- [14] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. G. Greenberg, and Y.-M. Wang. WebProphet: Automating performance prediction for web services. In *NSDI'10*, pages 143–158, 2010.
- [15] Malware domain list. <http://www.malwaredomainlist.com>.
- [16] Suspicious domains from SANS institute. https://isc.sans.edu/suspicious_domains.html.
- [17] H. Shiravi, A. Shiravi, and A. A. Ghorbani. A survey of visualization systems for network security. *IEEE Trans. Vis. Comput. Graph.*, 18(8):1313–1329, 2012.
- [18] H. Song, C. Muelder, and K.-L. Ma. Crucial nodes centric visual monitoring and analysis of computer networks. In *CyberSecurity*, pages 16–23, 2012.
- [19] 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.
- [20] K. Xu, D. Yao, Q. Ma, and A. Crowell. Detecting infection onset with behavior-based policies. In *NSS'11*, pages 57–64.
- [21] L. Xu, Z. Zhan, S. Xu, and K. Ye. Cross-layer detection of malicious websites. In *CODASPY'13*, pages 141–152, 2013.
- [22] W. Yu and R. M. Verma. Visualization of rule-based programming. In *SAC'08*, pages 1258–1259, 2008.
- [23] ZeuS tracker domain blacklist. <https://zeustracker.abuse.ch/blocklist.php>.
- [24] H. Zhang, W. Banick, D. Yao, and N. Ramakrishnan. User intention-based traffic dependence analysis for anomaly detection. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, pages 104–112, 2012.
- [25] H. Zhang, D. Yao, and N. Ramakrishnan. Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery. In *ASIACCS'14*, pages 39–50, 2014.