A Spatiotemporal Event Correlation Approach to Computer Security

Yinglian Xie

CMU-CS-05-175

August 2005

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the US Army Research Office under contract no. DAAD19-02-1-0389, the US Air Force Research Laboratory under grant no. F30602-02-1-0139, and the National Science Foundation under grant no. ANI-0326472 and ANI-9814929. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Correlation, Computer security, Anomaly detection, Forensics, PCA, Wavelet analysis, File updates, Random walks, Spectral analysis, Sampling, Worm, Network flows

Abstract

Correlation is a recognized technique in security to improve the effectiveness of threat identification and analysis process. Existing correlation approaches mostly focus on correlating temporally located events, or combining alerts from multiple intrusion detection systems. Such approaches either generate high false alarm rates due to single host activity changes, or fail to detect stealthy attacks that evade detection from local monitors.

This thesis explores a new *spatiotemporal event correlation* approach to capture the abnormal patterns of a wide class of attacks, whose activities, when observed individually, may not seem suspicious or distinguishable from normal activity changes. This approach correlates events across both space and time, identifying aggregated abnormal event patterns to the host state updates. By exploring both the temporal and spatial locality of host state changes, our approach identifies malicious events that are hard to detect in isolation, without foreknowledge of normal changes or system-specific knowledge. To demonstrate the effectiveness of spatiotemporal event correlation, we instantiate the approach in two example security applications: anomaly detection and network forensics.

For anomaly detection, we present a "pointillist" method to detect similar, coincident changes to the patterns of file updates that are shared across multiple hosts. The correlation is performed by clustering points, each representing an individual host state transition, in a multi-dimensional feature space. We implement this approach in a prototype system called *Seurat* and demonstrate its effectiveness using a combination of real workstation traces, simulated attacks, and manually launched real worms.

For network forensics, we present a general forensics framework called *Dragnet*, and propose a "random moonwalk" technique that can determine both the host responsible for originating a worm attack and the set of attack flows that make up the initial stages of the attack tree via which the worm infected successive generations of victims. Our technique exploits the "wide tree" shape of a worm propagation by performing random walks backward in time along paths of flows. Using analysis, simulation, and experiments with real world traces, we show how the technique works against both today's fast propagating worms and a wide class of stealthy worms that attempt to hide their attack flows among background traffic.

While the high level idea is the same, the two applications use different types of event data, different data representations, and different correlation algorithms, suggesting that spatiotemporal event correlation will be a general solution to reliably and effectively capture the global abnormal patterns for a wide variety of security applications.

Acknowledgements

I am extremely grateful to my advisors David O'Hallaron and Hui Zhang. Without their guidance and tremendous support, this thesis would not be possible. They guided me through every aspect of my graduate study. They not only teach me how to identify problems, how to solve problems, and how to build systems, but also help me improve my writing skills, speaking kills, and communication skills. Numerous times, they helped me set milestones together, and guided me to make progress steadily. When there are difficulties, their encouragement and optimism help me to gain more confidence, and to come up with solutions. I treasure my PhD study as a privileged opportunity to learn from both great teachers and masters. Their insights and suggestions greatly increased the quality of this thesis work.

I would like to thank other members in my thesis committee - Mike Reiter and Albert Greenberg. Mike is a source of inspiration, and guided the work in this thesis from the very beginning. His guidance, suggestions, and help greatly improved every aspect of this thesis step by step. This thesis work would not be possible without him either. I would also like to thank Albert for his valuable suggestions on the Dragnet project that became part of this thesis. His comments and feedbacks on our overall work has helped improve my thesis.

I thank my collaborators Dave Maltz, Hyang-Ah Kim, and Vyas Sekar. Much of this thesis work is done in collaboration with them, who have been both team players and friends. My work with them is both happy and fruitful. Dave actively discussed the Dragnet work together with me and provided a lot of help to move the project forward successfully. Hyang-Ah, Vyas, and I help each other and grow together.

Thanks to my fellow students Tiankai Tu, Julio Lopez, Yanghua-Chu, Aditya Ganjam, Debabrata Dash, Andy Myers, Justin Weisz, Hong Yan, and friends Shuheng Zhou, Jun Gao, Mukesh Agrawal, Ningning Hu, Suman Nath, Runting Shi, Jimeng Sun, Yan Li, as well as many others, for making my student journey in CMU so enjoyable.

I thank my husband Qifa Ke for his love, encouragement, and support. Although we work in different areas, we discuss different aspects of research together, and move forward together. Our discussion often results in useful suggestions that improved not only this thesis work, but my research approaches and styles in general. Finally, I really thank my parents for their great love and tremendous support in every aspect of my life. They support me to pursue my own goals with confidence, to progress with sweat and hard work, and to face and overcome obstacles with courage. This thesis is dedicated to my parents!

Contents

List of Figures

Chapter 1

Introduction

Correlation, defined as "establishing or finding relationship between entities", is a recognized technique in security to improve the effectiveness of threat identification and analysis process by combining information from multiple sources. With the increasing deployment of various security monitors and sensors, effectively analyzing audit data for extracting only desired information is an important step to attack discovery, attack response, forensic analysis, and prediction of future attacks. By looking at collective information on a set of events rather than the individual ones, one can identify more types of attacks with fewer false positives.

The world of network security, however, is an arms race. While many correlation techniques today are effective at identifying a wide variety of existing attacks, future attacks can be more stealthy, constantly changing their signatures to evade the detection of individually deployed monitors. Nevertheless, in order to infect a large number of hosts in a network and disrupt their normal functions, large scale attacks will exhibit discernible patterns when locally generated events are viewed in an aggregated way. Detecting and identifying these more sophisticated attacks requires us to correlate the huge volume of events taking place at distributed locations in the network. While the sheer quantity of event records could be overwhelming, the increasing performance and capacities of computing devices, networks, and storage devices make it possible to store, transfer, and analyze large quantities of audit data in a global coordinated way.

The topic of this thesis is a new *spatiotemporal event correlation* approach to capture the global abnormal patterns of a wide class of attacks, whose activities, when observed individually, may not seem suspicious or distinguishable from normal host activity changes. In this introductory chapter, we first review existing correlation approaches in computer security. We then more formally define our approach, describe the challenges, and present the methodology used in the thesis.

1.1 Existing Correlation Approaches

Figure 1.1 shows the different ways of classifying correlation approaches in computer security. Based on the source of input data, we can classify the existing approaches as temporal correlation and spatial correlation. The temporal correlation approach relates and analyzes event sequences that span across a range of time period. Such approaches either look for deviations from a normal event model defined through learning, or specify a set of rules to encode sequences of events that known attacks must follow.

In the learning based temporal correlation, the system is usually trained to learn characteristics of normal events. Further investigation is carried when there are significant deviations from the learned normal model. For example, Warraender et al. have proposed a number of machine learning approaches to detect anomalous program executions based on short sequences of run-time system calls [108]. Because such approaches build normal models upon past known activities, any new unseen event is suspicious, possibly resulting in a high false positive rate [26].

Rule-based temporal correlations usually define a set of rules that will be used to match a sequence of events across different times. Such event sequence can be encoded as either normal activities (in which case, an alarm will be raised in case of rule violation), or attacks (in which case, an alarm will be raised in case of rule matching). Although rule-based temporal correlation generally raises fewer false alarms for security investigation, it may not cope with new types of attacks. Furthermore, such approaches often require system administrators to manually specify rules based on accurate, specific knowledge about normal user activities or known attack patterns, which is not only time consuming, but also difficult.

The spatial correlation approach focuses on analyzing events that take place across multiple locations for security diagnosis. Such events can also distribute across a time range. The input to the correlation modules can be either low level observed raw events or high level alert events generate from local audit data. Examples of low level raw events include file system updates, network packets, and system call invocations. High level alerts usually refer to filtered abnormal events or aggregated event summaries output by IDS systems such as Snort [83] or Bro [75]. In order to distinguish these two types of inputs in this document, we define low level raw events as *events* and define high level alert events as *alerts*.

Most of the existing spatial correlation techniques focus on correlating high level alerts. Since a single attack can often induce multiple different alerts generated locally from raw event inputs, these alerts can be further aggregated to output attack scenarios to system administrators. Both statistical methods and rule based methods can be applied to spatial correlation of alerts. Example techniques include statisti-

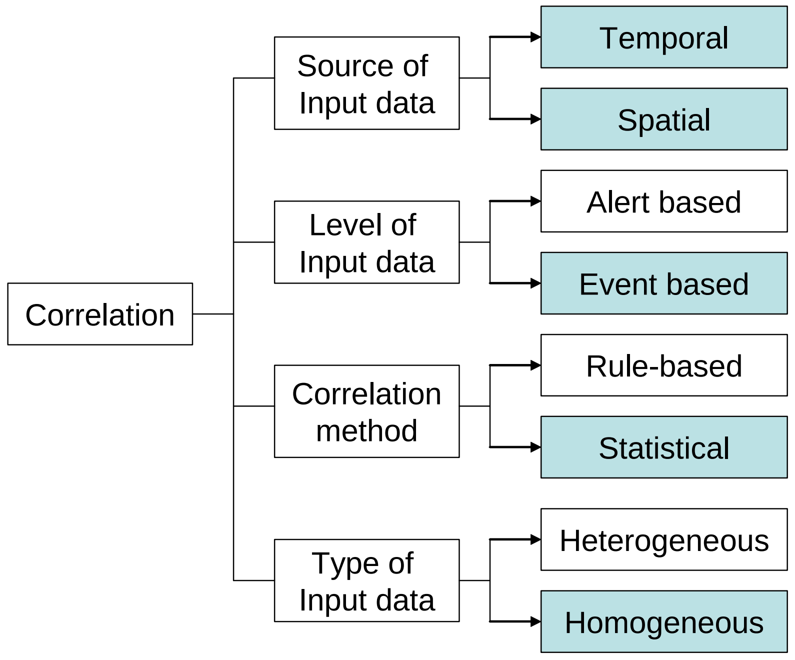


Figure 1.1: Classification of correlation approaches in security.

cally clustering alerts based on similarity of alert attributes [103] for attack causality analysis, and combining alerts using well defined logical rules [23]. Compared with low level event records, the number of high level alerts is significantly smaller. Hence correlating these alerts generally has manageable complexity. The disadvantage, however, is its limited detection capability on stealthy attacks that do not induce alerts based on only locally observed events [80], even though they may exhibit abnormal patterns when viewed globally. As attackers get more sophisticated, they may try to blend attack events gradually into normal activities to evade individual monitor detection.

Another orthogonal classification of correlation approaches is based on whether the input events are of a same type or different types. Correlating heterogenous types of events can potentially improve the accuracy of alarms based on different views of system states. For example, Abad et al have proposed both a top-down approach and a bottom-up approach to correlate anomalies from different types of logs for intrusion detection [3]. In the top-down approach, known attacks are analyzed to determine attack signatures from various logs, while in the bottom-up approach, anomalies from multiples types of logs are correlated to detect new attacks. Other examples include [4], where the authors correlate alerts from heterogenous sensors using a similarity measure based on overlapping features. Because many these existing approaches still require local event filtering before correlation, their capability is,

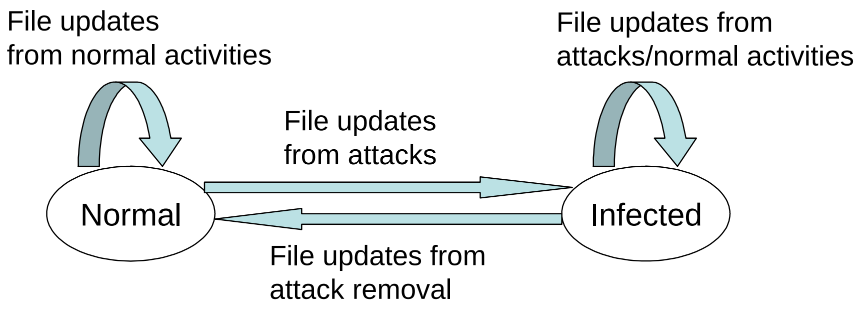

File updates from normal activities
File updates from attacks/normal activities
File updates from attacks
Normal
Infected
File updates from attack removal

Figure 1.2: File update events vs. host states.

again, limited for identifying stealthy attack patterns that can evade the detection of individual monitors.

In summary, most of the existing correlation approaches have focused on either temporal correlation, or correlating alerts from individual monitors. They usually generate high false positive rates, or fail to detect stealthy attacks that do not seem abnormal based on local views.

1.2 Spatiotemporal Event Correlation

In this thesis, we explore a new approach of correlation for computer security. This approach correlates events across both space and time, identifying aggregated abnormal event patterns to the host state updates. We focus on low level observed events, such as host file system updates or network communication flows, instead of high level IDS alerts. As such, we define our approach as *spatiotemporal event correlation*.

More formally, a state of a host is a well-defined logical or operational mode. An event is a sequence of actions that, directly or indirectly, cause the state of a host to transit from one to another. Such transition can happen between the same states or different states. For the purpose of security applications, we consider only two types of logical states of a host: “normal” and “infected”. A host is in the normal state if it is in the lack of any existing or potential malicious code execution, otherwise, the host is in the infected state. Example events that may cause a host's state to transit from “normal” to “infected” include file system updates, network communication flows, keyboard inputs, or mouse clicks from human users.

In the case of file system updates, as illustrated by Figure 1.2, normal file updates, generated by user activities or system routine maintenance, will transit the host state between normal states. However, file creations, modifications, or deletions induced by a virus or worm for the first time, will change the state of a vulnerable host

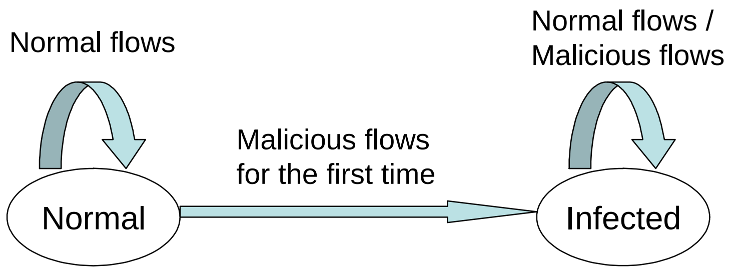


Figure 1.3: Network flow events vs. host states.

from “normal” to “infected”. Once a host is in the infected state, further file updates caused by viruses or worms, will transit the host state between infected states without changing it. Finally, after we detect the infection, those file updates associated with the virus removal process will change the host state from “infected” back again to “normal”. Similarly shown in Figure 1.3, the reception of a communication flow that carries malicious payload for the first time will turn a vulnerable host’s state from “normal” to “infected”, while the reception of other types of flows will not change the host state.

Spatiotemporal event correlation is based on a key observation that events of interest in a network system often have both temporal and spatial locality. In particular, events induced by malware propagation often exhibit spatial locality, in the sense that similar updates or events tend to occur across many of the hosts in a network system. If one host is compromised by an attack, other hosts with similar vulnerabilities in the network are likely to be compromised as well, hence generating similar events or updates. These events also exhibit temporal locality in the sense that they tend to be clustered closely in time. If one host is compromised by a malicious attack, other hosts are likely to be compromised by the same attack soon afterwards. Each such event, when viewed individually, may not seem suspicious or abnormal. When they are viewed collectively, their abnormal patterns may stand out due to their locality across space and time. The goal of spatiotemporal event correlation is therefore to identify atypical such aggregate events, or the lack of typical ones.

1.2.1 Why Spatiotemporal Event Correlation?

The correlation capability across both space and time allows us to learn the patterns of normal events at individual locations over time, and thus detect new abnormal events. More importantly, we can combine and compare these events across multiple locations, hence reliably identifying only aggregated abnormal event patterns that are caused by malicious intrusions. Such approach can eliminate false alarms caused by normal activity pattern shifts at single locations, without foreknowledge of normal changes and without system-specific knowledge for rules.

Meanwhile, by globally correlating events across multiple locations, it is possible to identify stealthy attacks that may not be detectable by looking only at events at individual locations. While many existing approaches such as [44] have focused on identifying virulent, fast propagating attacks, future attacks can potentially be much stealthier. According to reports on recent attack trends [19, 90], both the effectiveness of infection and the level of attack sophistication have been increasing. On one hand, the degree of automation in attack tools and the speed of discovering vulnerable hosts have continued to advance, leaving less time for attack response and attack defense. On the other hand, attacks are getting increasingly *stealthier* to mimic normal host communication behaviors. They can propagate via various methods (e.g., hitlist scan, peer-to-peer propagation), self-evolve to use different signatures (e.g., metamorphic worms, polymorphic worms), and exploit well-known protocols and ports (e.g., IRC, HTTP). Such attack events, when viewed from a single location in isolation, may seem subtle or invisible to trigger local alerts, their abnormal structures or patterns will potentially stand out when viewed aggregately. By identifying these global abnormal patterns or structures based on events instead of alerts, the spatiotemporal event correlation approach can be agnostic to attack signatures or scanning rates, and potentially be applicable to a wide range of attacks.

In summary, by correlating events across both space (multiple locations) and time (past and present):

- *We can distinguish malicious behavior from normal activity changes more reliably to reduce false positive rates.*
- *We can identify a wide class of attacks that do not trigger local alarms, but exhibit discernable global patterns.*

1.2.2 Challenges Involved

The spatiotemporal event correlation approach requires us to process low level raw event inputs across both space and time. There are thus a larger volume of data to be collected and analyzed, compared with analyzing events from single locations, or correlating pre-filtered alerts generated by local monitors. Furthermore, the data to be collected and analyzed may belong to different domains or entities, containing private information about both administrative domains and end users. Consequently, there will be a number of challenges for the approach to be effective in practice as a result of the increased scale of data and the need for sharing data among different users or service providers.

1. **Compact event representation:** We need compact representations of events in order to reduce both the amount of audit data to be processed and the complexity of the correlation engine. A more compact and generic representation is also more robust to various types of attacks, because there will be less room for attackers to exploit in order to evade detection.

2. **Efficient correlation algorithm:** The key part of correlation lies in the algorithmic components that effectively correlate the events to extract "interesting" global patterns for attack detection and analysis. Since the volume of audit data to be correlated can be much larger than the number of pre-filtered alerts, the correlation algorithm itself should have low complexity for it to be used in practice. The correlation algorithms may need to operate over a centralized data repository (in the case of one domain), or will need to interact with distributed data retrieval and query mechanisms (in the case of multiple domains).

3. **A framework for data storage and query:** The distributed nature of audit events requires a framework to collect, store, and query these events for efficient correlation analysis. One challenging problem is how to deploy data monitors to maximize coverage while minimizing costs. Once collected, these audit data may need to be stored at different repositories due to either the scale of the network size, or the boundary of administrative domains. In such case, queries to the events must be efficiently routed to the correlation module for fast detection and response.

4. **Privacy protection:** Since the events to be correlated relate with the states of different hosts and users, issues of trust and cooperation raise challenges with respect to privacy protection, especially when audit data distribute across multiple independent domains. In addition, low level observed events tend to contain more sensitive information about both ISP and user privacy than high level alerts. It is important for the data query and correlation process to leak as minimum information as possible about both the domain proprietaries and end users, while still be able to effectively identify abnormal patterns for detection and analysis.

1.2.3 Thesis Approach

Spatiotemporal event correlation is a general approach that can be applicable to various security problems. While the high level idea is the same, given a specific problem, one needs to address each of the above challenges based on application specific semantics.

In the scope of this thesis, we explore the spatiotemporal event correlation approach in the context of two important security applications. We focus on addressing the first two challenges mentioned above. We show that, in spite of the data complexity, one can effectively reduce the volume of input data through feature selection mechanisms and compact graph representations. We also present two different algorithms that effectively identify the global abnormal patterns or abnormal graph structures by leveraging application domain knowledge.

To address the third challenge, this thesis uses a centralized architecture to store and analyze events. A framework for distributed data retrieval and correlation is perhaps more appealing in larger scaled systems, and left as future work.

This thesis will also discuss and address certain issues related with privacy protection in the two example applications based on application specific properties. Completely addressing this issue, however, is beyond the scope of this document, and identified as future work.

We note that the spatiotemporal event correlation approach specifies the sources of input data and the levels of input data (see Figure 1.1). It puts no requirement about the correlation methods and the types of input data to use. The latter two are thus orthogonal dimensions, and should be decided based on detailed application requirements. In the context of the two applications in this thesis, we focus on statistical correlation methods to eliminate the need of rule specifications by human users or administrators. For the types of input data, this thesis considers mostly homogeneous types of events during the correlation process, but is not limited to them. In fact, we will show that, in both applications, the incorporation of multiple types of events will enhance the effectiveness of attack detection and analysis. Hence the algorithms that we explore are complementary to those that utilize heterogeneous types of input events. The highlighted boxes in Figure 1.1 show the focus of this thesis among the ramifications.

1.3 Two Example Applications

To demonstrate the viability and effectiveness of spatiotemporal event correlation in security, this thesis focuses on two representative security applications: anomaly detection and network forensics. In the application of anomaly detection, we present a prototype system called *Seurat* that can be used to effectively detect attacks that target at multiple hosts by correlating host file system updates across both space and time [114]. In the application of network forensics, this thesis argues that it is important for the network to support automatic forensic analysis abilities *after* an attack has happened. We present such a general framework called *Dragnet* [91], and propose a *random moonwalk* algorithm that determines the origin of epidemic

Application	Anomaly detection	Network forensics
Prototype / Framework	Seurat	Dragnet
Event representation	Feature vectors	Host contact graphs
Correlation algorithm	Feature reduction & clustering	Random moonwalks
Data access	Centralized/future work	Centralized/future work
Privacy protection	Domain specific/Future work	Domain specific/Future work

Figure 1.4: The high level summary of the two example applications.

spreading attacks by exploring the structure of host communication graphs through correlation [117]. Figure 1.4 summarizes how we address the challenges for each application in high level. We introduce both of them briefly next.

1.3.1 Spatiotemporal Event Correlation for Anomaly Detection

Anomaly detection is a widely used approach for detecting attacks in cyber security analysis. Such techniques usually define a model of normal host or network activities. Deviations from the normal model indicate anomalous events and should raise an alarm. Compared with approaches that detect known attacks via pre-defined signatures, anomaly detection identifies new types of attacks that have not been seen before.

Existing techniques of anomaly detection focus on detecting anomalous events based on a normal model built from single host activity patterns. Because these techniques explore only the temporal locality of events occurring on individual hosts, they can potentially generate high false positive rates due to the difficulty of separating malicious events from normal activity changes.

In this thesis, we exploit both the spatial and temporal locality of events in a network system for anomaly detection. We focus on identifying aggregated abnormal events to detect rapidly propagating Internet worms, virus, zombies, or other malicious attacks that compromise multiple hosts in a network system at a time (e.g., one or two days). Once these automated attacks are launched, most of the vulnerable hosts get compromised due to the propagation of the attacks and the scanning preferences of the automated attack tools. Correlating events across multiple hosts can therefore expose malicious activities and reduce those false alarms generated by normal activity pattern changes at individual hosts.

Our prototype system, Seurat, represents events that cause host state transitions as file system updates. The correlation is performed by clustering points, each representing an individual host state transition due to one or more file system changes, in a multi-dimensional feature space. Each feature indicates the change of a file attribute, with all features together describing the host state transitions of an individual machine during a given period (e.g., one day). Over time, the abstraction of point patterns inherently reflects the aggregated host activities. For normal host state changes, the points should follow some regular pattern by roughly falling into several clusters. Abnormal changes, which are hard to detect, or distinguished from single host normal pattern changes by monitoring that host alone, will stand out when they are correlated with other normal host state changes. Such correlation method therefore shares some flavor of *pointillism* – a style of painting that applies small dots onto a surface so that from a distance the dots blend together into meaningful patterns, and we call it the *pointillist* approach.

In this application, the number of file updates at each host daily could be on the order of thousands. Our feature reduction mechanisms can successfully reduce the input data complexity by orders of magnitude. The extensive experiment evaluation shows that Seurat can effectively detect the propagation of well known worms and viruses with a low false alarm rate.

1.3.2 Spatiotemporal Event Correlation for Network Forensics

While end-system based approaches to defend and respond to attacks show promise in the short-term, future attackers are bound to come up with mechanisms that outwit existing signature-based detection and analysis techniques. We believe the Internet architecture should be extended to include auditing mechanisms that enable the forensic analysis of network data, with a goal of identifying the true originator of each attack — even if the attacker recruits innocent hosts as zombies or stepping stones to propagate the attack.

In this thesis, we outline a framework for network forensic analysis called *Dragnet*, with the promise to dramatically change investigations of Internet-based attacks. The key components of this framework are *Attacker Identification* and *Attack Reconstruction*. They together will provide accountability for attacks in both wide area networks and intranets, to deter future attackers.

As a first step toward realizing the Dragnet framework, this thesis focuses on the specific problem of identifying the origin of epidemic spreading attacks such as Internet worms. Our goal is not only to identify the “patient zero” of the epidemic, but also to reconstruct the sequence of events during the initial spread of the attack and identify which communications were the *causal flows* by which one host infected the next. The notion of events in this application is thus denoted by the communication flows between end hosts, and the causal flows correspond to those events that transit host states from normal to infected.

Given such flow notion of events, our correlation algorithm exploits one invariant across all epidemic-style attacks (present and future): for the attack to progress there must be communication among attacker and the associated set of compromised hosts. The communication flows that cause new hosts to become infected form a causal tree, where a causal flow from one computer (the "parent") to its victim (the "child") forms a directed "edge" in this tree. The algorithm works by repeatedly sampling paths on the host communication graph with random walks. Each walk randomly traverses the edges of the graph *backwards* in time, and is called a *random moonwalk*. By correlating these communication events that take place at multiple locations in a network, the overall tree structure of an attack's propagation, especially those initial levels, stands out after repeated random moonwalks to trace back the worm origin. Thus the random moonwalk algorithm can be agnostic to attack signatures or scanning rates, and potentially be applicable to all worm attacks.

In this application, spatiotemporal event correlation enables us to identify abnormal patterns or structures that cannot be identified by existing approaches. Complexity reduction is achieved through both efficient graph representations of communication events and statistical sampling method. We show that the random moonwalk algorithm is both effective and robust. It can detect initial causal flows with high accuracy for both fast propagating worms and a wide variety of stealthy attacks.

1.4 Contributions and Thesis Outline

The main contribution of this thesis is a general solution for more reliably identifying a wide range of attacks, whose activities, when viewed individually, may seem normal or indistinguishable from normal pattern changes, but nevertheless exhibit global abnormal event patterns or structures. This same high level concept of spatiotemporal event correlation guides our problem formulation and algorithm design in both example applications we study. Next, we overview the thesis organization and discuss the specific contributions made in each application.

In Chapter 2, we present Seurat, a prototype system for anomaly detection by correlating file system updates across both space and time. Our primary contributions in Seurat is a novel pointillist approach for detecting aggregated file update events shared across multiple hosts. We start with a binary feature vector space, describing how we define the vectors and reduce feature dimensions for clustering. We then present a more generalized feature vector space to incorporate additional information for detecting more stealthy attacks. We show that despite the large volume of file updates daily, Seurat can effectively detect various attacks and identify only relevant files involved.

In Chapter 3, we present the Dragnet framework for network forensics. We make contributions in problem formulations to perform large scaled *postmortem* forensic analysis, and identify its two key components — *Attacker Identification* and *Attack Reconstruction*. We then discuss in this chapter the required infrastructure support to analyze network traffic across space and time. Our major contribution in Dragnet is the random moonwalk sampling algorithm, which is the first known technique to identify origins of epidemic attacks. This algorithm is extensively evaluated with a wide class of attack scenarios to demonstrate its effectiveness and robustness using analysis, simulation experiments, and real trace study. We show that our algorithm is closely related with spectral analysis, a well known technique for analyzing graph structures. Finally, we present how our algorithm can be elegantly adapted to distributed scenarios.

In Chapter 4, we survey related work in two parts. In the first part, we discuss various other efforts in leveraging distributed information and correlation mechanisms for security. In the second part, we present related work for each specific application we study.

Finally, Chapter 5 summarizes the thesis work, discusses limitations, and outlines future research directions.

Chapter 2

Anomaly Detection Using Spatiotemporal Event Correlation

2.1 Introduction

Anomaly detection, together with misuse detection, are two major categories of intrusion detection methods, aiming at detecting *any set of actions that attempt to compromise the integrity, confidentiality, or availability of information resources* [40].

Anomaly detection approaches assume known knowledge of expected normal host or system states. Activities or changes that are different from the pre-defined normal model are anomalous and can potentially be associated with intrusive attempts. Its objective is thus to determine whether observed events are "normal" or "abnormal". Such methods can detect new types of intrusions that have not been observed before. Compared with this approach, misuse detection approaches capture the distinguishing features of malicious attacks with a signature (e.g., a block of attack code). Intrusions are detected via signature match. Such approach is widely used for detecting known attacks, but cannot catch unknown attacks without a specific signature beforehand, where the efficacy of anomaly detection comes in.

In this thesis, we explore the spatiotemporal event correlation approach for anomaly detection. The idea is to correlate host state transitions defined as file system updates across both space (multiple hosts) and time (past and present), detecting similar coincident changes to the patterns of host state updates that are shared across multiple hosts. Example causes of such coincident events include administrative updates that modify files that have not been modified before, and malware propagations that cause certain log files, which are modified daily, to cease being updated. In both cases, host file system updates exhibit both the temporal and spatial locality that can be exploited by the spatiotemporal event correlation approach.

By exploring both the temporal and spatial locality of host state changes in a network system, spatiotemporal event correlation identifies anomalies *without foreknowledge of normal changes* and *without system-specific knowledge*. Existing approaches focus on the temporal locality of host state transitions, overlooking the spatial locality among different hosts in a network system. They either define a model of normal host state change patterns through learning, or specify detailed rules about normal changes. Learning based approaches train the system to learn characteristics of normal changes. Since they focus only on the temporal locality of single-host state transitions, any significant deviation from the normal model is suspicious and should raise an alarm, possibly resulting in a high false positive rate [26]. Rule-based approaches such as Tripwire [46] require accurate, specific knowledge of system configurations and daily user activity patterns on a specific host. Violation of rules then suggests malicious intrusions. Although rule-based intrusion detection raises fewer false alarms, it requires system administrators to manually specify a set of rules for each host. The correlation capability cross both space and time allows us to learn the patterns of normal state changes over time, and to detect those anomalous events correlated among multiple hosts due to malicious intrusions. This obviates the need for specific rules while eliminating the false alarms caused by single host activity pattern shifts.

The correlation is performed by clustering points, each representing an individual host state transition, in a multi-dimensional feature space. Each feature indicates the change of a file attribute, with all features together describing the host state transitions of an individual machine during a given period (e.g., one day). Over time, the abstraction of point patterns inherently reflects the aggregated host activities. For normal host state changes, the points should follow some regular pattern by roughly falling into several clusters. Abnormal changes, which are hard to detect by monitoring that host alone, will stand out when they are correlated with other normal host state changes. Hence our approach shares some flavor of *pointillism* – a style of painting that applies small dots onto a surface so that from a distance the dots blend together into meaningful patterns.

Figure 2.1 illustrates the pointillist approach to anomaly detection. There are five hosts in the network system. We represent state changes on each host daily as a point in a 2-dimensional space in this example. On normal days, the points roughly fall into the dash-circled region. The appearance of a new cluster consisting of three points (indicated by the solid circle) suggests the incidence of an anomaly on host A, B, and D, which may all have been compromised by the same attack. Furthermore, if we know that certain hosts (e.g., host A) are already compromised (possibly detected by other means such as a network based IDS), then we can correlate the state changes of the compromised hosts with the state changes of all other hosts in the network system to detect more infected hosts (e.g., host B and D).

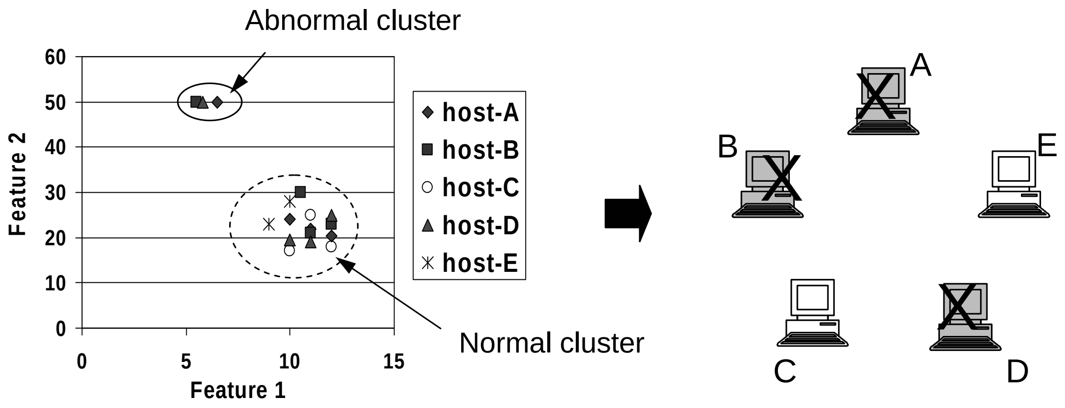


Figure 2.1: Pointillist approach to anomaly detection. Normal points are clustered by the dashed circle. The appearance of a new cluster consisting of three points suggests anomalous events on host A, B, and D.

We have implemented a prototype system, called *Seurat* [1], that uses file system updates to represent host state changes for anomaly detection. Seurat successfully detects the propagation of a manually launched Linux worm and a list of well known Windows worms and viruses on a number of hosts in an isolated cluster. Seurat has a low false alarm rate when evaluated by a real deployment in both Linux and Windows systems. These alarms are caused by either administrative updates or network wide experiments. The false negative rate and detection latency, evaluated with both simulated attacks and real attacks, are low for fast propagating attacks. For slowly propagating attacks, there is a tradeoff between false negative rate and detection latency. For each alarm, Seurat identifies the list of hosts involved and the related files, which we expect will be extremely helpful for system administrators to examine the root cause and dismiss false alarms.

The rest of this chapter is organized as follows: Section 2.2 describes the Seurat threat model. Section 2.3 introduces the algorithm for correlating host file system updates across both space and time. Section 2.4 evaluates the pointillist approach. Section 2.5 describes an extension of Seurat to detect stealthy attacks by exploiting more features of file updates. Section 2.6 discusses the limitations of our system and Section 2.7 suggests possible improvements for future work.

[1] Seurat is the 19th century founder of pointillism.

2.2 Attack Model

In this thesis, we focus on anomaly detection in a single network system. A *network system* is a collection of host computers connected by a network in a single administrative domain. We note that our technical approach of correlation is not limited by the administrative domain boundaries. However, issues of trust and privacy may raise concerns regarding distributed data collection and querying, which we will not address completely in this thesis. For this reason, we limit our discussion to hosts inside a single administrative domain and adopt a centralized architecture for event storage and analysis.

The goal of Seurat is to automatically identify anomalous events by correlating the state change events of all hosts in a network system. Hence Seurat defines an anomalous event as an unexpected state change close in time across *multiple* hosts in a network system.

We focus on rapidly propagating Internet worms, viruses, or other malicious attacks that compromise multiple hosts in a network system at a time (e.g., one or two days). We have observed that, once fast, automated attacks are launched, most of the vulnerable hosts get compromised due to the rapid propagation of the attack and the scanning preferences of the automated attack tools. According to CERT's analysis [20], the level of automation in attack tools continues to increase, making it faster to search vulnerable hosts and propagate attacks. Recently, the Slammer worm hit 90 percent of vulnerable systems in the Internet within 10 minutes [63]. Worse, the lack of diversity in systems and software run by Internet-attached hosts enables massive and fast attacks. Computer clusters tend to be configured with the same operating systems and software. In such systems, host state changes due to attacks have strong temporal and spatial locality that can be exploited by Seurat.

Although Seurat will more effectively detect system changes due to fast propagating attacks, it can be generalized to detect slowly propagating attacks as well. This can be done by varying the time resolution of reporting and correlating the collective host state changes. We will discuss this issue further in Section 2.6. However, Seurat's global correlation cannot detect abnormal state changes that are unique to only a single host in the network system.

Seurat represents events that cause host state changes using *file system updates.* [77] found that 83% of the intrusion tools and network worms they surveyed modify one or more system files. These modifications would be noticed by monitoring file system updates. There are many security tools such as Tripwire [46] and AIDE [57] that rely on monitoring abnormal file system updates for intrusion detection.

We use the file name, including its complete path, to identify a file in the network system. We regard different instances of a file that correspond to a common path name as a same file across different hosts, since we are mostly interested in system files which tend to have canonical path names exploited by malicious attacks. We treat files with different path names on different hosts as different files, even when they are identical in content.

For the detection of anomalies caused by attacks, we have found that this representation of host state changes is effective and useful. However, we may need different approaches for other applications of Seurat such as file sharing detection, or for the detection of more sophisticated future attacks that alter files at arbitrary locations as they propagate. For example, we can investigate the use of file content digests instead of file names as future work.

2.3 Correlation-based Anomaly Detection

We define a d-dimensional feature vector $H_{ij} = \langle v_1, v_2, \ldots, v_d \rangle$ to represent the file system update attributes for host i during time period j. Each H_{ij} can be plotted as a point in a d-dimensional feature space. Our pointillist approach is based on correlating the feature vectors by clustering. Over time, for normal file updates, the points follow some regular pattern (e.g., roughly fall into several clusters). From time to time, Seurat compares the newly generated points against points from previous time periods. The appearance of a new cluster, consisting only of newly generated points, indicates abnormal file updates and Seurat raises an alarm.

In the rest of this section, we first present how we define the feature vector space and the distances among points. We then describe the methods Seurat uses to reduce feature vector dimensions for clustering to work most effectively. Finally, we discuss how Seurat detects abnormal file updates by clustering.

2.3.1 A Binary Feature Vector Space

Many attacks install new files on a compromised host, or modify files that are infrequently updated. Various information such as host file updates, file update times, and file size changes can be used as indicators of the existence of those attacks. To simplify exposition, we describe binary feature vectors for representing host file updates first. We will then explore a more general feature vector space that utilizes other type of information Seurat collects in Section 2.5.

Each dimension in the binary feature vector space corresponds to a unique file (indexed by the full-path file name). As such, the dimension d of the space is the number of file names present on any machine in the network system. We define the *detection window* to be the period that we are interested in finding anomalies.

$$\begin{array}{cccccccccc} & & & & F_1 & F_2 & F_3 & F_4 & F_5 & \\ V_1 = & H_{11} = & < & & 1, & 1, & 0, & 1, & 1 & > \\ V_2 = & H_{21} = & < & & 1, & 1, & 1, & 0, & 0 & > \\ V_3 = & H_{12} = & < & & 1, & 1, & 0, & 1, & 0 & > \end{array}$$

Figure 2.2: Representing host file updates as feature vectors. F_1, F_2, F_3, F_4, F_5 are five different files (i.e., file names). Accordingly, the feature vector space has 5 dimensions in the example.

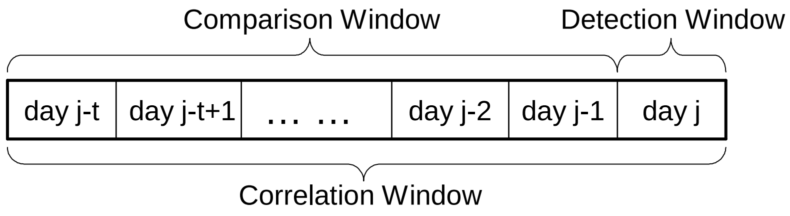


Figure 2.3: Detection window, comparison window, and correlation window. The detection window is day j. The comparison window is from day $j-t$ to day $j-1$. The correlation window is from day $j-t$ to day j.

In the current prototype, the detection window is one day. For each vector $H_{ij} = \langle v_1, v_2, \ldots, v_d \rangle$, we set v_k to 1 if host i has updated (added, modified, or removed) the k-th file on day j, otherwise, we set v_k to 0.

The vectors generated in the detection window will be correlated with vectors generated on multiple previous days. We treat the generated feature vectors as a set of independent points. The set can include vectors generated by the same host on multiple days, and vectors generated by multiple hosts on the same day. In the rest of the chapter, we use $V = \langle v_1, v_2, \ldots, v_d \rangle$ to denote a feature vector for convenience. Figure 2.2 shows how we represent the host file updates using feature vectors.

The correlation is based on the distances among vectors. Seurat uses a cosine distance metric, which is a common similarity measure between binary vectors [10, 45]. We define the distance $D(V_1, V_2)$ between two vectors V_1 and V_2 as their angle θ computed by the cosine value:

$$D(V_1, V_2) = \theta = cos^{-1}\left(\frac{V_1 \cdot V_2}{|V_1||V_2|}\right)$$

For each day j (the detection window), Seurat correlates the newly generated vectors with vectors generated in a number of previous days $j-1, j-2, \ldots$. We define the abnormal file update events on day j as the file update patterns that have not occurred on previous days. We define the *comparison window* of day j as the days that we look back for comparison, and the *correlation window* of day j as the inclusive period of day j and its comparison window. Vectors generated outside the correlation window of day j are not used to identify abnormal file updates on day j. Figure 2.3 illustrates the concepts of detection window, comparison window, and correlation window.

Since each vector generated during the comparison window serves as an example of normal file updates to compare against in the clustering process, we explore the temporal locality of normal update events by choosing an appropriate comparison window for each day. The comparison window size is a configurable parameter of Seurat. It reflects how far we look back into history to implicitly define the model of normal file updates. For example, some files such as `/var/spool/anacron/cron.weekly` on Linux platforms are updated weekly. In order to regard such weekly updates as normal updates, administrators have to choose a comparison window size larger than a week. Similarly, the size of the detection window reflects the degree of temporal locality of abnormal update events.

Since Seurat correlates file updates across multiple hosts, we are interested in only those files that have been updated by at least two different hosts. Files that have been updated by only one single host in the network system throughout the correlation window are more likely to be user files. As such, we do not select them as relevant dimensions to define the feature vector space.

2.3.2 Feature Selection

Most file updates are irrelevant to anomalous events even after we filter out the file updates reported by only a single host. Those files become noise dimensions when we correlate the vectors (points) to identify abnormal updates, and increase the complexity of the correlation process. We need more selective ways to choose relevant files and reduce feature vector dimensions. We have implemented two methods for this purpose: (1) *wavelet-based selection*, and (2) *principal component analysis* (PCA).

Wavelet-based Selection

The wavelet-based selection method regards each individual file update status as a discrete time series signal S. Given a file i, the value of the signal on day n, denoted by $S_i(n)$, is defined as the total number of hosts that update file i on day n in the network system. Each such signal S_i can be decomposed into a low frequency signal cA_i reflecting the long term update trend, and a high frequency signal cD_i

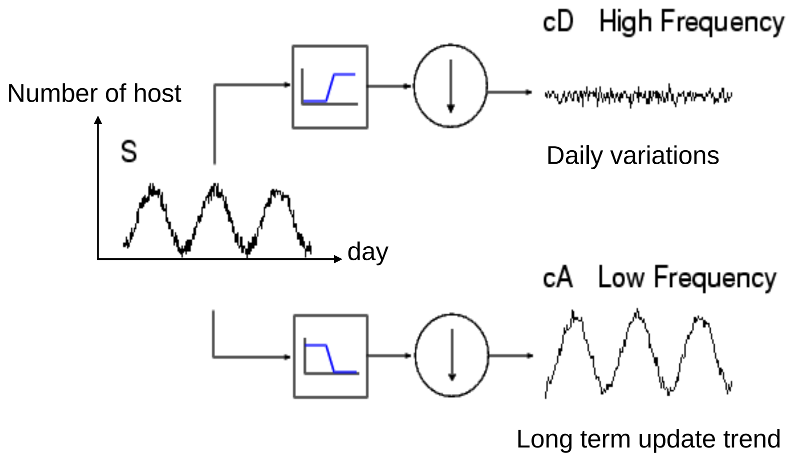


Figure 2.4: Representing file update status with wavelet transformation. The original signal is S, which can be decomposed into a low frequency signal cA reflecting the long term update trend, and a high frequency signal cD reflecting the daily variations from the long-term trend.

reflecting the day-to-day variation from the long term trend (see Figure 2.4). If the high frequency signal cD_i shows a spike or a dip on a certain day, we know that a significantly larger or smaller number of hosts updated file i than on a normal day, respectively. We then select file i as a relevant feature dimension in defining the feature vector space.

Seurat detects signal spikes and dips using the residual signal of the long-term trend. The similar technique has been used to detect disease outbreaks [119] and network traffic anomalies [7]. To detect anomalies on day j, the algorithm takes as input the list of files that have been updated by at least two different hosts in the correlation window of day j. Then, from these files the algorithm selects a subset that will be used to define the feature vector space.

Figure 2.5 shows the steps to select features by wavelet-based method. Given a fixed correlation window of day j, the algorithm starts with constructing a time series signal S_i for each file i, and decomposes S_i into cA_i and cD_i using a single-level Daubechies wavelet transformation as described. Then we compute the residual signal value $R_i(j)$ of day j by subtracting the trend value $cA_i(j-1)$ of day $j-1$ from the original signal value $S_i(j)$ of day j. If $|R_i(j)|$ exceeds a preset threshold α, then the actual number of hosts who have updated file i on day j is significantly larger or smaller than the prediction $cA_i(j-1)$ based on the long term trend. Therefore, Seurat selects file i as an interesting feature dimension for anomaly detection on day j. As an example, Figure 2.6 shows the original signal and the residual signal of a file using a 32-day correlation window in a 22-host teaching cluster. Note the threshold

```
For each file $i$:

  1. Construct a time series signal:
     $S_i = cA_i + cD_i$

  2. Compute the residual signal value of day $j$:
     $R_i(j) = S_i(j) - cA_i(j-1)$

  3. if $|R_i(j)| > \alpha$, then select file $i$ as a feature dimension.
```

Figure 2.5: Wavelet-based feature selection.

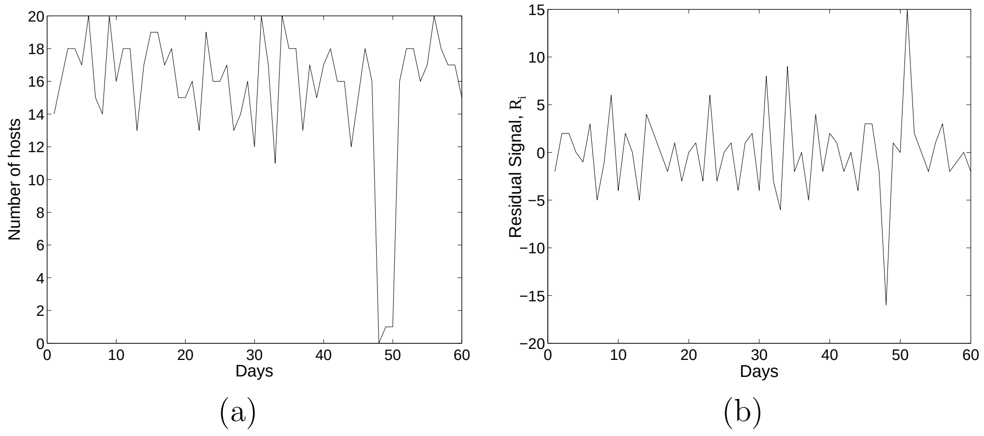


Figure 2.6: Wavelet transformation of file update status. (a) The original signal of the file update status (b) The residual signal after wavelet transformation

value α of each file is a parameter selected based on the statistical distribution of historical residual values.

PCA-based Dimension Reduction

PCA is a statistical method to reduce data dimensionality without much loss of information [43]. Given a set of d-dimensional data points, PCA finds a set of d'-dimensional vectors, called *principal components*, that account for the variance of the input data as much as possible. Dimensionality reduction is achieved by projecting the original d-dimensional data onto the subspace spanned by these d' orthogonal vectors. Most of the intrinsic information of the d-dimensional data is preserved in the subspace.

We note that the updates of different files are usually correlated. For example, when a software package is updated on a host, many of the related files will be modified together. Thus we can perform PCA to identify the correlation of file updates.

Given a d-dimensional feature space $\mathcal{Z}_2^d$, and a list of m feature vectors $V_1, V_2, \ldots, V_m \in \mathcal{Z}_2^d$, we perform the following steps using PCA to obtain a new list of feature vectors $V_1', V_2', \ldots, V_m' \in \mathcal{Z}_2^{d'} (d' < d)$ with reduced number of dimensions:

1. Standardize each feature vector $V_k = \langle v_{1k}, v_{2k}, \ldots, v_{dk} \rangle$ $(1 \leq k \leq m)$ by subtracting each of its elements v_{ik} by the mean value of the corresponding dimension $u_i (1 \leq i \leq d)$. We use $\overline{V}_k = \langle \overline{v}_{1k}, \overline{v}_{2k}, \ldots, \overline{v}_{dk} \rangle \in \mathcal{Z}_2^d$ to denote the standardized vector for the original feature vector V_k. Then,

$$\overline{v}_{ik} = v_{ik} - u_i \text{ (where } u_i = \frac{\sum_{j=1}^{m} v_{ij}}{m}, 1 \leq i \leq d)$$

2. Use the standardized feature vectors $\overline{V}_1, \overline{V}_2, \ldots, \overline{V}_m$ as input data to PCA in order to identify a set of principal components that are orthogonal vectors defining a set of transformed dimensions of the original feature space $\mathcal{Z}_2^d$. Select the first d' principal components that account for most of the input data variances (e.g., 90% of data variances) to define a subspace $\mathcal{Z}_2^{d'}$.

3. Project each standardized feature vector $\overline{V}_k \in \mathcal{Z}_2^d$ onto the PCA selected subspace $\mathcal{Z}_2^{d'}$ to obtain the corresponding reduced dimension vector $V_k' \in \mathcal{Z}_2^{d'}$.

Note that PCA is complementary to wavelet-based selection. Once we fix the correlation window of a particular day, we first pick a set of files to define the feature vector space by wavelet-based selection. We then perform PCA to reduce the data dimensionality further.

2.3.3 Anomaly Detection by Clustering

Once we obtain a list of transformed feature vectors using feature selection, we cluster the vectors based on the distance between every pair of them.

We call the cluster a *new cluster* if it consists of multiple vectors only from the detection window. The appearance of a new cluster indicates a change in file update patterns occurred during the detection window and should raise an alarm.

There are many existing algorithms for clustering, for example, K-means [33, 34] or Single Linkage Hierarchical Clustering [45]. Seurat uses a simple iterative algorithm, which is a common method for K-means initialization, to cluster vectors without prior knowledge of the number of clusters [62]. The algorithm assumes each cluster has a hub. A vector belongs to the cluster whose hub is closest to that vector compared with the distances from other hubs to that vector. The algorithm starts with one cluster whose hub is randomly chosen. Then, it iteratively selects a vector that has the largest distance to its own hub as a new hub, and re-clusters all the vectors based on their distances to all the selected hubs. This process continues until there is no vector whose distance to its hub is larger than the half of the average hub-hub distance.

We choose this simple iterative algorithm because it runs much faster, and works equally well as the Single Linkage Hierarchical algorithm in our experiments. The reason that even the simple clustering algorithm works well is that the ratio of inter-cluster distance to intra-cluster distance significantly increases after feature selection.

Once we detect a new cluster and generate an alarm, we can further identify the involved hosts and the files from which the cluster resulted. The suspicious hosts are just the ones whose file updates correspond to the feature vectors in the new cluster. To determine which files possibly cause the alarm, we only focus on the files picked by the wavelet-based selection to define the feature vector space. For each of those files, if it is updated by all the hosts in the new cluster during the detection window, but has not been updated by any host during the corresponding comparison window, Seurat outputs this file as a candidate file. Similarly, Seurat also reports the set of files that have been updated during the comparison window, but are not updated by any host in the new cluster during the detection window.

Based on the suspicious hosts and the selected files for explaining root causes, system administrators can decide whether the updates are known administrative updates that should be suppressed, or some abnormal events that should be further investigated. If the updates are caused by malicious attacks, administrators can take remedial counter measures for the new cluster. Furthermore, additional compromised hosts can be identified by checking if the new cluster expands later and if other hosts have updated the same set of candidate files. Note that the alarms due to administrative updates can be suppressed if the administrator provides Seurat with the list of involved files beforehand.

2.4 Experiments

We have developed a multi-platform (Linux and Windows) prototype of Seurat that consists of a lightweight data collection tool and a correlation module. The data collection tool scans the file system of the host where it is running and generates a daily summary of file update attributes. Seurat harvests the summary reports from multiple hosts in a network system and the correlation module uses the reports for anomaly detection.

We have installed the Seurat data collection tool on a number of campus office machines and a teaching cluster that are used by students daily. By default, the tool scans the attributes of all system files on a host. The attributes of a file include the file name, type, device number, permissions, size, inode number, important timestamps, and a 16-byte MD5 checksum of file content. Each day, each host compares the newly scanned disk snapshot against that from the previous day and generates a file update summary report. In the current prototype, all the reports are uploaded daily to a centralized server where system administrators can monitor and correlate the file updates using the correlation module.

In this section, we study the effectiveness of Seurat's pointillist approach for detecting aggregated anomalous events. Using the daily file update reports from our real deployment, we study the false positive rate and the corresponding causes in Section 2.4.1 and Section 2.4.2. We evaluate the false negative rate with simulated attacks in Section 2.4.3. In order to verify the effectiveness of our approach on real malicious attacks, we launched real worms and viruses into isolated computer clusters. We report the results in Section 2.4.4 and Section 2.4.5.

2.4.1 False Positives on Linux Platforms

The best way to study the effectiveness of our approach is to test it with real data. We have deployed the Seurat data collection tool on both Linux and Windows hosts, and correlate the reports for daily anomaly detection. The results from both deployments are similar. We first illustrate the details of the experiment and the results with the data from our Linux host deployment, where we have longer period of data collection than for our Windows deployment. Then, we describe the results of the Windows host deployment briefly in Section 2.4.2.

We deployed Seurat on a teaching cluster of 22 Linux hosts and have been collecting the daily file update reports since Nov 2003. The teaching cluster is mostly used by students for their programming assignments. The cluster is also occasionally used by a few graduate students for running network experiments. For privacy protection, personal files under user home directories are not scanned for Linux platforms.

In this experiment, we use the file update reports from Dec 1, 2003 until Feb 29, 2004 to evaluate the false positive rate. During this period, there are a few days when a couple of hosts failed to generate or upload reports due to system failure or reconfigurations. For those small number of missing reports, we simply ignore them because they do not affect the aggregated file update patterns.

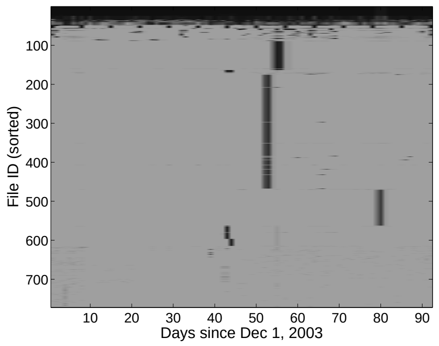
We set the correlation window to 32 days in order to accommodate monthly file update patterns. That is, we correlate the update pattern from day 1 to day 32 to identify abnormal events on day 32, and correlate the update pattern from day 2 to day 33 to detect anomalies on day 33, etc. Thus, our detection starts from Jan 1, 2004, since we do not have 32-day correlation windows for the days in Dec 2003.

Dimension Reduction

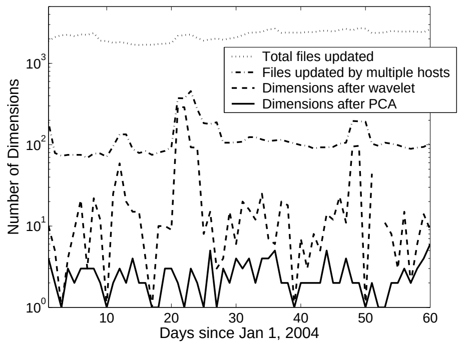
Once we fixed the correlation window of a particular day, we identify relevant files using wavelet-based selection with a constant threshold $\alpha = 2$ to define the feature vector space for simplicity. We then perform PCA to reduce the data dimensionality further by picking the first several principal components that account for 98% of the input data variance.

Throughout the entire period of 91 days, 772 files with unique file names were updated by at least two different hosts. Figure 2.7 (a) shows the number of hosts that updated each file during the data collection period. We observe that only a small number files (e.g.,`/var/adm/syslog/mail.log`) are updated regularly by all of the hosts, while most other files (e.g., `/var/run/named.pid`) are updated irregularly, depending on the system usage or the applications running.

Figure 2.7 (b) shows the results of feature selection. There were, on average, over 2000 files updated by hosts in the cluster during each correlation window (i.e., one day). Among those files, there were on average 140 files updated by at least two different hosts daily. After wavelet-based selection, the average number of feature dimensions is 17. PCA further reduces the vector space dimension to an average of 2. Overall, we achieved about 3 orders of magnitude dimensionality reduction.



(a)



(b)

Figure 2.7: Feature selection and dimension reduction. (a) File update patterns. Files are sorted by the cumulative number of hosts that have updated them throughout the 91 days. The darker the color is, the more hosts updated the corresponding file. (b) The number of feature vector dimensions after wavelet-based selection and PCA consecutively.

False Alarms

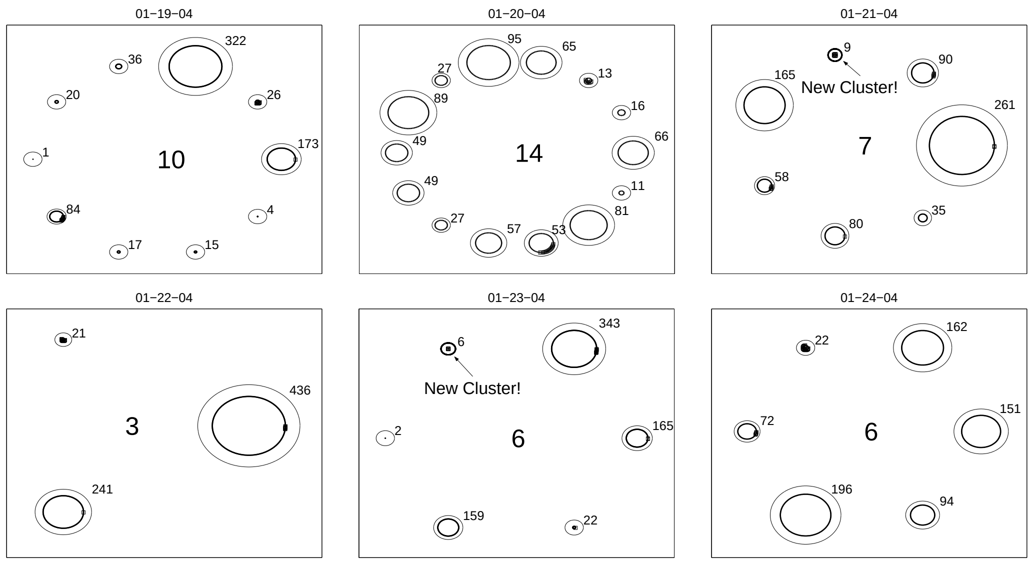


Figure 2.8: Clustering feature vectors for anomaly detection at Linux platforms. Each circle represents a cluster. The number at the center of the figure shows the total number of clusters. The radius of a circle corresponds to the number of points in the cluster, which is also indicated beside the circle. The squared dots correspond to the new points generated on the day under detection. New clusters are identified by a thicker circle.

After dimension reduction, we perform clustering of feature vectors and identify new clusters for each day. Figure 2.8 illustrates the clustering results of 6 consecutive days from Jan 19, 2004 to Jan 24, 2004. There are two new clusters identified on Jan 21 and Jan 23, which involve 9 hosts and 6 hosts, respectively. Since Seurat outputs a list of suspicious files as the cause of each alarm, system administrators can tell if the new clusters are caused by malicious intrusions.

Based on the list of files output by Seurat, we can figure out that the new clusters on Jan 21 and Jan 23 reflect large scale file updates due to a system reconfiguration at the beginning of the spring semester. For both days, Seurat accurately pinpoints the exact hosts that are involved. The reconfiguration started from Jan 21, when a large number of binaries, header files, and library files were modified on 9 out of the 22 hosts. Since the events are known to system administrators, we treat the identified vectors as normal for future anomaly detection. Thus, no alarm is triggered on Jan 22, when the same set of library files were modified on 12 other hosts. On Jan 23, the reconfiguration continued to remove a set of printer files on 6 out of the 22 hosts.

Again, administrators can mark this event as normal and we spot no new cluster on Jan 24, when 14 other hosts underwent the same set of file updates.

In total, Seurat raises alarms on 9 out of the 60 days under detection, among which 6 were due to system reconfigurations. Since the system administrators are aware of such events in advance, they can simply suppress these alarms. The 3 other alarms are generated on 3 consecutive days when a graduate student performed a network experiment that involved simultaneous file updates at multiple hosts. Such events are rare, and should alert the system administrators.

2.4.2 False Positives on Microsoft Windows Platforms

We installed the Seurat data collection tool on 15 Microsoft Windows XP machines that are office desktops used by volunteering faculty and graduate students in a campus network. We have collected file update reports daily from these machines since September 2004.

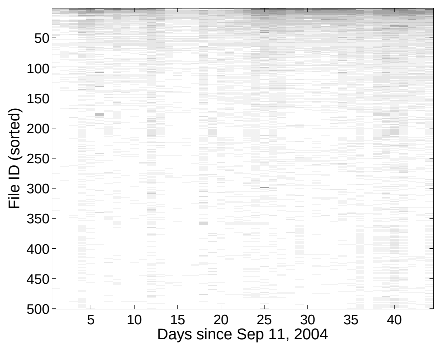
Since there are no default user home directories for Windows platforms, all the files in the `C:\` drive are scanned, including the `C:\Documents and Settings\` directory where user related Windows registry files (which are more likely to be modified by an attack) reside. Note each Windows user will have a common set of similar directories and files under the directory `C:\Documents and Settings\userid\` where “userid” is the user’s Windows login ID. In such cases, we convert the “userid” in the path name to a generic user ID in order to both correlate the changes of user-related Windows registry files and preserve user privacy. Consequently, we may have more unique files updated across multiple machines to perform Seurat correlation than the case where we only look at system file changes.

For this experiment, we use the file update reports from Sep 10, 2004 until Oct 25,2004, when the majority of the volunteered machines are running the Seurat data collection tool. Similar to the data collected from our Linux deployment, there are a few days in this period when a couple of hosts failed to upload reports because their users turned off the machines or temporarily disabled the Seurat data collection tool for running their own timing-critical tasks.

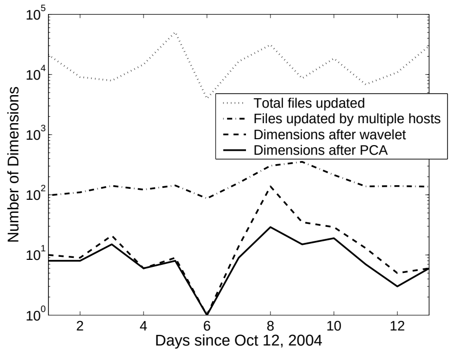
We again set the correlation window size to 32 days. As such, our anomaly detection starts from Oct 11, 2004 and lasts for 15 days.

Dimension Reduction

Throughout the entire period of 46 days, 18,072 files with unique file names (after converting user IDs to a generic user ID) were updated by at least two different hosts on Windows platforms. We pick the 500 most frequently updated files out of the total



(a)



(b)

Figure 2.9: Feature selection and dimension reduction at Microsoft Windows Platforms. (a) File update patterns. Files are sorted by the cumulative number of hosts that have updated them throughout the 46 days. The darker the color is, the more hosts updated the corresponding file. (b) The number of feature vector dimensions after wavelet-based selection and PCA consecutively.

18,072 files, and Figure 2.9 (a) shows the number of hosts that updated each such selected file during the data collection period. Compared with the data from our Linux deployment, the Windows file update patterns are less regular. One reason is that the monitored machines are under different administrative subdomains, unlike the Linux teaching cluster where the machines have more homogeneous configurations. Thus, the locations of non-standardized applications in the file system could be different across different machines. We found that only a small number of log files are updated regularly by most of the hosts (e.g., `C:\WINDOWS\security\logs\winlogon.log`). However, the majority of file update patterns are irregular, depending on the actually launched applications and how the system is being used.

Figure 2.9 (b) shows the Seurat feature selection results on the Windows machines during the 15 days of anomaly detection. Although we observe significantly more daily file updates on participating Windows hosts than Linux hosts, the average number of files that are updated by at least two different hosts during each correlation window is 165, similar to the Linux hosts. Wavelet-based selection reduces the average number of feature dimensions to 22, and PCA further reduces the number of dimensions to an average of 10 every day.

False Alarms

Seurat raised one alarm during the 15 days of anomaly detection from Oct 11, 2004 to Oct 25, 2004. The alarm was raised on Oct 18, 2004, when two new clusters were identified with each cluster involving two participating hosts. The lists of suspicious file changes reported by Seurat suggest that both new clusters were caused by Windows Update[2]. In one new cluster, the set of files identified are all located under the directory `C:\Program Files\WindowsUpdate`. For the other new cluster, Seurat output 30 files which had not been updated by any host before but were updated by the two corresponding hosts in the new cluster. Most of these identified files are `dll` library files under the directory `C:\WINDOWS\system32`, suggesting that the file modification events might be related to system updates. After talking to the owners of the two involved hosts, we confirmed that the detected machines indeed installed a Windows update package called "Cumulative Security Update for Internet Explorer for Windows XP Service Pack 2 (KB834707)" on the same day that Seurat spotted the new cluster.

2.4.3 False Negatives

The primary goal of this experiment is to study the false negative rate and detection latency of Seurat as the stealthiness of the attack changes. We use simulated attacks by manually updating files on the selected host reports, as if they were infected.

We first examine the detection rate of Seurat by varying the degree of attack aggressiveness. We model the attack propagation speed as the number of hosts infected on each day (the detection window), and model the attack stealthiness on a local host as the number of new files installed by this attack. Our simulation runs on the same teaching cluster that we described in Section 2.4.1. Since the aggregated file update patterns are different for each day, we randomly pick ten days in Feb 2004, when there was no intrusion. On each selected day, we simulate attacks by manually inserting artificial new files into a number of host reports on only that day, and use the modified reports as input for detection algorithm. We then remove those modified entries, and repeat the experiments with another day. The detection rate is calculated as the number of days that Seurat spots new clusters over the total ten days.

Figure 2.10 shows the detection rate of Seurat by varying the number of files inserted on each host and the number of hosts infected. On one hand, the detection rate monotonically increases as we increase the number of files inserted on each host by an attack. Since the inserted files do not exist before, each of them will be selected as a feature dimension by the wavelet-based selection, leading to larger distances between the points of infected host state changes and the points of normal host state changes. Therefore, the more new files are injected by an attack, the higher the detection rate gets. On the other hand, as we increase the number of infected hosts,

[2]More information on Windows Update can be found at http://windowsupdate.microsoft.com and http://go.microsoft.com/fwlink?linkid=23699

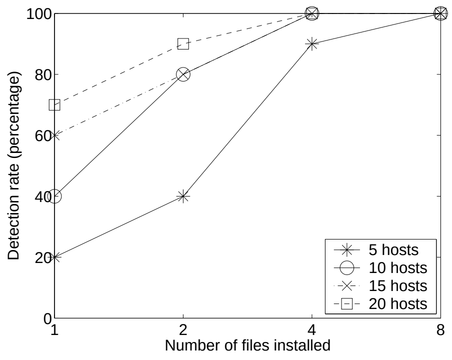


Figure 2.10: Detection rate of simulated attacks. We vary the number of hosts infected and the number of files inserted on each host by the simulated attacks.

Worms	Adore	Ramen-A	Ramen-B	Slapper-A	Slapper-B	Kork
Files modified	10	8	12	3	4	5
2 infected hosts	80%	80%	90%	30%	40%	30%
4 infected hosts	100%	100%	90%	70%	80%	70%
8 infected hosts	100%	100%	100%	100%	100%	100%

Figure 2.11: Detection rate of emulated worms. Percentage of experiments when Seurat detects the worms. For each experiment, we launch a worm on a different day. We vary the number of hosts compromised by the attacks and the type of worms.

the number of points for abnormal host state changes becomes large enough to create an independent new cluster. Thus, rapidly propagating attacks are more likely to be caught. Accordingly, detecting a slowly propagating attack requires a larger detection window, hence longer detection latency, in order to accumulate enough infected hosts. We revisit this issue in Section 2.6.

We further evaluate the detection rate of Seurat on six Linux worms with simulated attacks. To do so, we compile a subset of files modified by each worm based on the descriptions from public Web sites such as Symantec [99] and F-Secure information center [30]. We then manually modify the described files in a number of selected host reports to simulate the corresponding worm attacks. Again, for each worm, we vary the number of infected hosts, and run our experiments on the teaching cluster with ten randomly selected days.

Table 2.11 shows the number of files modified by each worm and the detection rate of Seurat. Note that we measured the detection rate by counting the number of days that Seurat spots new clusters over the total ten days. In general, the more files modified by a worm, the more likely the worm will be detected. But the position of a file in the file system directory tree also matters. For example, both Slapper-B worm and Kork worm insert 4 new files into a compromised host. However, Kork worm additionally modifies `/etc/passwd` to create accounts with root privileges. Because there are many hosts that have updated `/etc/passwd` during a series of system reconfiguration events, the inclusion of such files in the feature vector space reduces the distances from abnormal points to normal points, resulting in higher false negative rates. We discuss this further in Section 2.6.

2.4.4 Real Attacks on Linux Platforms

Now we proceed to examine the efficacy of Seurat during a real worm outbreak. The best way to show this would be to have Seurat detect an anomaly caused by a new worm propagation. Instead of waiting for a new worm's outbreak, we have set up an isolated computer cluster where, without damaging the real network, we can launch worms and record file system changes. This way, we have full control over the number of hosts infected, and can repeat the experiments. Because the isolated cluster has no real users, we merge the data acquired from the isolated cluster with the data we have collected from the Linux teaching cluster in order to conduct experiments.

We obtained the binaries and source codes of a few popular worms from public Web sites such as whitehats [111] and packetstorm [72]. Extensively testing Seurat, with various real worms in the isolated cluster, requires tremendous effort in setting up each host with the right versions of vulnerable software. As such, we limit our attention to one worm for illustrative purposes and present the result with the Lion worm [86] in this experiment.

The Lion worm was found in early 2001. Lion exploits a vulnerability of BIND 8.2, 8.2-P1, 8.2.1, 8.2.2-Px. Once Lion infects a system, it sets up backdoors, leaks out confidential information (`/etc/passwd, /etc/shadow`) via email, and scans the Internet to recruit vulnerable systems. Lion scans the network by randomly picking the first 16 bits of an IP address, and then sequentially probing all the 2^{16} IP addresses in the space of the block. After that, Lion randomly selects another such address block to continue scanning. As a result, once a host is infected by Lion, all the vulnerable hosts nearby (in the same IP address block) will be infected soon. Lion affects file systems: the worm puts related binaries and shell scripts under the `/dev/.lib` directory, copies itself into the `/tmp` directory, changes system files under the `/etc` directory, and tries to wipe out some log files.

We configured the isolated cluster with three Lion-vulnerable hosts and one additional machine that launched the worm. The vulnerable machines were running RedHat 6.2 including the vulnerable BIND 8.2.2-P5. The cluster used one C class network address block. Every machine in the cluster was connected to a 100Mbps Ethernet and was running a DNS server (`named`) of the BIND distribution as a caching-only name server. This setup is similar to the Linux teaching cluster described in Section 2.4.1, where all the hosts are running `named` as caching-only servers for efficient domain name lookup. The Seurat data collection tool generated a file system update report on every machine daily.

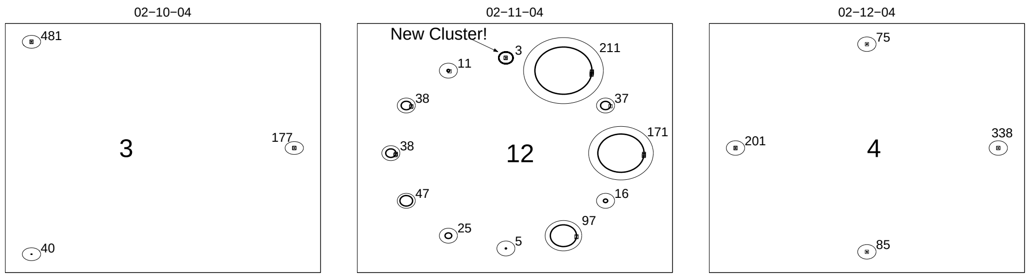


Figure 2.12: Intrusion detection by Seurat. Seurat identified a new cluster of three hosts on Feb 11, 2004, when we manually launched the Lion worm. The number of clusters formed in each day varies due to an artifact of the feature vector selection and the clustering algorithm.

After we launched the Lion worm, all three vulnerable hosts in the isolated cluster were infected quickly one after another. We merge the file update report by the each compromised host with a different normal host report generated on Feb 11, 2004, when we know there was no anomaly. Figure 2.12 shows the clustering results of three consecutive days from Feb 10, 2004 to Feb 12, 2004 using the merged reports.

On the attack day, there are 64 files picked by the wavelet-based selection. The number of feature dimensions is reduced to 9 after PCA. Seurat successfully detects a new cluster consisting of the 3 infected hosts. Figure 2.13 lists the 22 files selected by Seurat as the causes of the alarm. These files provide enough hints to the administrators to confirm the existence of the Lion worm. Once detected, these compromised hosts as well as the list of suspicious files can be marked for future detection. If, in the following days, there are more hosts that are clustered together with the already infected machines, or experience the same file updates, then we may conclude they are infected by the same attack.

2.4.5 Real Attacks on Microsoft Windows Platforms

In recent years, the number of worms and viruses targeting Windows platforms has increased dramatically. The Symantec company reported 4,496 new instances of malicious code exploiting vulnerabilities of the Microsoft Windows systems during the

File ID.	File name	File ID.	File name
1	`/sbin/asp`	12	`/var/spool/mail`
2	`/dev/.lib`	13	`/dev/.lib/bindx.sh`
3	`/dev/.lib/star.sh`	14	`/tmp/ramen.tgz`
4	`/var/spool/mail/root`	15	`/dev/.lib/scan.sh`
5	`/dev/.lib/bind`	16	`/dev/.lib/pscan`
6	`/etc/hosts.deny`	17	`/var/spool/mqueue`
7	`/dev/.lib/randb`	18	`/dev/.lib/hack.sh`
8	`/sbin`	19	`/dev/.lib/.hack`
9	`/var/log`	20	`/dev/.lib/index.html`
10	`/dev/.lib/bindname.log`	21	`/dev/.lib/asp62`
11	`/dev/.lib/index.htm`	22	`/var/log/sendmail.st`

Figure 2.13: Suspicious files for the new cluster on Feb 11, 2004.

first six months of 2004 only [100]. In this section, we examine the effectiveness of Seurat in detecting worm/virus infections on Windows systems.

We obtained the executables of recently found Windows worms and viruses, manually launched them in an isolated computer cluster of 6 vulnerable, unpatched Windows XP hosts inside one C class network address space. The 6 Windows XP hosts are virtual machines deployed over 3 physical machines connected to a 100Mbps Ethernet running VMWare. For the scanning worms (i.e., Blaster worm), we launched them from a seed machine and let them propagate. For viruses that need human users to activate them, we emulated the virus propagations by manually clicking the virus-infected programs or documents at each host (i.e., mass-mailing viruses such as LoveLetter, NetSky, Bagle). The Seurat data collection tool then reported the file system updates on every Windows virtual machine daily. We merged the reported file changes with the reports from our real Windows system deployment described in Section 2.4.2.

In order to examine the performance with different attack propagation rates, we again vary the number of hosts infected each time and randomly select 10 days to inject attacks. Figure 2.14 lists the detection rates of the Windows worms and viruses that we manually launched inside the cluster and the number of suspicious files identified to explain the corresponding root causes.

For every attack, Seurat detected the appearance of a new cluster with high probability and successfully pinpointed the exact compromised hosts that were involved even when there were only 2 hosts infected. In the Blaster worm case, after we injected the attack into the isolated cluster, only 4 out of the 6 Windows virtual machines actually got infected. The other two were not infected for undetermined reasons.

Name	Blaster	LoveLetter	NetSky	Bagle
Type	Worm	Virus	Virus	Virus
2 infected hosts	90%	100%	80%	100%
4 infected hosts	90%	100%	100%	100%
Suspicious files identified	18	216	42	335

Figure 2.14: Detection rates of Windows worms and viruses. We merged the file update reports with reports from real deployment on 10 randomly selected days. The detection rate is the percentage of the days when Seurat detected the worms and viruses. The last row lists the number of suspicious files involved with the new cluster identified by Seurat.

Therefore, we could not compute the detection rate of Seurat by further increasing the number of infected hosts even though Seurat did not reach 100% detection rate with 4 infected hosts. The list of suspicious files further identified by Seurat indeed indicates that all of the launched worms and viruses changed files in unexpected ways. As an example, Figure 2.15 lists the suspicious files output by Seurat to explain the cause of a new cluster detected after we launched the Blaster worm. We expect these files will greatly facilitate system administrators to diagnose root causes and take counter measures. In summary, the high detection rates of Seurat in our Windows real worm experiments confirmed its effectiveness of detecting aggregated abnormal file update patterns inside a network.

2.5 Anomaly Detection with More Attributes

The binary feature vector space, described in Section 2.3.1, focuses mostly on file updates in the lack thereof. It may not be as effective in detecting stealthy attacks that modify only frequently, regularly updated files, which, even when they are modified in an unexpected way (e.g., entries removed from append-only log files), will exhibit normal update patterns and thus conceal the existence of an attack when represented using the binary vector space.

To detect such stealthy mimicry attacks, we can harvest the additional file update attributes that are reported by the Seurat data collection tool, for example, the file size changes and the last modification times. Next, we first describe a more general feature vector space that incorporates the additional information on file update events in Section 2.5.1. We then further explore how the incorporation of extra information impacts the detection performance in Section 2.5.2.

File ID.	File name
1	`C:\WINDOWS\system32\msblast.exe`
2	`C:\WINDOWS\Prefetch\MSBLAST.EXE-09FF84F2.pf`
3	`C:\WINDOWS\bootstat.dat`
4	`C:\WINDOWS\system32\config\SECURITY`
5	`C:\WINDOWS\Prefetch\TFTP.EXE-2FB50BCA.pf`
6	`C:\WINDOWS\system32\wpa.dbl`
7	`C:\WINDOWS\0.log`
8	`C:\WINDOWS\system32`
9	`C:\WINDOWS\system32\config\SAM`
10	`C:\WINDOWS\system32\config\default`
11	`C:\Program Files\VMware\VMware Tools\tools.conf`
12	`C:\WINDOWS\Prefetch\NETSTAT.EXE-2B2B4428.pf`
13	`C:\WINDOWS\Debug\oakley.log`
14	`C:\WINDOWS\system32\config\systemprofile\Cookies\index.dat`
15	`C:\Documents and Settings\gluser\Local Settings\desktop.ini`
16	`C:\WINDOWS\Debug\oakley.log.sav`
17	`C:\WINDOWS\system32\wbem\Logs`
18	`C:\WINDOWS\system32\config\default.LOG`

Figure 2.15: Suspicious files identified to explain the cause of a new cluster detected after we launched the Blaster worm.

2.5.1 A General Feature Vector Space

This section describes a general feature vector space that incorporates the additional file update information. Similar to the binary vector space, each dimension in the general vector space corresponds to a unique file. The value of each dimension, however, will be a numerical value reflecting the degree of change of the corresponding attribute of interest (e.g., file size change). The distance $D(V_1, V_2)$ between a pair of feature vectors V_1 and V_2 is defined as their Euclidean distance:

$$D(V_1, V_2) = ||V_1 - V_2||_2$$

In the above formula, $||.||_2$ denotes the L_2 norm. With such representation, two vectors with a small Euclidean distance are more likely to have similar values along every dimension and thus experience similar updates. In our current prototype, we consider only homogeneous file attributes when defining the general feature vector space. That is, we do not consider different types of attributes (e.g., file size and last modification time) simultaneously in the same vector space.

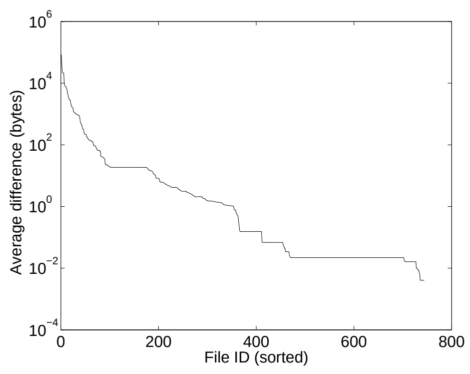
Given a file update attribute, we can use different ways to set the numerical value of a dimension. We consider file size changes and last modification times in our current prototype as they can be naturally represented as numerical values in the following ways:

- **File size change:** We expect that for each file associated with a propagating attack, the number of bytes changed across different compromised hosts would be close to each other. So given a file, we define the value of the corresponding dimension to be the number of bytes increased in a detection window (i.e., one day). A negative value indicates file size decrease, while a zero value suggests no file size change.
- **Last modification time:** The temporal locality of file updates due to attacks suggests that abnormal updates usually occur close in time with each other. Thus, for each file, we define the value of the corresponding dimension to be the last modification time in terms of the number of minutes elapsed since the beginning of a detection window (12:00 pm for each day in our current prototype).

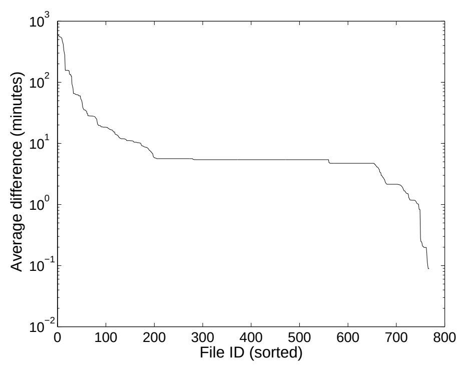
We note that the Euclidean distance computation weighs each dimension equally. Thus the distance between two vectors could be dominated by a small number of dimensions with larger differences. For example, a file whose size increased from 1,000 bytes to 2,000 bytes will have more impact on the computed distance than another file whose size increased from 1000 bytes to 1020 bytes. We take the 3-month file update reports collected from our Linux system deployment and examine the distributions of the one dimensional difference in feature vector spaces defined by file size changes and last modification times in Figure 2.16 (a) and (b), respectively. For each dimension (i.e., each file), we compute the average difference between every possible pair of vectors that Seurat generated during the entire data collection period. We observe that the one dimensional difference is highly biased across different dimensions (up to 7 orders of magnitude different) for both file size changes and last modification times. Such highly skewed distribution suggests that we need to normalize the values across different feature dimensions when computing the Euclidean distances.

In our current prototype, Seurat uses a standard max-min normalization method [45]. We normalize all the values of a dimension to within the range $[-100, 100]$. That is, for the i-th dimension, the maximal positive value v_i^{max+} is normalized to 100, while the minimal negative value v_i^{min-} is normalized to -100. Any other value v_i of the same dimension will be normalized to v_i^{norm}, where

$$v_i^{norm} = \begin{cases} 100 \times (v_i / v_i^{max+}) & \text{if } v_i \geq 0 \\ -100 \times (v_i / v_i^{min-}) & \text{if } v_i < 0 \end{cases}$$



(a)



(b)

Figure 2.16: The distribution of average one-dimensional difference across all dimensions (each dimension corresponds to a unique file). (a) Average difference between file size changes. (b) Average difference between last modification times.

Note there is a special case at data normalization when a file is deleted during a detection window. Seurat currently sets the value of a dimension to the minimal negative normalized value (i.e., -100) if the corresponding file is removed. This is consistent with the current definition of the feature dimension values. In the case of file size changes, a file removal event would result in the maximal decrease of the number of bytes, hence minimal negative value change. In the case of file last modification time, there will be no exact timestamp of when a file is removed using the current 1 day file update scanning cycle. Hence we use the minimal negative value to represent the unknown removal time. An alternative way is to treat file removals as special events that will be represented (and hence detected) only using the binary feature vectors.

2.5.2 Performance Impact

In this section, we examine how the incorporation of additional file update attributes will impact the detection rate and the false positive rate. We again use file size changes and last modification times to define the general feature vector space for anomaly detection. We use two sets of data for this experiment: (1) the file update reports collected from our real Linux deployment, and (2) the file update reports from the isolated cluster where we manually launched the Linux Lion worm.

Similar to the experiments with binary feature vectors, we set the correlation window size to 32 days, and use wavelet-based analysis to select a subset of relevant files. We then normalize the values of the selected feature dimensions and perform

	Reasons for Alarms	Number of Alarms
Detected using binary vector space	System reconfiguration events	6
	Network-wide experiment	3
Not detected using binary vector space	Unseen file updates detected	3
	Unseen update patterns detected	10

Figure 2.17: False alarms generated using feature vectors defined by file size changes.

PCA to further reduce the data dimensionality.

Attribute I: File Size Changes

We first use only file size changes to define the general feature vector space. Throughout the 60 days of detection under real deployment, Seurat, with this general feature vector space, raised a total of 22 alarms.

In order to understand why Seurat generated more alarms than by using the binary vector space, we examine the cause of each alarm based on the list of the suspicious files identified after clustering, and categorize the alarms in Figure 2.17.

We observe that using the general feature vector space, Seurat not only identified all the abnormal file update events that are detected by using the binary vectors, but also detected 10 file size update patterns that have not been observed before. Further inspection based on the abnormal file size update patterns suggests they are mostly caused by the log file size changes due to the variations of system usage daily. We expect that the number of such false alarms will be reduced by setting a larger correlation window to accommodate more instances of regular updates. Seurat also identified three events when each time there is a single new file being modified for the first time across 3, 6, and 5 hosts, respectively. We list the date and the file name identified for each of these 3 events in Figure 2.18. Again, we could investigate them based on the identified files. We found that, after each event, all the other hosts in the Linux cluster had the same file updates on the following day, which results in all hosts in the cluster being updated at last. They were administrative update that could be filtered if administrators provided Seurat with the update schedule beforehand. Since these three events changed only a single new file each time, they are more likely to evade detection using only the binary feature vector space with a low detect rate as we have explained in Section 2.4.3.

We proceed to examine the efficacy of the general feature vector space using the manually launched Linux Lion worm. Again, we merged the file update reports from the compromised hosts with the reports collected from the Linux teaching cluster on Feb 11, 2004. Seurat successfully identified a new cluster of the exact three compro-

Date in 2004	Number of hosts	File name
Jan 28	3	`/var/spool/lpd/norton`
Feb 6	6	`/etc/printcap`
Feb 23	5	`/usr/local/lib/nannyconfig/qpage`

Figure 2.18: Details of the three events when a single new file was modified for the first time. "Number of hosts" means the number of hosts involved in each file update event.

mised hosts using the file size change information. Figure 2.19 shows the clustering results on the day we injected the attack.

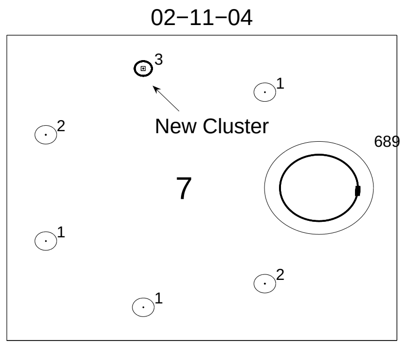


Figure 2.19: Seurat identifies a new cluster of the exact three infected hosts by the Lion worm with the general feature vector space defined using file size changes.

The above experiment results suggest that both the detection rate and the false positive rate will be more sensitive to the general vector space defined by file size changes compared with the binary vector space. On one hand, Seurat will be able to detect stealthy attacks that result in unexpected file size changes or install a very small number of new files with the additional information, and hence increase the detection rate. On the other hand, the false positive rate will also rise as legitimate unseen file size change patterns may also be misclassified as intrusion attempts. There is thus a tradeoff of detection rate and false positive rate when using the different types of feature vector spaces.

Attribute II: Last Modification Time

In this section, we use the last modification times to define the general feature vector space and repeat our experiments. Seurat also generated a total of 22 alarms dur-

	Reasons of Alarms	Number of Alarms
Detected using binary vector space	System reconfiguration events	6
	Network-wide experiment	3
Not detected using binary vector space	Unseen file updates detected	1
	Unseen update patterns detected	12

Figure 2.20: False alarms generated using feature vectors defined by last modification times.

ing the 60 days of anomaly detection using the file update reports from our Linux deployment (see Figure 2.20).

Similar to using file size changes, Seurat successfully identified all of the 9 alarms that were previously detected by using the binary feature vectors. A closer look at these 9 alarms, which were caused by system reconfigurations and network wide experiments, showed that they indeed have strong temporal correlations of file updates across different hosts. For majority of the rest of the alarms, however, Seurat could not pinpoint a small number files to explain for the new cluster. They are caused by the unseen overall update time patterns, suggesting that there exists no strong regularity of the last modification times across different hosts.

When we further evaluate Seurat with the manually launched Lion worm, it failed to detect the attack by generating a new cluster. Instead, using the general feature vector space defined by last modification times, each of the 3 vectors corresponding to the 3 compromised hosts forms an independent cluster itself. More careful investigation reveals that, although the Lion worm injected new files into a compromised host and caused unexpected file update time patterns on the same day, the infection times of the 3 compromised hosts are not close enough to be clustered together due to the attack propagation delay. This is because our data normalization described in Section 2.5.1 amplifies the differences between file update times. One way to address the problem is to use coarser granularity timestamps, for example, hours instead of minutes. However, selecting a best time granularity requires prior knowledge of attack propagation speed. In addition, attackers can more easily exploit the coarser granularity timestamps to blend the abnormal file updates with normal file updates. We plan to further investigate a more robust way of feature representation using last modification times as future work to this thesis.

2.6 Discussion

By identifying parallel occurrences of coincident events, Seurat will be most successful in detecting attacks that result in file modifications at multiple hosts. Certain attacks (e.g., password guessing attacks) that succeed only once or a few times in a network system may evade Seurat detection. The current prototype of Seurat also has limited detection capability to the following types of attacks.

Stealthy attack. Attackers may try to evade detection by slowing attack propagation. If an attacker is patient enough to infect only one host a day in the monitored network system, Seurat will not notice the intrusion with the current one-day detection window because Seurat focuses only on anomalous file changes common across multiple hosts. A larger detection window such as a couple of days or a week can help to catch slow, stealthy attacks. Note, however, that Seurat notices the attacks only after multiple hosts in the network system are compromised. In other words, if an attack propagates slowly, Seurat may not recognize the attack for the first few days after the initial successful compromise. There is thus a tradeoff between detection rate and detection latency.

Mimicry attack. An attacker can carefully design his attack to cause file updates that look similar to regular file changes, and mount a successful *mimicry attack* [104] by cloaking abnormal file updates with many normal but irregular changes during Seurat's clustering process. For example, in Section 2.4.3, we observed that the false negative rate of detecting the Kork worm was relatively higher due to the interference of irregular system reconfiguration. We leave it as future work to quantify this type of mimicry attack and the effectiveness of possible counter measures.

Random-file-access attack. Seurat correlates file updates based on their complete path names. Thus attackers can try to evade Seurat by installing attack files under different directories at different hosts, or replacing randomly chosen existing files with attack files. Many recent email viruses already change the virus file names when they propagate to a new host; we envision similar techniques could be employed by other types of attacks soon. Note, however, that even the random-file-access attack may need a few anchor files at fixed places, where Seurat still has the opportunity to detect such attacks. A more robust representation of a file, for example, an MD5 checksum, could help Seurat detect random-file-access attacks.

Memory-resident attack. Memory-resident and BIOS-resident only attacks make no file system updates. Thus Seurat will not be able to detect memory resident attacks by examining host file updates, nor those attacks that erase disk evidence before the Seurat data collection tool performs the scheduled disk scan.

Kernel/Seurat modification attack. The effectiveness of Seurat relies on the correctness of reports from the data collecting tools running on distributed hosts. So the host kernels and the Seurat data collection tools should run on machines protected by trusted computing platforms [102]. An alternative solution is to monitor file system changes in real time (will be discussed further in Section 2.7) and to protect file update logs using secure audit logging (e.g., [89]).

2.7 Future Work

Real-time anomaly detection. The current prototype periodically scans and reports file system updates with a 1-day cycle, which may be slow to detect fast propagating attacks. To shorten detection latency, we are enhancing the Seurat data collection module to monitor system calls related with file updates, and report the changes immediately to the correlation module. The reported file updates will be instantly reflected by setting the corresponding bits in the feature vectors at the Seurat correlation module, which continuously performs clustering of the new feature vectors for real time anomaly detection.

Distributed correlation module. Currently, Seurat moves the daily reports from distributed data collection tools to a centralized server, where the correlation module computes and clusters the host vectors. Despite the simplicity of centralized deployment, the centralized approach exposes Seurat to problems in scalability and reliability. First, since the input to the correlation are low level file update events, the amount of report data to be transferred to and stored at the centralized server is large. In our experience, a host generates a file update report of 3K-140KBytes daily in a compressed format, so the aggregate report size from hundreds or thousands of hosts with a long comparison window will be large. The report size will be larger when Seurat's data collection tool reports the host state changes in real time. Second, the monitored hosts could be in different administrative domains (i.e., hosts managed by different academic departments or labs) and it is often impractical to transfer the detailed reports from all the hosts to one centralized server due to privacy and confidentiality issues. Third, the centralized server can be a single point-of-failure. It is important for Seurat to work even when one correlation server is broken or a part of network is partitioned. A distributed correlation module will cope with those issues. As future work, we can investigate methods to correlate file update events in a distributed architecture such as EMERALD [79], AAFID [6], and Mingle [115].

Other applications. The approach of clustering coincident host state changes can be generalized to other types of applications such as detecting the propagation of spyware, illegal file sharing events, or erroneous software configuration changes. We are currently deploying Seurat on Planetlab [1] hosts for detecting software configuration errors by identifying host state vectors that do not fall into an expected cluster.

Chapter 3

Network Forensics Using Spatiotemporal Event Correlation

3.1 Introduction

Many types of Internet attacks utilize indirection as a means to hide their source. For example, the act of utilizing a chain of compromised machines, or "stepping stones," in an attack is a common means of foiling a defender's attempts to locate the source of an attack. Similarly, distributed denial-of-service (DDoS) attacks are often launched from compromised computers, sometimes called "zombies", both to harness the power of many machines and to obfuscate where the true source of the attack lies.

In all these attacks that utilize compromised computers to launch attack traffic, the overwhelming majority of the attack traffic originates from victims of the attack, as opposed to the true source of the attack. Such indirection is a highly successful means to provide anonymity to attackers, and to date there is little automated support for identifying the location (computer or network) from which such an attack is launched. Similarly, when an intranet succumbs to such an attack, there is little automated help to determine the internal computer that was compromised first.

In this thesis, we define an approach with the promise to dramatically change investigations of Internet-based attacks. Our high-level vision is an investigative capability for the Internet or intranets that permits identification and fine-grained analysis of the communication patterns leading to an attack, particularly including those attacks that utilize indirection as a means to hide their true source. Our goal is to bridge indirection techniques, such as stepping stones and zombies, to locate the original source of the attack or entry point to an intranet and to reconstruct how an attack unfolded.

We believe that this capability can significantly strengthen the hand of administrators in deterring attacks or permitting the correction of weak points in a network perimeter. For example, if an attack that propagated within an intranet can be traced back to its initial entry into the intranet—e.g., revealing an errant service or modem permitting connections that circumvent the firewall, or a user who is careless in the email attachments he opens—then that entry point can be corrected. And, of course, the ability to trace large-scale attacks to their ultimate source is a first step toward holding that attacker accountable.

While attacks can propagate via various rates and methods, use different signatures, and exploit different vulnerabilities, there is an invariant across all types (present and future) of attacks: for an attack to progress, there must be communication among attacker and the associated set of compromised hosts. Attacks utilizing indirection often exhibit network behavior that may not seem suspicious at the level of individual flows, but will nevertheless exhibit a discernible pattern when traffic is observed collectively. Such observation can be exploited to detect and trace attacks.

In this thesis, we formalize the problem of network forensic analysis, and present a *Dragnet* framework to define the required infrastructure and its properties where such observation can be exploited. In particular, we focus on the specific problem of crafting an algorithm that determines the origin of epidemic spreading attacks such as Internet worms. Our goal is not only to identify the "patient zero" of the epidemic, but also to reconstruct the sequence of attack flows that make up the initial stages of the attack tree via which the worm infected successive generations of victims. Instead of studying specific attacks that have been seen in the wild, we present an algorithm called *random moonwalks* that works by identifying the overall structure of an attack's propagation using *spatiotemporal event correlation.* Since the algorithm assumes no prior knowledge about attack specific properties, it can be agnostic to attack signatures or scanning rates, and potentially be applicable to all worm attacks.

The random moonwalk algorithm in essence is a Monte Carlo sampling method that identifies the "wide tree" structure of a worm propagation by performing random walks backward in time along paths of network flows. Correlating the flows traversed by repeated walks reveals the initial attack path, thereby aiding in identifying the source. Such algorithm is inherently robust to small amounts of missing data, and can be elegantly adapted to distributed scenarios where multiple domains collaboratively perform forensic analysis without sharing private data. We demonstrate through analysis, simulation, and experiments on real world traces that this approach can be highly effective in locating the origin of a wide class of worm attacks that propagate via different strategies, without the use of attack signatures.

The rest of this chapter is organized as follows. We first formalize the problem of network forensics in Section 3.2. We then outline an architecture for providing such new capability in Section 3.3. Section 3.4 describes the proposed algorithm for identifying the worm origins and the initial causal flows. We analytically study the effectiveness of the algorithm in Section 3.6. Section 3.7 and Section 3.8 evaluate the method using real-world network traces and simulation traces, respectively. Section 3.9 presents a formal interpretation of our algorithm, where we show that the algorithm can be regarded as performing spectral analysis on a flow graph constructed from the network trace. In Section 3.10, we discuss distributed forensic analysis in practice, examine the impact of missing data on performance, and present a distributed random moonwalk algorithm to allow multiple administrative domains for identifying the worm origin collaboratively. In Section 3.11, we study variations of our algorithm, where we may incorporate additional knowledge with biased sampling strategies. Finally, we discuss in Section 3.12 deployment issues as well as future work.

3.2 Problem Formulation

We define the problem of tracing and reconstruction of arbitrary network attacks in terms of two fundamental components, *Attacker Identification* and *Attack Reconstruction*. These act as building blocks on which attack investigation can be based and attackers held accountable. Attacker Identification is the ability to accurately pinpoint the source(s) of the attack or infection. Attack Reconstruction is the process of inferring which communications carry the attack forward. This not only identifies the compromised hosts for subsequent correction, but also provides crucial information about the attack propagation that can help in precluding future attacks of a similar kind. The focus of our work on identifying the true source of attacks through Attacker Identification and Attack Reconstruction differentiates it from other efforts that seek to identify when an attack is occurring or to reactively blunt the effect of an attack already in progress.

Figure 3.2 shows a general model of how an arbitrary network attack propagates from one host to another across administrative domains. We mark each of the attack flows with a timestamp to indicate when the attack was received by the victim. The attack originates at host A at time t_1, and then propagates to hosts B to H.

We model these network communication events between end-hosts using a directed *host contact graph* $G = \langle V, E \rangle$. The nodes of the graph $V = H \times T$, where H is the set of all hosts in the network and T is time. Each directed edge represents a network *flow* between two end hosts at a certain time, where the flow has a finite duration, and involves transfer of one or more packets. We represent each edge by a tuple $e = \langle u, v, t^s, t^e \rangle$ where $u \in H$ is the host that initiates the communication (the source of the flow), $v \in H$ is the host that receives the communication (the destination of

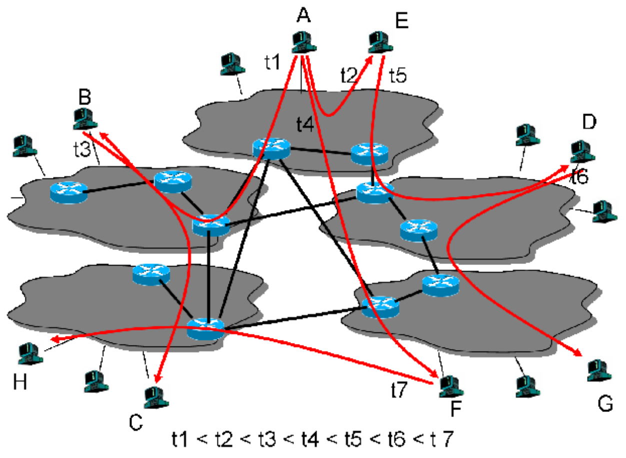


Figure 3.1: A simplified network topology showing routers, hosts, and ISP boundaries. The arrows between hosts illustrate communications that spread an attack through the network.

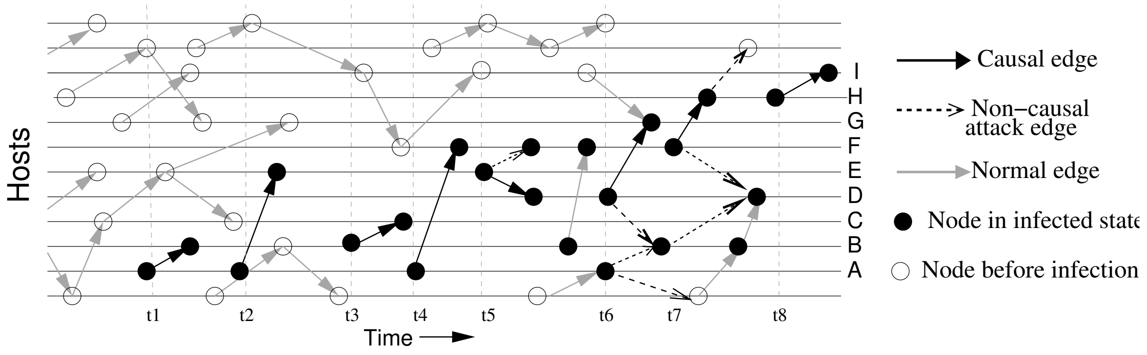


Figure 3.2: Example of host contact graph showing the communication between hosts. Attack edges are shown as arrows in black (both solid and dashed). Filled nodes correspond to hosts in an infected state.

the flow), and t^s, $t^e \in T$ are the start and end times of the flow. Edge e is thus from node $(u, t^s) \in V$ to node $(v, t^e) \in V$. Including time in the model is important, as a single host $h \in H$ that becomes infected during an attack behaves differently before the time it is infected than it does afterwards.

Figure 3.2 shows the host contact graph of a hypothetical network undergoing an attack. Time advances left to right. Each node (marked as a circle) in the graph corresponds to the state of a host at a certain time. The nodes on the same horizontal line show how the state of one host changes over time, and the nodes on the same

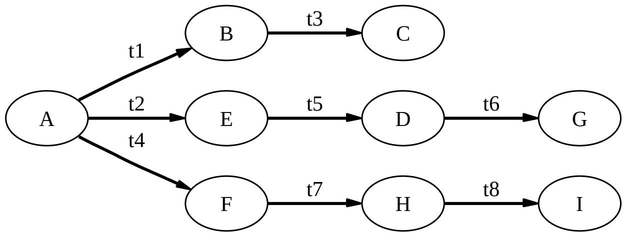


Figure 3.3: Example showing the causal tree, which contain causal edges with timestamps from the host contact graph.

vertical line represent the states of different hosts at the same time.

Each directed edge in Figure 3.2 represents a network flow. If a flow does not carry an infectious payload, we call that edge a *normal edge*. We define an edge as an *attack edge* (highlighted in the figure as either dashed or solid arrows) if it corresponds to a flow that carries attack traffic, whether or not the flow is successful in infecting the destination host. While a worm attack may induce a large number of attack flows in the network, only a few flows actually advance the attack by successfully infecting a new host. We define an edge as a *causal edge* (highlighted as a solid arrow) if it corresponds to a flow that actually infects its destination. For example, at time t_6, host D has attack edges to both hosts G and B. However, only the edge from D to G is a causal edge because G is infected by this contact, whereas B was infected earlier before time t_2.

The *causal tree* formalizes the concept of epidemic attack spread. The causal tree is formed by extracting the causal edges from the host contact graph and projecting the edges along the time axis. To be consistent with the notion of time in the host contact graph, we consider causal edges occurring earlier in time as edges in the higher levels of the causal tree. Figure 3.2 shows the causal tree for the attack in Figure 3.2, with each edge annotated with a timestamp. The edge with timestamp t_1 from the worm origin A is thus at the highest level of the tree.

Given the host contact graph, Attack Reconstruction identifies which edges are causal edges that advanced the attack. Even infected hosts may continue their normal activities and not all infection attempts succeed, so Attack Reconstruction must carefully infer which communication initiated by an infected host should be marked as carrying the attack, and reconstruct the host attack tree using the marked edges. Attacker Identification operates by working its way back up the causal tree to find the root sources of the attack. Even when the ultimate root and source of the attack tree cannot be found (e.g., due to missing data), the higher level nodes of the causal tree that are reconstructed point the way toward the true attack source and provide a starting point for out-of-band human investigation.

3.3 A Framework For Investigating Attacks

Identifying the propagation of an attack is particularly difficult as the adversary is intelligent: attackers are bound to come up with smarter mechanisms trying to evade detection. However, there is one fundamental invariant across all attacks (present and future): for the attack to progress there must be communication among attacker, the associated set of compromised hosts and the victim(s), and *this communication is visible to the network*.

The communication between attackers and victims may be subtle and invisible when observed from any single host without foreknowledge of the attack signature, but it will potentially stand out when viewed globally as the attack propagates. As a simple example, efforts to detect stepping stones have been premised on the identification of flows that exhibit closely correlated packet contents or inter-packet timings [120, 28]; while each flow in isolation may seem innocuous, together they reveal suspicious behavior. In general, our approach is to develop algorithms that correlate the communication events among individual hosts across both space and time, and identify the patterns that indicate a propagating attack. Since no accurate signature is required, our approach has the potential to be robust against changes in the behavior of attacks.

Applying our approach to a large scale network requires a means to: (1) gather and query host communication records from distributed network locations; and (2) design analysis algorithms for identifying global communication patterns given the host contact graph.

3.3.1 Network Auditing

We need widely deployed infrastructure support where distributed collection points log traffic records and store them in repositories for querying. As has been discussed in Section 3.2, we focus on end-host connectivity patterns in terms of directional network "flows", where each flow identifies a directional communication event between a source and a destination, and carries timestamps indicating the start and stop time of the flow. Compared with packet traces, flow records are more compact representations and need not capture packet payloads, which might be construed as a violation of end-user privacy.

At the scale of an intranet, traffic logging can be deployed as pervasively as necessary. At the Internet scale, the problem of obtaining an approximation of the complete host contact graph appears intractable. However, there are features of the Internet topology and routing that suggest auditing devices can be deployed at a relatively small number of locations to approximately construct the host contact graph. First,

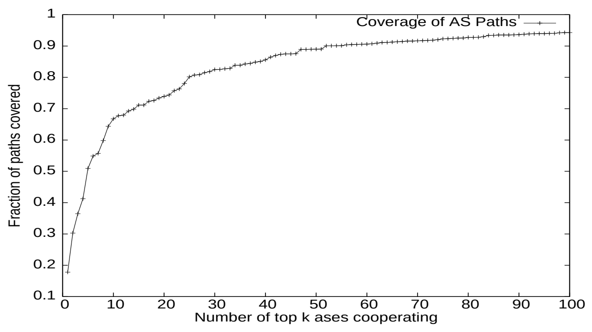


Figure 3.4: Path coverage vs number of highest degree ASes selected.

Internet routing is highly constrained by inter-domain policies and the AS hierarchy, which implies that top level ASes will observe a significant fraction of the traffic. Second, the Internet AS-level connectivity graph has been shown to be a power-law graph [31], suggesting that high degree ASes are good candidates for logging deployment.

To estimate how many observation points should be deployed in the Internet for a given fraction of the flows to be logged at least once, we investigate the optimal path coverage problem, related to the problem of deploying reverse path filters [74] and containment filters [64], assuming flows uniformly distribute among all available Internet paths. We define an AS graph $G = \langle V, E \rangle$, where V is the set of ASes, and E is the set of inter-AS connections. Let P be the set of paths used for routing traffic in G. We say an AS a $(a \in V)$ *covers* a path p $(p \in P)$ if a is on the path p. Given these notions, we want to select the smallest AS subset $V' \subseteq V$ that covers α percent of the paths in P.

We consider a greedy heuristic where the ASes are selected based on their degrees, with higher degree ASes given higher preference. Figure 3.4 shows the fraction of AS-paths that are "covered" by the first k high-degree ASes, using AS paths and AS degrees obtained from RouteViews [84] in Feb 2004. The figure shows that if the 50 ASes with highest node degrees deployed flow auditing, these ASes would "cover" approximately 90% of all the paths, and hence be able to log any flow traversing one of these paths.

To help reveal the causal relationship between flows for attack reconstruction, ideally all logging devices in the auditing infrastructure should have perfect time synchronization (perhaps achieved using GPS time sources or NTP [71]. In reality, the logging points will be located in diverse geographical locations and across multiple ASes, and time synchronization cannot be guaranteed. However, it appears both practical and sufficient to ensure two flows are timestamped consistent with the causal relationship between them (if any) [55]. "Causal consistent timestamps" mean that if a flow A is, in fact, caused by flow B, then A will have a timestamp later than

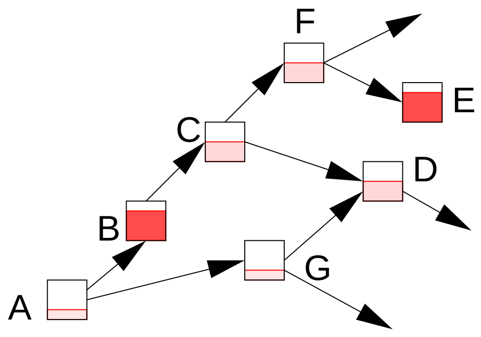


Figure 3.5: Concept behind attack reconstruction: the probability of being involved in an attack should be reinforced by network events. The fraction of host filled in represents the probability that host is involved in an attack, as computed by an intrusion detection system. Given the probability of B and E's involvement and the communication pattern between the hosts, the likelihood of C, D, and F's involvement should be increased.

B. This property is achieved if there is any single router that records both flows of interest or if there are any two time-synchronized routers (e.g., they are in the same administrative domain) where flow A is recorded by one router and flow B by the other.

3.3.2 Techniques for Attack Reconstruction

The key component of the analysis lies in the algorithmic components that analyze the data log to extract "interesting" communication patterns that are part of an ongoing network attack. The attack reconstruction algorithms may operate over a centralized data repository (in the case of an intranet), or will need to interact with distributed query routing and data retrieval mechanisms (in the case of a larger internet). There exists an entire spectrum of design decisions that may involve fundamental tradeoffs regarding issues such as scalability, effectiveness of the solutions, susceptibility to evasion, and overall performance. These design decisions will also have to take into consideration the deployment scenario, as the requirements and capabilities for an Internet vs Intranet scale analysis framework are vastly different.

One general category of reconstruction methods is to reveal the causal relationship between communications by spatially correlating local observations. Figure 3.5 illustrates the concept. The fraction of each host that is filled in represents the probability that that host is involved in the attack. These initial probability estimates are based on locally observed behavior, such as those a local host-based Intrusion Detection System (IDS) might observe. Based on the local observations alone, it is likely B and E are involved in an attack. Given the communication pattern among the hosts, however, it is extremely likely that hosts C, D, and F are also involved, although their locally observable behavior was not suspicious enough to trigger an IDS.

Initial Local Observations

The input to Attack Reconstruction is one or more estimates of the probability that a host is involved in an attack. We can leverage prior work in this area, as these estimates can come from any existing IDS or network monitoring system that detects the occurrence of attacks in the network. For example, we can use measures such as the fanout of the host in any given time interval, the rate at which the host generates traffic, and the set of destinations that the host contacted within a time interval. For each of these measures, we can use anomaly detection mechanisms to observe deviations from normal behavior and compute a confidence estimate for the probability an host is involved in an attack.

Attack Reconstruction Using Host Contact Graphs

Given the initial probabilities of hosts involved in an attack, the host contact graph obtained from the audit logs can be used to refine the involvement probability estimates and reconstruct the attack. The goals are to: (1) identify the edges that successfully attack a new host (i.e., are causal) and (2) the top-level nodes of the causal tree.

In this thesis, we explore one particular algorithm using *random moonwalks* to identify the origin of epidemic spreading attacks such as Internet Worms by finding initial causal flows on the causal tree. The algorithm takes network flows between end hosts as initial local observations, and assumes uniformly distributed probability estimate of a host being compromised. Given a host contact graph, the goal of our algorithm is to *identify a set of edges that, with high probability, are edges from the top level(s) (i.e., initial in time) of the causal tree.* Among the hosts listed as the sources of these edges will be the origin of the attack (or the host at which the attack first entered the intranet). It is critical that the technique have a reasonably low false-negative rate, so that the returned set contains at least one top level causal edge that identifies the attack origin. It is desirable that the technique have a low false-positive rate, so that the returned set does not include many normal edges, attack edges that do not infect the destination, or even causal edges that occur lower in the causal tree, since the sources of these edges are less likely to be the true origin of the attack.

In the next several sections, we focus on this specific algorithm, where we apply the spatiotemporal event correlation approach. We start by exploring the algorithm in a centralized scenario where we have the complete host contact graph available. We then theoretically study the algorithm to understand how it analyzes the underlying graph structure. Finally, we look at algorithm extensions in a distributed environment where portions of data might be missing, and each party can share only limited information.

3.4 The Random Moonwalk Algorithm

Given the complete knowledge of a host contact graph, our algorithm consists of repeatedly sampling paths from the graph and then correlating these samples. The edges that occur most frequently among the samples are selected as the edges most likely to be causal edges from levels higher up in the causal tree. The first key to the technique is that we do not sample individual edges — rather, each sample is a contiguous path of edges in the graph. The second key is that we create the path by starting at a randomly chosen edge, and then *walking backwards in time* along the graph, randomly choosing among potential predecessor edges at each step in the moonwalk.

The sampling process is controlled by three parameters: W - the number of walks (i.e., samples) performed, d - the maximum length of the path traversed by a single walk, and Δt - the sampling window size defined as the maximum time allowed between two consecutive edges in a walk. Each walk starts at an arbitrarily chosen edge $e_1 = \langle u_1, v_1, t_1^s, t_1^e \rangle$ representing a flow from host u_1 to host v_1. We then pick a next step backward in time uniformly from the set of edges that arrived at u_1 within the previous Δt seconds. That is, an edge $e_2 = \langle u_2, v_2, t_2^s, t_2^e \rangle$ such that $v_2 = u_1$ and $t_2^e < t_1^s < t_2^e + \Delta t$. Each walk stops when there is no edge within Δt seconds to continue the path, or the path has traversed the specified maximum number of hops d.

As the sampling is performed, a count is kept of how many times each edge from the host contact graph is traversed. After W walks have been performed, the algorithm returns the Z edges with the highest counts. Here, Z is a user specified parameter to determine how many edges are to be returned for further investigation. These edges are most likely to be top-level causal edges from the causal tree. As defined and used in this paper, the algorithm operates off-line with the parameters and host contact graph as inputs. As future work, we are investigating on-line versions that may also dynamically tune parameters.

Each random moonwalk made by the algorithm samples a *potential* causal chain of events. Because the walks wander into the past, the edge at step i (time $= t_1$) in a walk could be potentially caused by the edge at step $i+1$ (time $= t_2$, where $t_2 < t_1$). Since the walks begin at different randomly chosen edges, an edge that shows up frequently among many walks has the potential to be indirectly responsible for causing a large number edges in the host contact graph. Worm attacks have the property that a small number of edges (those high up in the causal tree) *are* indirectly responsible for causing a large number of edges in the host contact graph (the attack edges lower in the tree). Thus the edges implicated by our sampling algorithm are likely to be those high in the causal tree.

Two factors appear to aid in the convergence of the sampling algorithm, although it remains future work to determine the relative importance of each factor.

First, an infected host generally originates more flows than it receives. If the worm makes attack attempts very rarely this difference may be slight, but sending attack flows increases the rate of outgoing flows without increasing the rate of incoming flows. The result is that there are more edges that can lead a walk to an infected host than there are edges that lead away from it. This tends to concentrate walks towards the root of the tree.

Second, in normal communication patterns today, most hosts are clients that initiate communication with servers, and so are the originators of flows in the host contact graph. Since hosts receive relatively few flows, random moonwalks in a host contact graph without an ongoing worm attack tend to be very short, as many edges have no predecessors within the Δt sampling window. Worms, port scanning, and peer-to-peer systems are among the few applications that cause hosts to receive flows, and port scanning or peer-to-peer systems tend to lack the tree-structure that cause random moonwalks to concentrate.

3.5 Evaluation Methodology

We evaluate the random moonwalk algorithm using an analytical study, real trace experiments, and simulations, with different models of background traffic and different worm propagation rates. We first present in Section 3.6 analytical results with a simplified traffic model, showing that the random moonwalk technique has promise, and give analytical estimates on the performance of the algorithm. Section 3.7 presents experimental results with a large real network trace, to demonstrate the success of the algorithm in discovering the initial causal edges under various attack scenarios including worms propagating at very slow rates. We also discuss how to select parameter values for maximum walk length d and sampling window Δt for an arbitrary network trace. For completeness, we present in Section 3.8 a set of simulation experiments to show the performance of the algorithm under different background traffic models.

As discussed earlier, the output of the random moonwalk algorithm is a set of the Z edges that were traversed most frequently during the W moonwalks. Given the

Z returned edges, we use three performance metrics to evaluate the performance of the algorithm: (1) the detection accuracy in terms of the number of causal edges and attack edges returned, (2) the false positive rate of the set of edges returned, and (3) the number of suspect hosts identified by the algorithm as potential origins of the worm.

As our goal is to identify the initial causal edges whose source is the worm origin, attack edges and even causal edges from lower levels of the causal tree are considered as false positives. In the analytical study, we develop a model for reasoning about the false positive rates associated with finding only the *top-level causal edges*. In real attacks, the notion of *top-level* edges loses meaning, since the assumptions simplifying the notion of time and the unit duration of a flow (made in the analysis) no longer hold. Therefore, in the simulation and real trace studies, we evaluate performance using *detection accuracy* of the number of causal edges among the Z top frequency edges. We then use experiments to show that the majority of the returned causal edges are from the highest levels of the causal tree, with the worm origin as one of the sources of the edges.

3.6 Analytical Model

In this section, we present an analytical model that explains how well the random moonwalk sampling process works and why. Using the analytical model, we show how we can predict the sampling performance achieved from W walks with maximum length d and given Δt.

3.6.1 Assumptions

To enable tractable analysis of the random moonwalk sampling, we make simplifying assumptions about the structure of the host contact graph and the attack. Although our model is an over-simplification of real network traffic, it enables an estimation predicting the performance of the technique and sheds light on the intuition behind the effectiveness of the technique.

First, we assume the host contact graph is known, and it contains $|E|$ edges and $|H|$ hosts.

Second, we discretize time into units. We assume every flow has a length of one unit, and each flow starts at the beginning of a unit and finishes before the start of the next unit.

Third, we define the start time of the first attack flow, T_0, to be the origin of the time axis. Combined with the second assumption, this means that rather than

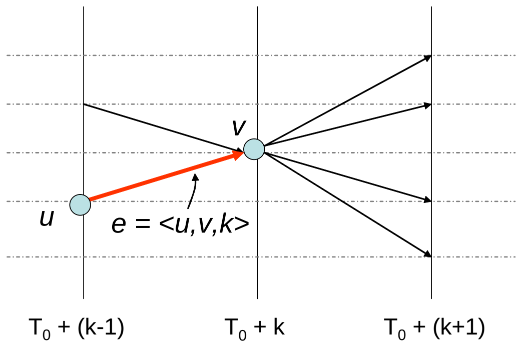


Figure 3.6: An edge at time k in the host contact graph.

describing both the start and end times of an edge in terms of continuous time variables, we can refer to its "time" as $k = t^e - T_0$ using just the flow end time t^e. The first attack edge is then at time $k = 1$, and an edge $e = \langle u, v, t^s, t^e \rangle$ is at time k if $t^e = T_0 + k$ (illustrated in Figure 3.6). In the analysis below, we use e^k to denote an edge at time k, $e^k = \langle u, v, k \rangle$. Edges that occurred before T_0 will have negative k values.

Fourth, we assume a normal host initiates B concurrent outgoing flows at each time unit. Once a host is infected, it starts malicious scanning by initiating a total of A outgoing flows at each subsequent time unit. The A outgoing flows include B normal flows and $A - B$ attack flows. Both the normal hosts and the infected hosts randomly select a destination host for every flow. Unlike a normal flow, not every attack flow will go to a valid host address. Suppose only fraction r of the address space is being used, then among the $A - B$ concurrent outgoing attack flows, $R = (A - B) \times r$ will go to existing hosts, while the rest $A - B - R$ will go to invalid destinations. This results in an infected host initiating a total of $B + R$ flows to valid destinations each time unit. The rate at which the worm spreads is thus determined by both A, the rate of scanning, and R, the effectiveness of the scans.

Finally, we assume that flows and packets are not lost or blocked, so that flows sent to a valid host are received by that host. This means that the total number of flows sent to valid hosts at time $k - 1$ will be the total number of flows received at time k. If the fraction of infected hosts at time $k - 1$ is given by $f(k - 1)$, then each host at time k will receive an average of $I(k)$ flows, where

$$I(k) = \underbrace{(B + R) \times f(k-1)}_{\textit{Flows from infected hosts}} + \underbrace{B \times (1 - f(k-1))}_{\textit{Flows from normal hosts}} \qquad (3.1)$$

With the notions introduced above, we can simplify the random moonwalk algorithm described in Section 2.3. For each walk, once we select an edge $e_1 = \langle u_1, v_1, k_1 \rangle$ as our current step, we consider an edge $e_2 = \langle u_2, v_2, k_2 \rangle$ as a candidate next step only if $v_2 = u_1$ and $k_2 + 1 = k_1$, i.e., $\Delta t = 1$.

3.6.2 Edge Probability Distribution

With the above assumptions and notation, we show analytically that the initial causal flows are more likely to be traversed by a random moonwalk, and thus be selected for identifying the ultimate source or entry point of the attack. We do so by estimating $P(e)$ — the probability of an edge e being traversed in a random moonwalk on the host contact graph.

We classify edges into two categories based on their destinations. We define an edge $e_m^k = \langle u, v, k \rangle$ as a *malicious-destination* edge if v is infected before or at time k. Otherwise, we define the edge as a *normal-destination* edge denoted as e_n^k. Since a causal edge will successfully infect the destination host immediately, a causal edge is always a malicious-destination edge. With the two categories of edges, we have the following approximations:

$$P(e_w^k) \approx \begin{cases} \frac{1}{|E|}\left[1 + \frac{A}{I(k)} + \frac{(B+R)\times\sum_{i=1}^{d-2} T_{k+i}}{I(k)I(k+1)}\right] & w = m; \\ \\ \frac{1}{|E|}\left[1 + \frac{B}{I(k)} + \frac{B\times\sum_{i=1}^{d-2} T_{k+i}}{I(k)I(k+1)}\right] & w = n. \end{cases}$$

where $T_k = Af(k) + B[1 - f(k)]$.

We present how we derive the above estimates in the Appendices (Chapter 6.1). Based on the above observations, the probability difference between the two categories of edges is estimated as:

$$P(e_m^k) - P(e_n^k) \approx \frac{1}{|E|}\left[\frac{A-B}{I(k)} + \frac{R\sum_{i=1}^{d-2} T_{k+i}}{I(k)I(k+1)}\right] \tag{3.2}$$

For fast propagating worms, $A >> B$ and $R > 0$, so it is clear malicious-destination edges (hence causal edges) have higher probability of being selected by the random moonwalks than normal-destination edges. The difference between the two probabilities (hence the effectiveness of random moonwalks) increases as the path length d increases and as the scanning rate A increases (i.e., the worm is more aggressive).

The analytic model presented in this section makes a worst-case assumption that

Number of Edges $\lvert E\rvert$	4.9×10^7
Number of Hosts $\lvert H\rvert$	10^5
Vulnerable fraction F	0.1
Valid host space r	0.5
Normal rate B	2
Infection rate A	400
Attack start time	0
Attack stop time	15

Figure 3.7: The parameters of a host contact graph with a fast propagating worm.

both normal and attack traffic choose the destination for each flow uniformly from among all possible hosts. Therefore, it cannot predict the performance of the algorithm on worms that send attack flows less frequently than normal flows (i.e., setting $A < B$ is meaningless). In the sections that follow, we show experimental evidence that the algorithm is effective even for very stealthy worms where infected hosts send attack flows more slowly than the rate at which normal flows are sent.

Interestingly, the effectiveness of the random moonwalk algorithm *increases* as the scan rate to valid hosts R increases. This means that the *fewer* packets the worm sends to invalid addresses, the *easier* it is to catch, which nicely complements honey-pot techniques that detect worms that send many packets to non-existent destinations.

To estimate how $P(e)$ distributes as an attack evolves, we need to estimate both $I(k)$, the expected number of incoming edges at a host at time k, and $f(k)$, the fraction of infected hosts in the network. The fraction of infected hosts $f(k)$ can be estimated using a logistic equation [97] that models the growth rate of epidemics. Since an infected host randomly scans the network to propagate the attack, among the total R concurrent outgoing attack flows to valid hosts, $R \times [F - f(k-1)]$ flows will infect vulnerable hosts that have not been infected before, where F is the fraction of vulnerable hosts in the network. Thus

$$f(k) = \begin{cases} 1/|H| & k = 0 \\ f(k-1)\,[1 + R \times (F - f(k-1))] & k > 0 \end{cases}$$

Figure 3.8 (a) shows the growth of the fraction of infected hosts as a fast propagating worm progresses on the host contact graph described by parameters in Figure 3.7. We observe that as the attack advances, the number of infected hosts grows quickly until all vulnerable hosts are compromised and the attack saturates. This rapid growth results in a non-uniform probability distribution of the edges being traversed.

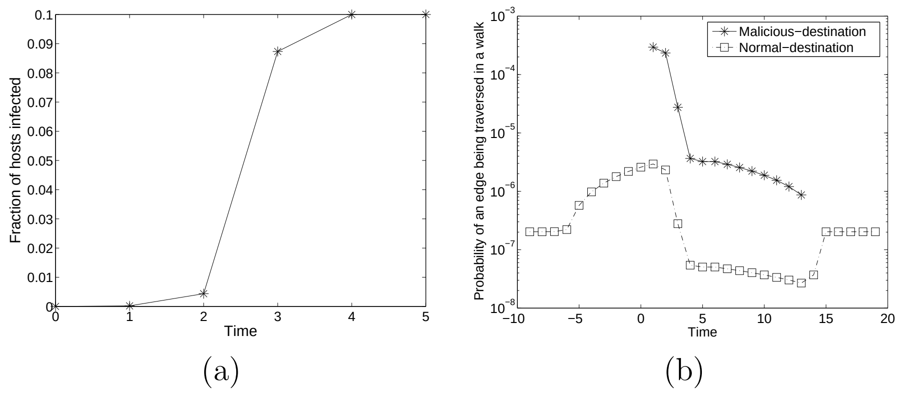


Figure 3.8: (a) Fraction of infected hosts as an attack advances. X-axis represents time in terms of units. Y-axis corresponds to the fraction of hosts that are already infected in the host contact graph. The total fraction of vulnerable hosts is 0.1. (b) Estimated probability of an edge being traversed in one random moonwalk.

Figure 3.8 (b) shows how $P(e_m^k)$ and $P(e_n^k)$ change over time in an attack scenario as described in Figure 3.7 with d set to 10 hops. The attack starts at time 0 and ends at time 15, so there are no values for $P(e_m^k)$ shown outside this range. The graph shows that the probability $P(e)$ is highest for malicious-destination edges at times close to the start of the attack. This occurs because the rapid spread of the worm and its zealous scanning means that for time $k > 2$, the majority of the edges received by a host are from infected hosts (i.e., $(B+R)\times f(k-1) > B\times[1-f(k-1)]$ for $k > 2$). This results in almost all walks started at times $k > 2$ selecting an attack edge as the next step backward. Further, as the total number of infected hosts increases with time, $I(k)$ increases monotonically in the time interval $[0, 5]$ (the attack saturates at $k = 4$). Therefore, random moonwalks tend to traverse edges between infected hosts, and converge to the topmost levels of the causal tree. The probability of traversing a normal edge at time k, $P(e_n^k)$, is a constant until $k = -5$ at which point it grows until $k = 2$, shortly after the attack starts. This growth occurs because walks started at times $0 < k < 10$ tend to concentrate as they walk backward in time along the attack edges until they walk past the beginning of the attack, at which point they begin diffusing through the normal edges. Thus normal edges received by nodes infected early in the causal tree are sampled more frequently than normal edges that occurred at $k < -5$.

Equation 3.2 and Figure 3.8 (b) suggest that random moonwalks will be most effective in selecting the malicious-destination edges that occur at the highest levels of the causal tree. Identifying these edges, in particular the $k = 1$ edges, reveals the origin or entry point of the attack.

3.6.3 False Positives and False Negatives

In this section, we analytically study the effectiveness of our algorithm by calculating the expected false positive and false negative rate. From the output set of the Z top frequency edges after W random moonwalks, we are particularly interested in finding the $k = 1$ causal edges, because the source of these edges is the origin of the attack. We thus focus on the $k = 1$ causal edges using the definitions below.

- *false positive rate* is the number of non-causal edges and the number of $k > 1$ causal edges in the set divided by the total number of non-causal edges; and
- *false negative rate* is the number of $k = 1$ causal edges not identified divided by the total number of causal edges.

Notice with this definition, we consider failed infection attempts (those scans that reach non-vulnerable hosts), repeated infection attempts (those scans that reach already infected hosts), and even lower level causal flows (those scans that successfully infect hosts at time $t^e > 1$) as false positives, if identified by our algorithm.

The number of times a $k = 1$ causal edge appears in W random moonwalks can be represented as a random variable $\mathbb{X}$ that follows a binomial distribution with $p = P(e^1_m)$. For large W, $\mathbb{X}$ can be approximated by a normal distribution [112] with mean $\mu = p \times W$ and standard deviation $\sigma = \sqrt{p(1-p)W}$. To ensure the $k = 1$ causal edges are included in the output set with a false negative rate of α, we need to select all the edges whose sample frequencies are above a threshold value of Z_α such that $Pr(\mathbb{X} < Z_\alpha) = \alpha$.

Among the selected edges will be the desired $k = 1$ causal edges and three types of false positives: (1) normal-destination edges, (2) malicious-destination edges with $k > 1$ (both causal and non-causal edges), and (3) $k = 1$ malicious-destination, but non-causal edges (i.e., a normal flow sent to a host at $k = 1$ which was also infected at $k = 1$). The last type of false positives arise because these normal edges have the same probability of being sampled as a $k = 1$ causal edge. These errors are unavoidable, but false positives from the first two categories can be reduced by increasing W.

To illustrate the performance of the algorithm, we use the same host contact graph described by Figure 3.7 where there are in total 10^4 causal flows out of the 4.9×10^7 flows. Among the 42 malicious-destination edges at $k = 1$, 20 are causal edges while the remaining 22 fall under the third category of false positives (i.e., normal edges sent to a host that was infected at $k = 1$); which means that in the ideal case 1 out of 2 edges selected will be causal edges. To estimate the false positives arising from the first two categories, we need to compute the probability of an edge e with $P(e) = p'$ having sample frequency $\mathbb{X}'(e) \geq Z_\alpha$ over the W random moonwalks, where e is either

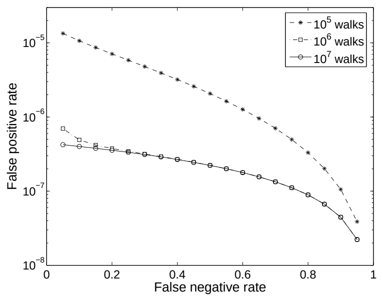


Figure 3.9: False negative rate vs. false positive rate of finding $k = 1$ causal edges.

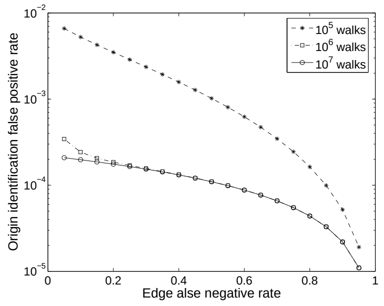


Figure 3.10: Estimation of the maximum false positive rate of identifying the attack source.

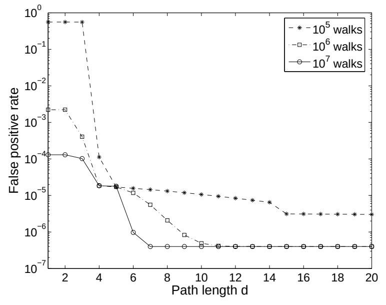


Figure 3.11: False positive rate of finding $k = 1$ causal edges vs. maximum path length d.

a normal-destination edge or a malicious-destination edge with $k > 1$. Again, $\mathbb{X}'(e)$ is a random variable approximated by a normal distribution. With a threshold value of Z_α used to select edges, suppose $Pr(\mathbb{X}'(e) \geq Z_\alpha) = \beta$. Let $|E(p')|$ be the total number of edges with $P(e) = p'$, then $\beta|E(p')|$ edges will have sample frequencies larger than the threshold Z_α and be falsely included in the output set.

Figure 3.9 plots the false negative rate vs. false positive rate for identifying the $k = 1$ causal edges as the number of walks W varies using the parameters described in Figure 3.7. In general, the false positive rates are low even for small false negative rates. With 10^6 walks, the false positive rate is 0.5×10^{-6} with a false negative rate of 0.1. This means that the chance of a non-causal edge or a lower-level causal edge being selected by the technique, when 90% of the $k = 1$ causal edges are identified, is about 0.5 in a million. The false positive rate drops with increased number of walks, but the rate of decrease slows when the number of walks is larger than 10^6.

We are primarily interested in identifying the worm origin, and the source of every flow returned by the algorithm is a candidate for the origin of the worm. Thus it would be ideal to present to a network administrator a small set of suspect hosts that need to be investigated further. We define the *origin identification false positive rate* as the number of innocent hosts among the sources of the flows selected by the algorithm divided by the total number of hosts minus one (we assume the worm has a single origin). We compute a conservative upper bound by assuming every selected flow returned by the algorithm is from a unique source.

Figure 3.10 plots the origin identification false positive rate vs. causal edge false negative rate for different numbers of walks. Since there are multiple causal edges from the worm origin, identifying the origin should work well even if there is a slightly higher false negative rate for causal edges. In this example, if we wish to select 70% of the $k = 1$ causal edges to confirm the attack origin, then after 10^6 walks there will be at most 16 candidate hosts for the worm origin from a total of 10^5 hosts, greatly reducing the suspect set for further investigation.

3.6.4 Parameter Selection

Understanding the impact of the choice of input parameters d and W on the performance of the random moonwalks is important as these parameters determine the amount of sampling effort required.

Figure 3.11 shows the false positive rate for different values of d (the maximum length of the random moonwalk) and W (the number of walks) with the false negative rate held constant at 0.1. We observe that longer walks generally result in lower false positive rates. This is also suggested by Equation 3.2, where the difference between $P(e_m^k)$ and $P(e_n^k)$ increases as d increases. The reason is that when random moonwalks start from lower level edges of the attack tree, they may end before reaching the origin of the attack, increasing the false positive rate. We will further address the impact of parameters d and the sampling window size $\triangle t$ on performance using real-world traces in Section 3.7.4.

3.7 Real Trace Study

In this section, we present our experimental results using real world traces collected from a university network. The objective of the trace based study was to both test the effectiveness of the random moonwalk algorithm using real traffic and to study the performance of the algorithm in different attack scenarios. As our analytical study argues the effectiveness of the algorithm for fast propagating attacks, we focus the real trace study on stealthy attacks that generate low traffic volumes that might escape traditional scanner and super-spreader detection mechanisms.

The traffic trace was collected over a four hour period at the backbone of a class-B university network, where we can observe a significant fraction of the intra-campus network traffic. Each record in the trace corresponds to a directional flow between two hosts with timestamps. We excluded flow records between campus hosts and non-campus hosts to study the performance of our technique on worm propagation inside an intranet. The resulting trace has about 1.4 million flows involving 8040 campus hosts.

With the four hour trace serving as real-world background traffic, we add flow records to the trace that represent worm-like traffic with varying scanning rates. We vary the fraction of vulnerable hosts F, by randomly selecting the desired fraction of hosts from the set of 8040 total internal hosts. For the following experiments, except

	Worm inter-scan duration (second)					
Trace	10	20	30	50	75	100
Normalized worm rate	2	1	0.67	0.4	0.27	0.2
Total flows $\|E\|$ (million)	2.02	1.67	1.57	1. 49	1.43	1.42
Number of hosts infected	804	804	804	804	726	702
Fraction of attack edges	0.296	0.157	0.103	0.053	0.013	0.012
Optimal Δt (second)	400	800	1600	1600	1600	3200

Figure 3.12: Description of traces with different rate worm traffic artificially added into a real traffic trace collected from the backbone of a university network.

Section 3.7.7, we choose $F = 0.1$. Each worm outbreak starts roughly 2800 seconds into the trace, and lasts for 8000 seconds. Once a host is infected, it generates one attack flow every t seconds to a randomly selected destination from among the 8040 hosts. In the real trace, 90% of the hosts send fewer than one flow every 20 seconds. To describe how aggressive a worm is, we define the *normalized worm rate* as the ratio of the rate an infected host sends attack flows to the 90 percentile of the normal connection rate (e.g., a worm sending one flow per 20 second has a normalized worm rate of 1, and a worm sending one flow every 200 seconds has a normalized rate of 0.1). Figure 3.12 lists the characteristics of the worms we introduced to the real world trace. We use "Trace-x" to refer a trace with worm rate of one attack flow per x seconds.

We introduce two additional metrics to compare the performance across worms of different scanning rates. Given the set of the top Z frequency edges after sampling, the *detection accuracy of causal edges* is the number of causal edges in the set divided by Z, and the *detection accuracy of attack edges* is the number of attack edges in the set divided by Z.

For each experiment, we use the parameter values selected from Figure 3.12, and discuss how we compute the optimal parameter values in Section 3.7.4. We repeat each experiment run 5 times with each run consisting of 10^4 walks (unless otherwise specified) and plot the mean of the 5 runs for the following results.

3.7.1 Detecting the Existence of an Attack

To determine whether the random moonwalk technique can detect if an attack is present, 10^4 random moonwalks were performed on Trace-10. Figure 3.13 shows the number of times each edge was sampled, and the outline of the plot indicates the count of the most frequently sampled edge for each second. The dashed lines indicate the actual attack start time, saturation time, and the attack finish time. The

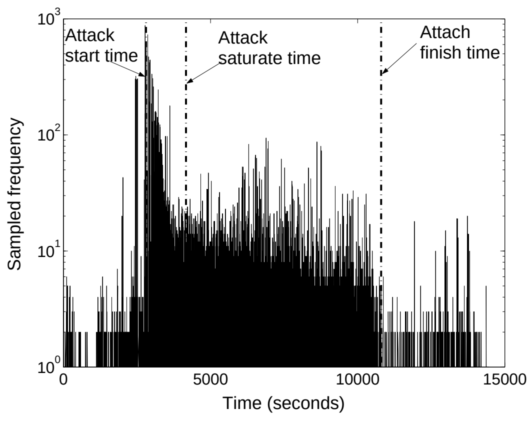


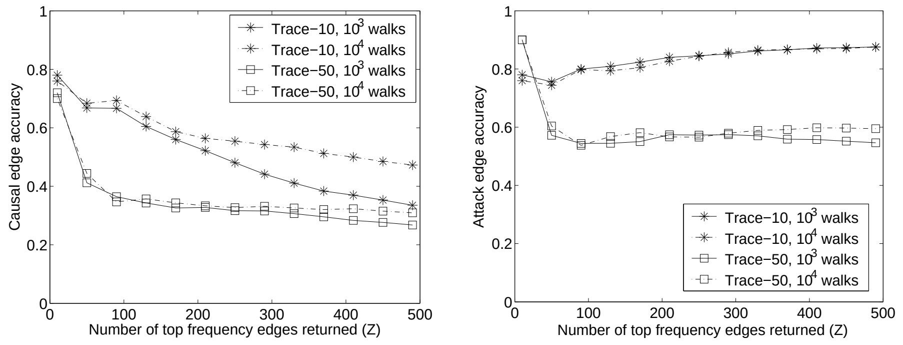
Figure 3.13: Stem plot of edge frequency counts with $W = 10^4$ walks on Trace-10.

figure shows that edges occurring before and after the attack have a relative evenly distributed sampling frequency. Edges between time 2700 and 10000 are sampled more frequently, with a peak frequency as high as 800. This strongly suggests the existence of abnormal structures in the host contact graph, which may potentially constitute an epidemic spreading attack.

In particular, the peak of the frequency counts occurring around 2800 seconds corresponds to the onset of the attack (the worm was introduced at $T_0 = 2807s$) with initial causal flows having highest probability of being traversed. The turning point after the peak (4200 seconds in this case) corresponds to the attack saturation time when all vulnerable hosts are infected. Knowledge that an attack is taking place and the information on precisely when it started is useful to network operators, and could be used to focus resources (such as random moonwalks) on the portions of the trace that are most likely to yield information about the attack origin.

3.7.2 Identifying Causal Edges and Initial Infected Hosts

We first examine the detection accuracy of causal edges and the size of the suspect set identified for further investigation. Figure 3.14 (a) shows the detection accuracy, varying the number of top frequency Z edges, with different number of walks. First, we observe random moonwalks achieve high detection accuracy of causal edges, in particular when Z is small. Although there are only 800 causal edges out of the approximately 1.5-2 $\times$ 10^6 flows, as high as 7-8 out of the top 10 flows are causal flows, regardless of the worm propagating rate. Second, the causal edge accuracy decreases sub-linearly as we increase Z, demonstrating the capability of finding causal flows beyond the few initial ones. These edges may additionally reveal the attack propagation



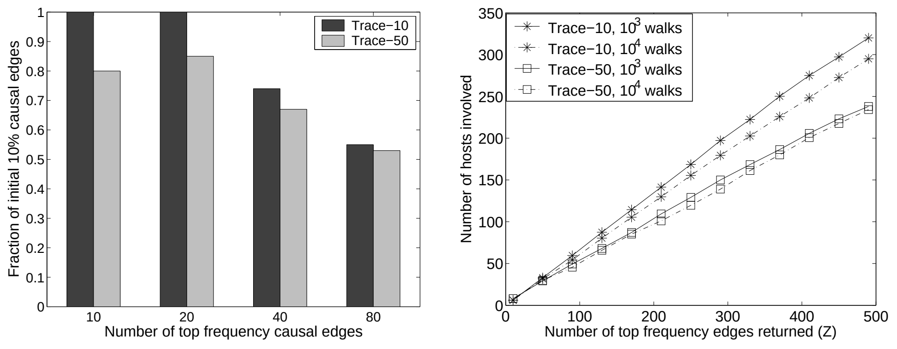
(a) Causal edge accuracy (b) Attack edge accuracy

Figure 3.14: Detection accuracy of causal edges and attack edges vs. number of top frequency edges (Z) returned for Trace-10 and Trace-50. Note there are only 800 causal edges from among approximately 1.5-2 $\times 10^6$ total flows.

paths, and help reconstruct the causal tree. Finally, increasing the number of walks results in higher causal edge accuracy in general, but a small number of samples can already achieve comparable performance when we focus on the small number of top flows, i.e., when $Z \leq 100$. As a contrast, we show the detection accuracy of attack edges in Figure 3.14 (b). We find that as expected the accuracy of attack edges is fairly high. But a high detection accuracy of attack edges does not always imply high detection accuracy of causal edges. For example, the attack edge accuracy for Trace-10 increases with larger Z, while the causal edge detection accuracy decreases. In Section 3.7.5, we will further address the comparison between causal edge and attack edge accuracies with alternative edge selection strategies.

We proceed to examine whether the detected causal edges correspond to the initial causal edges. We focus on the initial 80 causal flows (10% of the total causal flows) in the attack and plot the fraction of such flows among the actual returned causal edges in Figure 3.15 (a). As expected, the majority of the causal flows actually detected correspond to the initial ones that can be traced back to the attack origin, confirming the results in our analytical study.

Given the selected top frequency flows, we examine how many hosts are involved with initiating these flows. Since the identified flows are likely to be top level causal flows, these hosts are good candidates as hosts on the top level causal tree that can be chosen for further investigation. We assume that the source host of every selected flow is potentially the worm origin, and plot the total number of such hosts as we vary the number of selected flows Z in Figure 3.15 (b). These numbers thus give an upper bound on the amount of further effort required for worm origin identification (without explicitly exploiting the structure of the graph composed of the selected



(a) (b)

Figure 3.15: (a) Fraction of initial causal edges among the actual returned causal edges. (b) The number of source hosts involved as suspect top level hosts vs. number of top frequency edges (Z) returned.

flows). Although the number of hosts grows linearly as Z increases, the slope is less than one, suggesting the existence of a small number of sources contributing to a large number of flows. For example, after 10^4 walks, if we plan to use the top 50 flows for reconstructing the top level causal tree, we will have in total only 30 source hosts out of the 8040 hosts even with a slowly propagating worm that generates one scan per 50 seconds. In the next section, we show how the structure of the graph composed of these returned high frequency flows can additionally help to identify the worm origin.

3.7.3 Reconstructing the Top Level Causal Tree

Once we obtain the worm origin suspect set and the Z selected flows, a number of methods could be used to pinpoint the exact attack source. Potential methods include correlating the contents or sizes of the selected flows, or using additional out-of-band information regarding the set of infected hosts. Alternately one can exploit the structure of the graph composed of the Z flows. We simply take the 60 top-frequency flows selected from Trace-50 after 10^4 walks and construct a graph of these flows (Figure 3.16). The flow graph constructed from Trace-20 is shown in the Appendices (Chapter 6.2).

The artificially introduced worm in Trace-50 starts at host 8033, and each infected host sends only one attack flow every 50 seconds. Among the top 60 flows found by random moonwalks and shown in Figure 3.16, there are 35 causal flows and 17 flows that carry attack traffic but are not the flows that actually caused their destinations to become infected. The random moonwalks identify host 8033 as the actual worm origin and show the large tree branching structure below it. We also observe quite a few flows

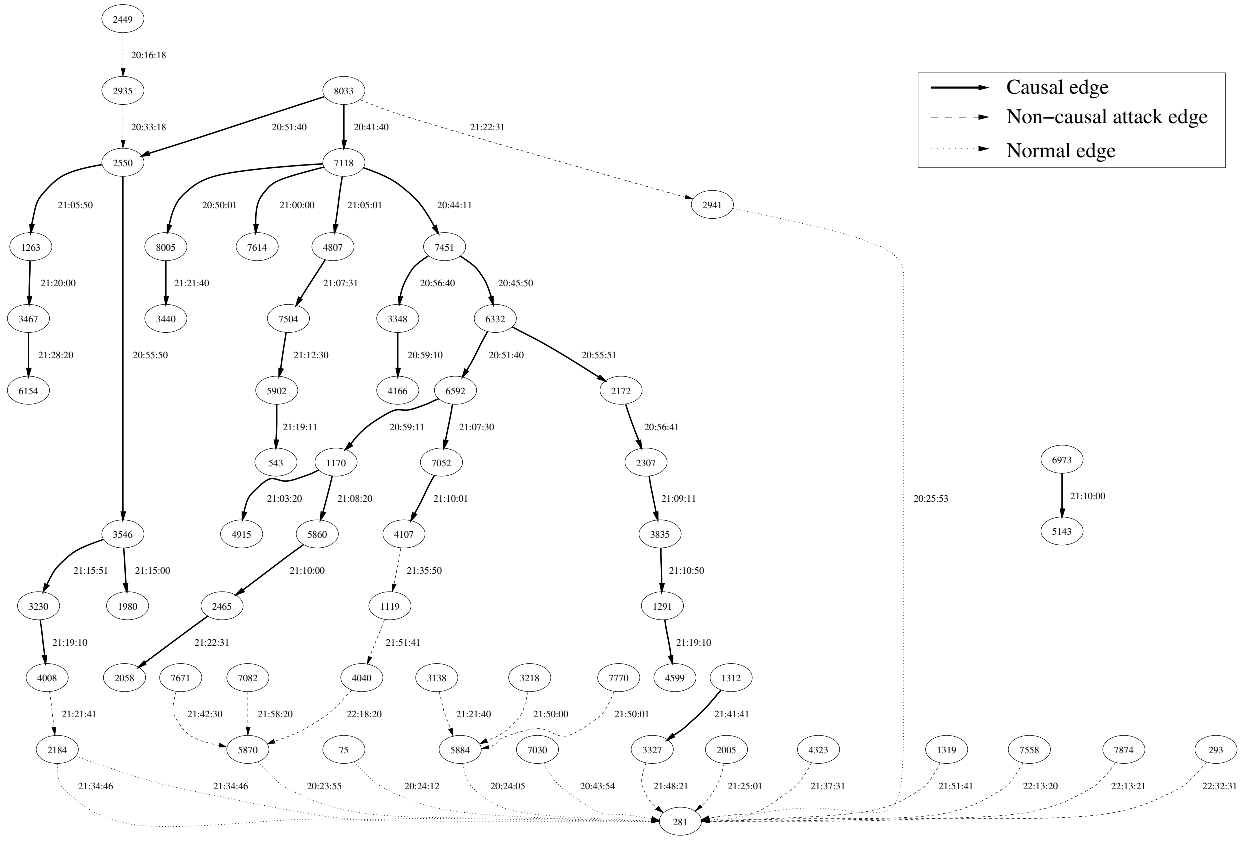


Figure 3.16: Graph of the 60 top frequency flows returned by the random moonwalk algorithm when run on Trace-50. Note the graph is neither the host contact graph, nor the causal tree. Hosts are represented by circles annotated with the host ID. Flows are represented as directed arrows between hosts, and are annotated with timestamps. Solid arrows denote causal edges, dashed arrows denote non-causal attack edges, and dotted edges correspond to normal traffic flows.

with destination host 281. It turned out that in the background trace we collected, host 281 was infected by some variant of the Blaster worm [11], and it generates scans with a peak rate of 72 flows per second. Manual investigation into the real trace revealed no successful infection events associated with such scan traffic. As a result, there is no causal tree actually induced by host 281. However, due to the high scanning rate, the few flows sent to host 281 are frequently selected by random moonwalks that trace back to host 281, and this explains why these normal flows to host 281 appear. Even though there is unrelated aggressive scanning taking place, the random moonwalks still cull out the top levels of the causal tree automatically. Such results show the effectiveness of random moonwalks at extracting the tree structure of slow worm propagation patterns (in our example, one scan every 50 seconds) to identify the worm source, even in the presence of aggressive scanners and other pathological background traffic events. As future work, we can pursue refinement techniques to further improve the accuracy of identifying the worm origin(s) and to reconstruct the higher levels of the causal tree.

3.7.4 Parameter Selection

Given a network trace that may contain worm traffic, we need to select the best parameter values without prior knowledge of worm propagating characteristics. This section studies the performance impact of the input parameters d (maximum path length) and Δt (sampling window size). We use Trace-20 and Trace-50 as representative traces for the following study.

We first fix Δt to 800 seconds for both traces (800 seconds may not be the optimal value for each trace) and vary the maximum path length d in terms of hop counts. Figure 3.17 (a) shows the detection accuracy of the top 100 frequency edges (i.e., $Z = 100$). We observe that the detection accuracy for both attack edges and causal edges increases with longer path length. As discussed earlier in our analysis in Section 3.6.4, longer paths tend to walk across a larger portion of the attack tree. As we further increase the path length, the detection accuracy saturates as the path length of each walk is bounded by the start of the trace. A longer maximum path length improves detection accuracy, but also implies greater sampling overhead since more edges will be involved in each walk.

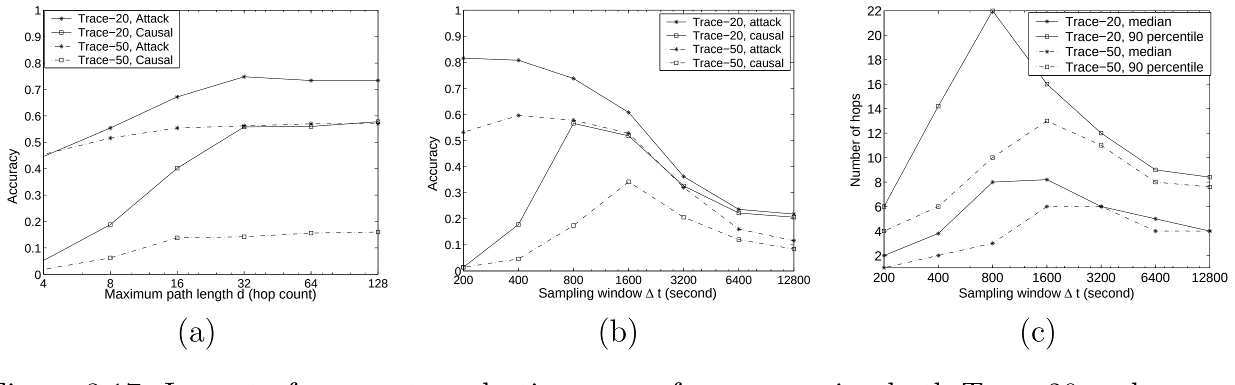


Figure 3.17: Impact of parameter selection on performance using both Trace-20 and Trace-50. (a) Detection accuracy vs. d (b) Detection accuracy vs. Δt (c) Actual path length vs. Δt.

Next, we vary the sampling window size Δt with the maximum path length d set equal to ∞ so each walk can continue as far as possible. Figure 3.17 (b) shows the impact of Δt on the detection accuracy of the 100 top frequency edges. In both traces, when we increase Δt, the detection accuracy of the causal edges first increases and then decreases. The detection accuracy of attack edges, however, is highest for smaller Δt's and becomes lower with a larger Δt. We also observe that with the slowly propagating worm in Trace-50, we need a larger Δt to achieve the best detection accuracy compared with the faster propagating worm in Trace-20.

To understand the reason, we show in Figure 3.17 (c) the variation of the actual path lengths (in terms of hop-count) with Δt. When Δt is small, walks terminate at shorter path lengths, as a walk is more likely to reach a host that received no flows within the previous Δt seconds. While shorter walks cannot reach the top levels of the causal tree, they are more likely to stumble across attack edges at lower levels, resulting in high detection accuracy for attack edges but low accuracy for causal edges. Increasing Δt gives a random moonwalk a greater chance to traverse top level edges, in particular the causal ones, but these long paths also involve more normal flows since they can walk backward to before the start of the attack, reducing the number of attack edges involved. Thus the detection accuracy of causal edges increases while that of attack edges decreases. Finally, further increasing Δt has a negative impact on the actual lengths of walks as each walk tend to be shorter by jumping across a larger portion of the trace every step. The walks also involve more normal traffic, since attack flows are generally clustered in time and a large Δt can skip over large portions of the attack. As a result, we observe low detection accuracy for both types of edges when Δt is too large.

For both Trace-20 and Trace-50, we achieve the best detection accuracy for causal edges when actual path lengths are maximally long. For worms that generate flows with a slower rate, a larger Δt maximizes the actual path lengths and achieves better performance.

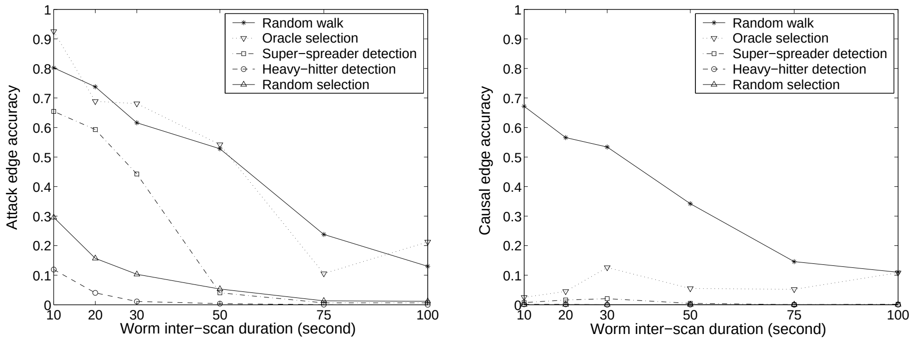
In summary, given a trace with unknown worm properties, the best sampling performance is obtained by choosing the Δt that gives the longest actual path lengths, in terms of number of hops that the moonwalks traverse. For all our experiments, we used the above guideline to choose an optimal Δt for each trace (see Figure 3.12). An adaptive version of random moonwalk sampling could launch walks with different values of Δt and choose one that maximizes the observed path lengths.

3.7.5 Performance vs. Worm Scanning Rate

In this experiment we compare the random moonwalk algorithm with other common methods for identifying potentially anomalous behavior, while varying the rate at which infected hosts scan new victims. Again, we use the detection accuracy of both causal and attack edges as our performance metrics, and we compare the following five techniques:

- **Random moonwalk selection:** Pick the $Z = 100$ edges with the highest frequency after performing 10^4 random moonwalks.
- **Heavy-hitter detection:** Find the 800 hosts that generated the largest number of flows in the trace (the "heavy-hitters"). Randomly pick 100 flows between two heavy-hitters. (We select 800 hosts as we know there are about 800 infected hosts in the traces.)
- **Super-spreader detection:** Find the 800 hosts that contacted the largest number of distinct destination hosts (the "super-spreaders"). Randomly pick 100 flows between two super-spreaders.
- **Oracle selection:** Assume an oracle that identifies the set of infected hosts with zero false positive rate. The oracle randomly selects 100 flows between these hosts.
- **Random selection:** Randomly pick 100 flows from each trace.

Both heavy-hitter and super-spreader heuristics have been traditionally used to detect patterns of malicious activity in IDSes [83, 75].



(a) Attack edges (b) Causal edges

Figure 3.18: Detection accuracy vs. worm scanning rate. The X-axis represents the worm inter-scan duration. For example, a window of $x = 20$ means an infected host generates an infection flow every 20 seconds.

As expected, the detection accuracy for attack edges decreases with an increased worm inter-scan duration (Figure 3.18 (a)), since a worm that sends attack traffic at a slower rate will create fewer attack edges in the host contact graph. Random moonwalk selection and oracle selection have similar performance and perform substantially better than the other strategies. Perhaps surprisingly, heavy-hitter detection performs even worse than random selection, as the heavy-hitter method is likely to select servers, and most of the communication between servers is legitimate traffic.

The real success of the random moonwalk algorithm, however, is not in picking attack edges. Rather it lies in its ability to extract causal edges from a large noisy host contact graph. This is evident from Figure 3.18 (b), where we notice that all other techniques, including oracle selection, have a low detection accuracy for causal edges across all worm scanning rates. For attacks that spread at rates of one scan every 10-30 seconds, the causal edge detection accuracy of random moonwalk selection is greater than 0.5, implying that roughly 50 out of the top 100 edges are always causal edges. This establishes the capability of finding the causal edges by globally correlating the host traffic patterns for very stealthy attacks using the random moonwalk algorithm. On the other hand, the poor performance of even the oracle selection suggests that detecting infected hosts alone does not help extracting the causal edges to reconstruct the top level causal tree and trace back the worm origin.

3.7.6 Performance vs. Worm Scanning Method

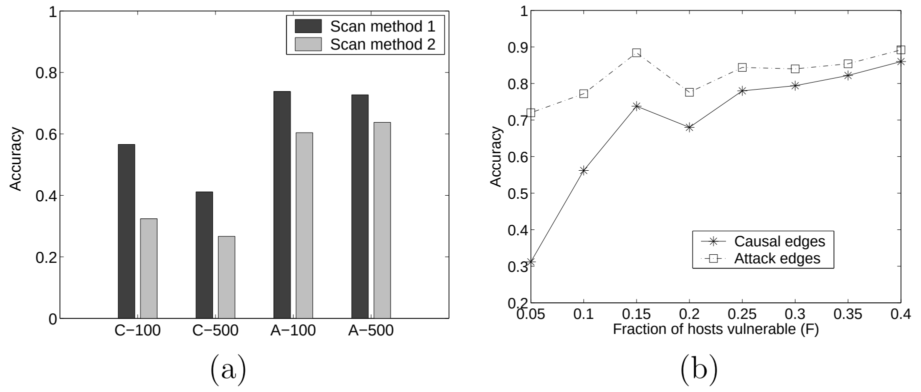


Figure 3.19: (a) Comparing detection accuracy using worms with different scanning methods using Trace-20. (b) Comparing detection accuracy using worms targeting different fraction of vulnerable hosts F.

In this experiment, we study the effectiveness of random moonwalks using worms with different scanning methods. Since many existing techniques identify worm scanners by looking at only flows sent to non-existent hosts [113, 44], a smart worm can evade such detection by carefully targeting only valid addresses. We therefore evaluate the performance of our technique using two worms with different scanning methods. The first scanning method randomly scans only valid host addresses, while the second method randomly scans both existent and non-existent host addresses with 50% of host address space being used. For both worms, an infected host starts scanning at the rate of one attack flow every 20 seconds.

Figure 3.19 (a) compares the detection accuracy of the top $Z = 100$ and $Z = 500$ frequency edges for the two different worms. For both causal edges (represented by C-100 and C-500) and the attack edges (represented by A-100 and A-500), random moonwalks achieve better detection accuracy for the "smart-scanning" worm, which is consistent with our analytical study in Section 3.6.2. As random moonwalk sampling identifies the subtle global tree patterns of worm propagation, instead of relying on the scanning behavior of each specific infected host, it is inherently more robust to other worm scanning strategies [97, 113]. Such results are also encouraging for detecting those worms that may evade detection techniques employed by many existing scan-detectors, which essentially use the number of connections to unused address chunks as a metric of interest [44, 88, 47].

3.7.7 Performance vs. Fraction of Hosts Vulnerable

This section studies the performance of the random moonwalk algorithm with different fraction of hosts infected (i.e., we vary F). With a greater number of hosts infected by an attack, the degree of anonymity provided to the true attacker is also greater. In this experiment, we fix the worm scanning rate to be one attack flow per 20 seconds, and vary the fraction of hosts vulnerable F during each attack. Figure 3.19 (b) shows the performance in terms of the detection accuracies of both causal edges and attack edges. Within the range of $F = [0.05, 0.4]$, we observe that the detection accuracies increase as we increase the fraction of hosts infected. Empirically, our experiments also show that the detection accuracy increases for more slowly propagating attacks (e.g., one scan per 50 seconds) as they infect more hosts in the network along time. We plan to further quantify the impact of F on performance as future work.

3.8 Simulation Study

The goal of our simulation study is to evaluate the effectiveness of random moonwalks using different background traffic models of normal host communication. Our hypothesis is that the simplified traffic model in our analytical study, where background (i.e., normal) traffic, modeled as uniform scanning, is a worst case model for performance of our algorithm. Realistic host contact graphs tend to be much sparser, meaning the chance of communication between two arbitrary hosts is very low since host connectivity patterns usually display locality in the set of destinations contacted. An epidemic "tree" structure will more easily stand out in such scenarios, and thus be detected with higher accuracy.

In particular, we model the host connectivity patterns in terms of both the out-degree of normal hosts and the connection locality. The out-degree of each normal

Trace	1	2	3	4	5
Host out-degree D	$\|H\|$	$C < \|H\|$	$\|H\|$	Power-law	Power-law
Connection locality	Uniform	Uniform	Power-law	Uniform	Power-law
Causal edge accuracy	0.506	0.472	0.546	0.616	0.614

Figure 3.20: Detection accuracy of causal edges using different background traffic models. “Power-law” means the controlled variable follows a power-law distribution (with a coefficient set to 3). “Uniform” means the controlled variable follows a uniform distribution. $|H|$ denotes the total number of hosts in the network, and C is a constant number smaller than $|H|$.

host is the size (denoted as D) of the *contact set*, which represents the set of destinations the host originates flows to under normal circumstances. Connection locality is modeled by assuming each host selects destinations preferentially (within the *contact set*) according to either a uniform or power-law distribution. Figure 3.20 lists the background traffic generated using different combinations of the host out-degree and connection locality. All the simulations run with $|H| = 10^4$ nodes for 3000 seconds of simulated time. We introduce worm attacks lasting 500 seconds with a fixed propagating rate ($A/B \simeq 7$) that infect $F = 0.1$ fraction of hosts. Recall that A is the connection rate of an infected host (including normal connections), and B is the connection rate of a normal host. The resulting traces have about 10^6 total flows with 1000 causal flows. For each trace, we perform 10^4 random moonwalks and compute the detection accuracy of causal edges among the returned top $Z = 100$ frequency flows.

Overall, the random moonwalks achieve high detection accuracy across all background traffic models. As expected, the power-law distribution of the host out-degree results in best performance as the corresponding normal host contact graphs are sparse. The power-law distribution connection locality has similar performance impact since each host tends to talk only to a few hosts within the contact set more frequently, resulting in a relatively sparser host contact graph too. In contrast, uniform destination selection with constant contact set size (i.e., $D = C$, or $D = |H|$) models random scanning background traffic, and yields the worst performance.

3.9 Random Moonwalks and Spectral Graph Analysis

Random Moonwalks exploits the global tree structure of attack propagation via a special way of sampling. While each next step in a walk is selected randomly from the possible choices, the aggregated effect is not random, but deterministic in essence. We show in this section that the random moonwalk algorithm can be interpreted as performing spectral graph analysis on a *flow graph* transformed from the host contact graph that we defined earlier.

Spectral graph analysis is a popular method in social science and information retrieval by exploiting the underlying graph structure of social or information entities. Such method studies graphs by their adjacency matrices or Laplacian matrices [22]. The graph structures are shown to have tight relationships with the eigenvalues and the eigenvectors of the associated matrix. Such properties can be used for clustering or ordering nodes on the graphs. For example, existing works have performed link analysis [53, 81] to infer correlations among social network entities from their activity patterns. In Web graph analysis, a number of eigenvector methods, such as Pageranks [73] and HITS [50], have shown to be effective for identifying high quality pages for Web search.

Random moonwalks have strong connections with spectral graph analysis, in that it can be regarded as a simulation method to compute the largest eigenvector of a special type of flow graphs. The underlying structure of a graph can be analyzed using its largest eigenvector, in order to identify the existence of abnormal host communication patterns. In particular, the largest elements in the principal eigenvectors correspond to those abnormal events that directly or indirectly cause the structural changes of host communication graphs. Such a spectral analysis view offers better insight to the effectiveness of the algorithm under different attack models, and raises opportunities for more optimized computation. We explain the details next, starting with a flow graph model, with which we perform the spectral analysis.

3.9.1 A Flow Graph Model

Graphs are a common method for representing the relationships among various entities. In this section, we introduce a different graph representation to revisit the previously described random moonwalk algorithm. With this new graph model, random moonwalks can be interpreted as a sampling process for computing the largest eigenvector of the corresponding adjacency matrix.

We adopt the same notion of flows as discussed in Section 3.4. Similar to the host contact graph, the flow graph is defined as a direct graph $G_f = \langle V, E \rangle$. Each node $n \in V$ on the flow graph corresponds to a network flow $e = \langle u, v, t^s, t^e \rangle$. Therefore, in the rest of this section, we use n instead of e to denote a flow, and use the term *node* and *flow* interchangeably.

Edges on the flow graphs capture the potential causal relations between flows. Given two nodes (i.e., two flows) $n_1 = \langle u_1, v_1, t_1^s, t_1^e \rangle$ and $n_2 = \langle u_2, v_2, t_2^s, t_2^e \rangle$, we define

Flow-ID	Source	Destination	Start-time	End-time
n1	A	B	10	20
n2	E	B	20	30
n3	B	C	25	30
n4	B	D	35	40
n5	B	C	45	60

Figure 3.21: Five example network flows. A, B, C, D, and E are five different hosts in the network.

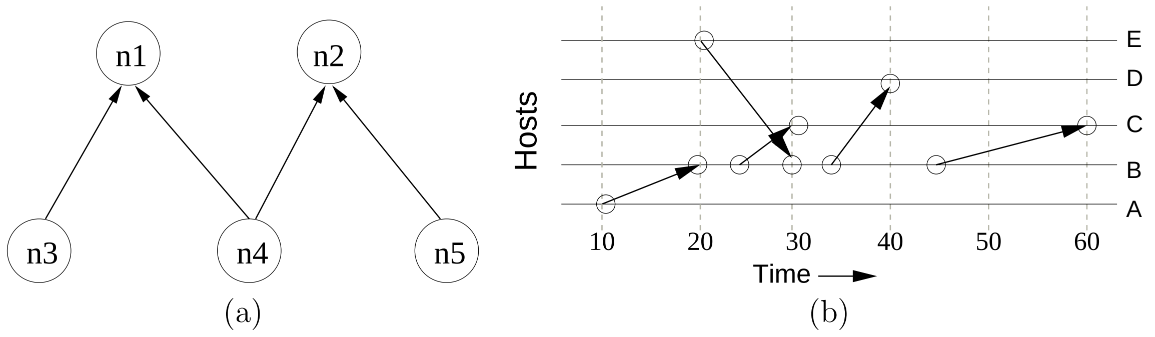


Figure 3.22: (a) The flow graph defined by the five network flows listed in Figure 3.21. (b) The host contact graph defined by the same five network flows.

a directed edge from n_2 to n_1, if $v_1 = u_2$ and $t_2^e \leq t_1^s < t_2^e + \delta t$, where δt is an input parameter used to generate the flow graph. Intuitively, the directed edge from node n_2 to n_1 reflects a potential causal relationship between the two corresponding flows, meaning flow n_2 might potentially be induced by flow n_1. The parameter δt represents the length of the time period, during which these two flows can be causally related. We shall see later that, for spectral analysis purpose, δt exactly corresponds to the sampling window size Δt defined in the random moonwalk process, i.e., $\delta t = \Delta t$.

As an example, Figure 3.21 lists five hypothetical network flows involving five different hosts A-E. Figure 3.22 (a) shows the corresponding flow graph defined by the five network flows with $\delta t = 20$ seconds. We also plot the corresponding host contact graph (see Section 3.4) defined by the same five network flows in Figure 3.22 (b). There are several significant differences between the two types of graphs. First, the main entities in the flow graph are network flows, while the main entities in the host contact graph are states of hosts at different time stages. Second, since each network flow is associated with a unique start time and end time, we no longer need to encode time explicitly in the flow graph. Finally, a flow graph is defined with an additional input parameter δt, while in the host contact graph, the notion of the sampling window size Δt is meaningful only when we perform the random moonwalks on the graph. Thus, the host contact graph and the associated flow graphs have a one-to-many correspondence by varying the values of Δt (hence δt).

Under this framework, it is easy to see how we can generate a flow graph, given an already defined host contact graph. First, the edges on the host contact graph and the nodes on the flow graph have a one-to-one correspondence. Once we determine the sampling window size Δt for random moonwalks on the host contact graph, we can generate a corresponding flow graph, by inserting an edge from node (i.e., flow) n_2 to n_1, if it is possible for us to moonwalk from flow n_2 to n_1 in the host contact graph.

3.9.2 Markov Random Walks on the Flow Graph

A flow graph defined using the above method can be considered as a Markov chain network, where each node corresponds to a state in the Markov chain, and each edge corresponds to a possible state transition. A random moonwalk on the host contact graph thus corresponds to a Markov random walk on the flow graph. At each state, the probability of transiting to a next downstream state will be evenly distributed among all possible downstream states, due to the randomness of the next step selection in a moonwalk. With such Markov random walk interpretation, the normalized adjacency matrix A_f defined by a flow graph G_f, is a stochastic matrix whose row sums are all $\mathbf{1}$. Let us now examine the spectral problem for the matrix A_f, namely the solutions to the equation:

$$A_f x = \lambda x$$

In order to ensure that random walks on the Markov flow graph G_f converges to a unique stationary distribution π_0, i.e., $A_f \pi_0 = \lambda \pi_0$, G_f needs to be irreducible, ergodic, and aperiodic [109]. However, because of the backward direction of time in performing the walks, the above defined flow graph G_f is not irreducible. In other words, it's possible to transit from a node (flow) n_i to another node (flow) n_j, if and only if flow n_j finishes earlier than the launch time of flow n_i, not vice versa.

We can apply a commonly used graph transformation method [73] to overcome the above problem without changing the random walk properties. The method works by inserting an additional directed edge from every *sink*, i.e., node without outgoing edges, to every other node on the graph. It is trivial to see that the resulting graph G'_f meets all three requirements (irreducible, ergodic, and aperiodic), and is therefore guaranteed to have a stationary distribution π_0. Since each random moonwalk starts with a randomly selected flow on the host contact graph, a Markov random walk on the original flow graph G_f is equivalent to a random walk on the transformed graph G'_f.

The stationary distribution π_0 of the transformed Markov flow graph G'_f, by its definition, represents the probability of reaching every node in a global stable state. Such stationary distribution can be estimated by the simulation method where we repeatedly perform random walks on the flow graph. We have shown that random walks on the flow graph are equivalent to random moonwalks on the host contact graph. One can therefore view random moonwalks on a host contact graph as a Monte Carlo sampling method to compute the stationary distribution of a corresponding flow graph. It has been shown that this stationary distribution equals to the largest eigenvector of the corresponding graph's normalized adjacency matrix. The highest probability flows being traversed in a random moonwalk therefore correspond to those nodes with largest numerical values in the largest eigenvector. These flows, with high probability, are the initial causal flows on the causal tree.

Such a spectral analysis view opens up the following two possibilities: First, the computation of a graph's largest eigenvector is a well-known problem. We can systematically explore more efficient methods to perform deterministic computation. Second, since the largest eigenvectors have strong connections with the underlying graph structures, we may be able to gain more insights on how different attack models will impact the flow graph structures and the detection performance consequently.

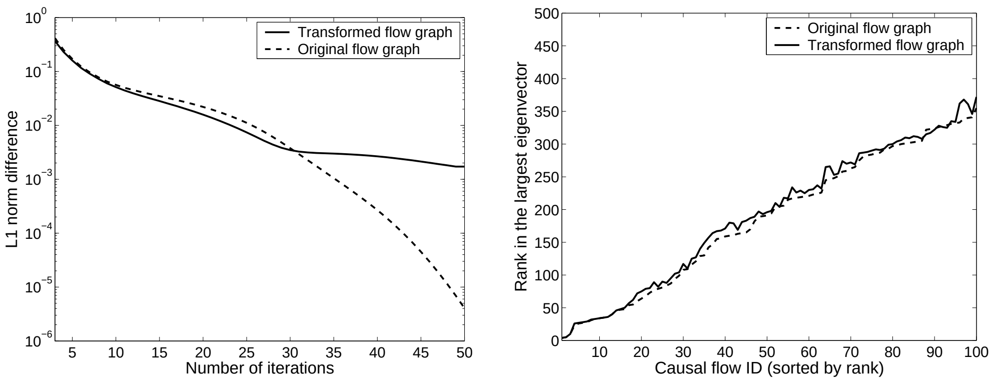
3.9.3 Deterministic Computation of Probabilities

A commonly used algorithm to compute a graph's principle eigenvector is matrix power method [38]. It finds the largest eigenvector π_0 of a matrix A using iterative matrix multiplications.

The method starts with a random normalized vector z as a initial guess for π_0. We then repeatedly multiply z with matrix A, and normalize the resulting vector $y = Az$ as another new start z, until z eventually converges to π_0. With matrix additions and multiplications, the complexity of such power method in both space and time is $O(|E|^2)$, where $|E|$ is the total number of nodes (i.e., flows) on the graph. Such quadratic complexity is very expensive in practice, as $|E|$ is on the order of millions for even a small campus network with thousands of hosts.

To optimize both space requirement and computational effort, we may apply compression techniques such as [24] on very large graphs for more compact representations, or exploit locality of matrix entries and efficient data structures for parallel and distributed computation [39, 14, 37]. We leave it as future work to extensively explore these techniques for our specific application.

Empirically, the adjacency matrix A_f of the original flow graph G_f can be used to approximate A'_f of the transformed flow graph G'_f in the matrix power method to obtain a similar converged solution, especially for the highest ranked elements



(a) (b)

Figure 3.23: (a) The convergence rate of the power method on both the transformed flow graph and the original flow graph. (b) The ranks of initial causal flows in the largest eigenvector. The causal flows are sorted based on the rank obtained from the original flow graph.

of our interest. Figure 3.23 (a) compares the convergence rate computed as the L_1 norm differences of the resulting vectors after each iteration, using both the original graph and the transformed graph obtained from our campus network trace with a normalized attack rate of 1. Correspondingly, we show the ranks of the initial causal flows in the largest eigenvectors computed based on the two different flow graphs in Figure 3.23 (b). Using both graphs, the solutions converge quickly in about 20-30 iterations. After 30 iterations, the resulting vectors computed from the original flow graph converge more quickly due to the existence of sinks, which do not re-distribute cumulated probabilities. Overall, the rankings of the initial causal flows on both graphs are very close to each other. Since the initial causal flows are more likely to be sinks, they get slight higher probabilities from the original graph, and thus slight better ranks (i.e., lower ranks) as shown in Figure 3.23 (b).

Note that the adjacency matrix of the original flow graph is triangular because of the uni-direction of state transitions. As future work, we can exploit this property, together with application semantics, to further optimize the power method computation for our special case.

3.9.4 Sampling Based Approximation

In Section 3.6.3, we have estimated the number of samples (i.e., walks) required verses the false positive rate of identifying the top level initial causal flows. A more general question is how accurate we can approximate the largest eigenvector using the Monte

Carlo simulation method? Existing studies [13, 5] on the convergence rate of Monte Carlo algorithms have mostly focused on Web graphs. Given a total of n nodes on the graph, to ensure a small variation of the target value for every node, the number of samples required is on the order of $O(n^2)$, based on a rough estimate in [13]. For the forensics application of identifying initial causal flows, such complexity estimate is quite pessimistic, as our analysis and empirical results indicate that a small number of samples can already achieve good detection accuracy.

Let us consider a flow graph, where each flow e has a deterministic probability of $P(e)$. We would like to know how many moonwalks W are required to be $1-\epsilon$ confident that the estimated probability $\hat{P(e)}$ is within a predetermined margin of error? We do so by estimating the standard deviation $\sigma(e)$ of the corresponding random variable $\mathbb{P}(e)$. We have already shown in Section 3.6.3 that for large W, $\mathbb{P}(e)$ can be approximately by a normal distribution, so that

$$\sigma(e) = \sqrt{\frac{P(e) \times (1 - P(e))}{W}}$$

To ensure $\sigma(e) \leq P(e)/k$, where k is a parameter quantifying the error margin, we need

$$W \leq (1 - P(e)) \times k^2/P(e)$$

The above estimate is general and applicable to all connected graphs. For our specific application, there appear to be two factors that aid in the convergence rate of the simulation process. First, the larger the deterministic $P(e)$ is, the faster the convergence rate will be with a smaller W. The tree structured worm propagation suggests that the small number of initial causal flows are among the highest ranked flows in the largest eigenvector, and thus have larger deterministic probabilities. Since these are the flows that we care, a small number of walks can already closely approximate their probabilities. Figure 3.24 (a) plots the flow probability distribution computed deterministically using the original flow graph, where there is a worm attack propagating at a normalized attack rate of 1. If we consider flows with $P(e) \simeq 0.01$ and $k = 10$, then we need about 10^4 moonwalks to approximate the probabilities of these high ranked flows.

The second factor that contributes to the small number of samples is that in our problem, it suffices to identify only the relative orders of the highest ranked flows and the algorithm will return a set Z of those flows. Suppose flow e_i is a initial causal flow that we are interested in finding, where i is the rank of that flow in the largest eigenvector. We can relax the error margin by reducing k, as long as $\hat{P(e_i)} \geq \hat{P(e_{i+q})}$ $(q \geq 1)$ is satisfied so that e_i will be present in the returned set. Using the same flow trace shown by Figure 3.24 (a), we plot in Figure 3.24 (b) the number of moonwalks

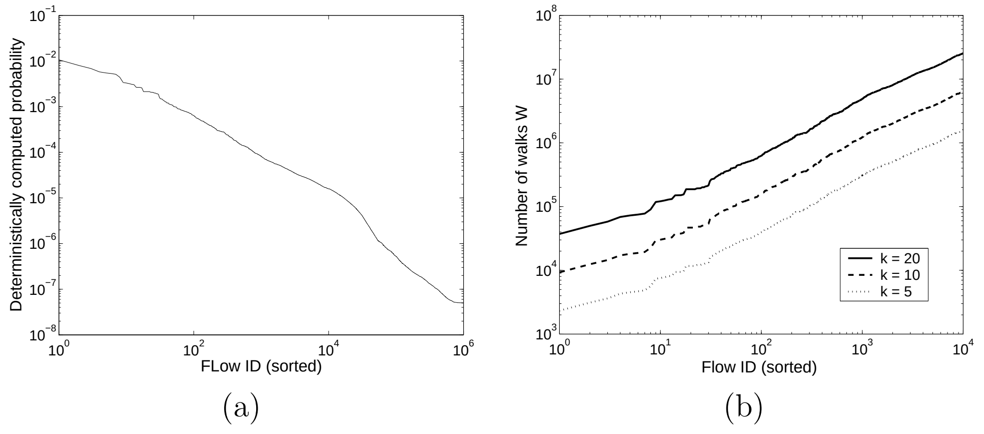


Figure 3.24: (a) Computed flow probability distribution with a worm attack. (b) Error margin parameter k vs. number of walks W.

W required by choosing different values of k. We observe that $10^4 - 10^5$ walks can lead to relative tight bounds of error margins ($k = 5, 10$) for the set of highest ranked flows.

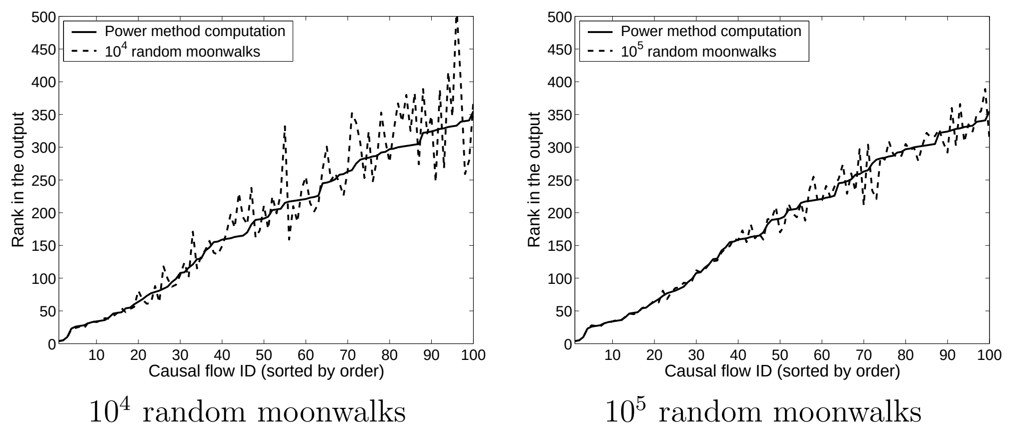


Figure 3.25: The ranks of initial causal flows computed deterministically compared against the ranks computed using random moonwalk sampling.

As a performance summary, Figure 3.25 plots the ranks of the initial causal flows computed by different numbers of random moonwalks, compared with the ranks that were computed deterministically using power method on the same flow graph. While increasing the number of walks to 10^5 reduces the variations of the sampled ranks, 10^4 walks can already approximate the 50 initial flows with good accuracy, and thus explains the overall high detection accuracies we have achieved with a small number of samples.

3.9.5 Revisit the Effectiveness of Random Moonwalks

Given a connected graph, the first several largest eigenvectors of the adjacency matrix have a wide spectrum of applications, for example, identifying important nodes for Web and social graphs [73, 50], segmenting images [61], and analyzing Internet topologies [36].

With an undirected graph, the largest eigenvector associated with the adjacency matrix' s principal eigenvalue inherently reflects the node degrees [36]. With a directed graph, however, the largest eigenvector expresses more than just the node degree [36]. The most similar usage to our application of largest eigenvectors on directed graphs, is Google's Pagerank computation, where each element in the largest eigenvector represents the eventual standing probability of a random Web surfer on the associated Web page.

Informally, two types of nodes (i.e. flows) are more likely to have higher ranks in the largest eigenvector of a directed graph: (1) high out-degree nodes, and (2) nodes that connect to high out-degree nodes. In the application of identifying the worm origins, the high out-degree nodes correspond to network flows that are *directly* responsible for creating a large number of new flows within the next Δt interval. Again, Δt is the sampling window size of random moonwalks. The second types of nodes, the ones connecting to high out-degree nodes, are flows that *indirectly* generated a large number of new flows. The notion of node out-degree is tightly associated with the size of Δt when we define the flow graph, so the node degree actually reflects the traffic rate induced by the flow into the network. Once we compute the largest eigenvector, those highest ranked flows are those that, directly or indirectly, induce a high aggregated traffic rate into the network subsequently. Such global effect is also what we observe during the propagation of a worm attack, and exploited by the random moonwalks.

In addition to degree, there is also a *normalization factor* that determines the ranks of nodes. This normalization factor is due to the use of *normalized* adjacency matrices. Intuitively, if there are a large number of flows that are equal likely to generate a same set of new flows, then the probability of each one being the causal event is small, resulting a low rank of every flow. Such normalization factor also explains why random moonwalks are effective in catching the *tree-structured* worm propagation patterns.

3.9.6 Impact of Attack Models on Performance

Since the largest eigenvector returned by random moonwalks depends on the structure of the corresponding flow graph, the next question is how attackers can evade detection, where the resulted flow graphs will look similar to a normal flow graph? In Section 3.7, we have shown how we can leverage the probability distribution of each flow being traversed in random moonwalks, which also corresponds to the distribution of elements in the largest eigenvector, to detect the existence of a random scanning worm attack. In this section, we explore different worm spreading strategies to study the applicable scope of our algorithm.

We take a simulated network trace as our background traffic, instead of using the real campus trace where there exists "normal" active scanners already. Both the host out-degree and contact set size follow power-law distributions, with the same setup as described in Section 3.8. Our normal host communication model contains no pathological patterns. We inject attack traffic propagating via different strategies into the simulation trace, with a target of compromising approximately equal number of hosts in the network for each attack. We then construct a flow graph using a computed optimal Δt.

Instead of performing random moonwalks, we deterministically compute the largest eigenvector of the resulting flow graphs using the matrix power method. The computed distributions and rankings are thus the theoretical optimal results that could be achieved with our algorithm. For each attack scenario, we compare the element distribution of the largest eigenvector resulted from worm traffic, against that with normal traffic only. We also compute the causal flow detection accuracy out of the 100 top frequency flows to study the algorithm performance.

Scanning Worms

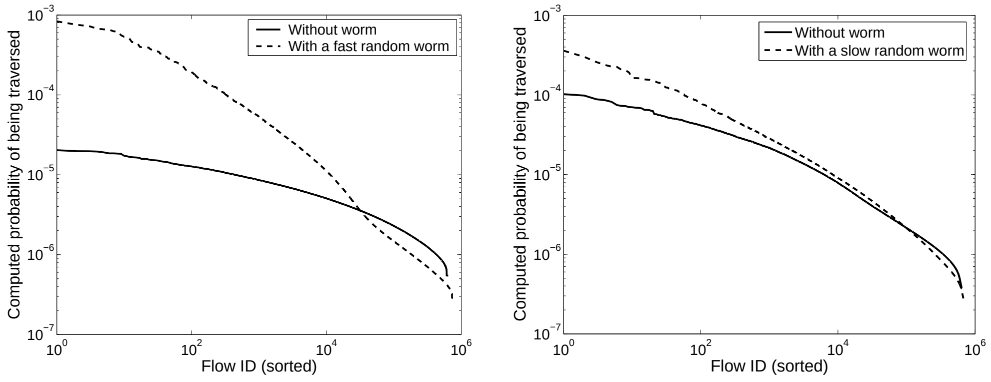
We first re-visit the random scanning worm model as our baseline study, where we inject two different epidemic spreading worm attacks with different rates into the trace. Each worm attack propagates with a fixed rate that infect $F = 0.1$ fraction of vulnerable hosts. For the slow worm, it needs 2000 seconds to infect all the vulnerable hosts (see Figure 3.26).

Figure 3.27 plots the probability distributions of each flow being traversed in random moonwalks, both with and without the attacks. The probability distribution, which is also the largest eigenvector distribution, becomes much more steep due to the worm traffic. With a faster propagating worm, we observe a more-biased distribution compared with that of a slower worm. This is as expected, since for the fast propagating worm, each initial causal flow indeed introduces higher aggregated traffic rate into the network, resulting in larger weights in the corresponding largest

Worm type	Random fast	Random slow
Fraction of hosts infected	0.1	0.099
Normalized attack rate	7	1
Scan duration (second)	500	2000
Optimal Δt (second)	70	400
Causal flow accuracy	57%	55%

Figure 3.26: The configurations of the two scanning worms.

eigenvector. We also note that the optimal Δ used for random moonwalks is smaller in the presence of a faster worm, confirming our analysis in Section 3.7.



(a) (b)

Figure 3.27: Probability distributions with and without random scanning worms. (a) The normalized attack rate is 7. (b) The normalized attack rate is 1.

With deterministic computation, the detection accuracies of causal flows are 57% and 55%, for fast and slow worms. They are very close to what we have achieved with 10^4 random moonwalks, which are 61.4% and 53%, respectively. These results re-iterate the effectiveness of our algorithm on random scanning worms, by leveraging the non-uniform distributions of the corresponding largest eigenvectors.

Scanning Hosts

Section 3.9.5 discussed that high out-degree nodes are also likely to have higher ranks in the largest eigenvector of a flow graph. Next, we examine the scenarios where, in the lack of tree-structured epidemic spreading attacks, how single scanning hosts would impact the probability distributions.

Scanner type	Random Scanner	Fast hitlist scanner	Slow hitlist scanner
Fraction of hosts infected	0.065	0.081	0.04
Normalized attack rate	90	7	3
Attack duration (second)	2000	2000	2000
Optimal Δt (second)	400	400	400
Causal flow accuracy	0%	0%	1%

Figure 3.28: The configurations of the three scanners.

With a single host spreading an infection, it is more difficult to compromise as many hosts as epidemic attacks. The single attack source has to be more aggressive by (1) scanning for longer time, (2) increasing the scanning rate, or (3) increasing the accuracy in hitting vulnerable hosts. We consider two types of scanning strategies. First, an attacker can randomly scan hosts with a more aggressive rate. While such type of attacks can easily be detected using existing methods such as [41, 83], we are more interested in how they will change the probability distribution of the largest eigenvectors due to certain flows' high out-degrees. Second, an attacker can pre-compile a list of vulnerable hosts, and scan these hosts only. The rate of such scans could be slowed down relatively to evade detection, while still allowing the malicious host to compromise a large number of hosts. We vary the rate of such scan in our experiment.

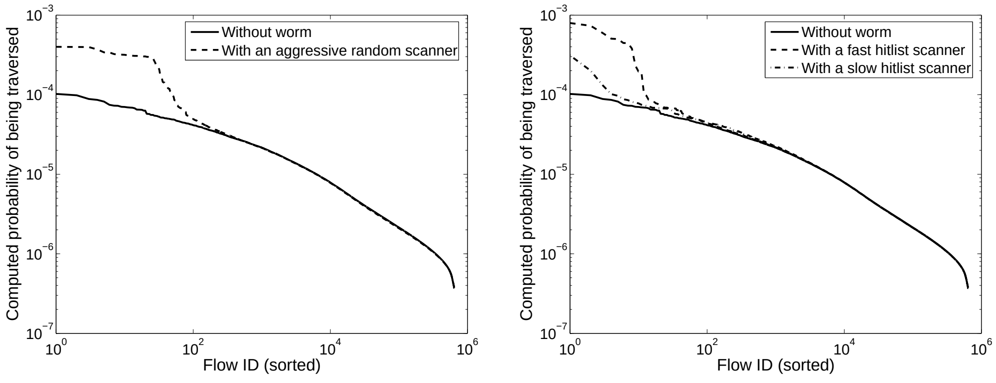


Figure 3.29: Probability distributions with and without random scanning hosts. (a) An aggressive random scanner (b) Scanners that spread infection via a pre-compiled hitlist.

Figure 3.28 shows the scanner configurations in the attacks we inject, and Figure 3.29 plots the probability distributions due to the two different types of scanning hosts. With an attack duration of 2000 seconds, none of the scanners can successfully infect all the vulnerable hosts.

Compared with the flow probability distribution without attacks, the probability distributions with both the aggressive random scanner and the fast hitlist scanner show noticeable changes, mostly due to the probability increases for the first 100 ranked flows. For the majority of lower ranked flows, there are no significant distribution changes caused by the aggressive scanning behavior. Further investigation confirms our hypothesis that, without the existence of epidemic structures in the flow graph, the highest ranked flows are indeed mostly high out-degree ones because of the high scanning rate. The slow hit-list scanner, in contrast, does not change the probability distribution as much due to its slow scanning rate. But it also failed to infect as many hosts as the more aggressive scanners. These are expected results, as our algorithm in essence identifies higher aggregate traffic rates built up in the network. For scanning attacks, the high out-degree flows are those flows preceding the causal ones, instead of being the causal flows themselves. So random moonwalks did not identify causal flows as high probability ones.

In summary, our algorithm can still identify the existence of aggressive scanners by detecting the changes of flow probabilities, if not causal flows directly. The stealthy scanners that spread slowly with pre-compiled hitlists may evade detection, but are not as efficient in compromising a large number of hosts. To detect these stealthy attacks, we might need additional information (e.g., attack signatures) or metrics (e.g., total number of hosts talked) for detection and forensic analysis.

Topologically Spreading Worms

Worms that spread topologically have caught greater attention recently [121]. Instead of random scanning the address space, the topological spreading worms infect hosts via pre-setup connections (e.g., use connections of P2P applications), or contact only known neighbors (e.g., through email address books). Since they infect hosts with valid addresses only, many techniques today such as [75, 82, 44, 88] will fail to detect these attacks by looking at connections to un-used address space. We again consider two types of strategies for spreading the infection topologically. For the first type of worm, each infected host randomly scans its neighbors in the contact set [1] with a high connection rate. We call it a *topological scanning worm.* For a second type of worm host, it sequentially contacts all the hosts in the contact set exactly once, and then stops the attack to avoid being detected due to a high scan rate. We define it as a *topological sequential worm.*

Figure 3.30 shows the experiment set up for the two different types of topological

[1]The neighbor set of a host will be defined by specific application semantics.

Worm type	Topological scanning	Topological sequential
Fraction of hosts compromised	0.086	0.086
Normalized attack rate	7	3
Maximum scans per host	$+\infty$	Size of its contact-set
Attack duration (second)	1000	625
Optimal Δt (second)	40	20
Causal flow accuracy	50%	75%

Figure 3.30: The configurations of the two types of topological spreading attacks.

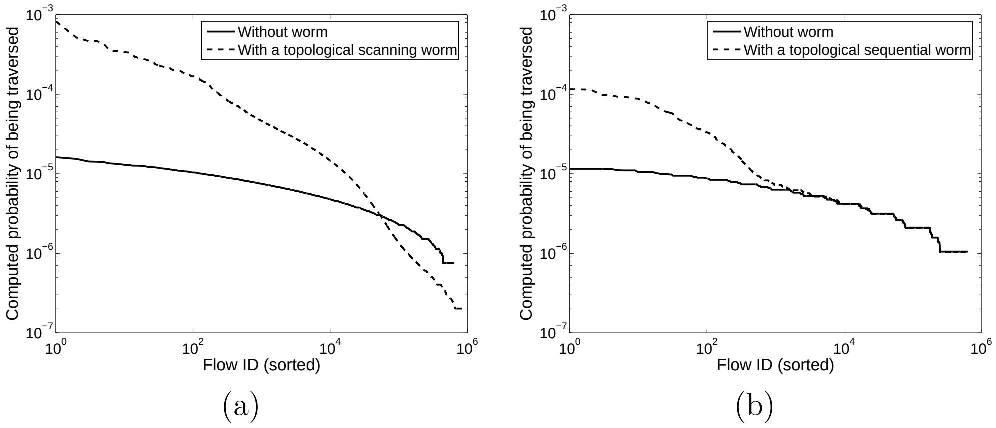


Figure 3.31: Probability distributions with and without topological spreading worms. (a) Each infected host scans its neighbor set randomly. (b) Each infected host scans its neighbors exactly once.

worms. Because topological worms may not be able to contact the whole host address space, we additionally increase the fraction of vulnerable hosts to $F = 0.5$, assuming attacks that spread topologically have a higher success rate in infection attempts. Thus these attacks can eventually infect approximately 0.1 fraction of hosts, like other worms. With a sequential worm, the scanning is more effective and infects all hosts that could be infected in 625 seconds, after which time the attack stops.

Figure 3.31 shows the probability distribution changes for the two different topological worms. Both topological worms resulted in significant distribution changes like random scanning worms, even though they contact only known neighbors in their contact sets. The detection accuracies of causal flows are approximately equal or even higher than the case of random scanning worms. Perhaps surprisingly, the causal flow detection accuracy for the topological sequential worm is much higher than the topological scanning worm, even though the former propagates via a slower rate and does

Worm type	Random hitlist	Coordinated hitlist
Fraction of hosts infected	0.1	0.1
Normalized attack rate	1	0.5
Maximum scans per host	$+\infty$	2
Attack duration (second)	500	500
Optimal Δt (second)	100	400
Causal flow accuracy	47%	6%

Figure 3.32: The configurations of the two hitlist spreading worms.

not perform as many scans. There are two reasons to explain the observation. First, the sequential worm hits vulnerable hosts more effectively with sequential scan, resulting in more effective propagation, and thus higher aggregate traffic rate earlier on. The second reason is the normalization factor in building the tree-structured worm traffic patterns. We examine the false positive flows returned by the topological scanning worm, and found that many of them were repeated infection attempts. Thus random moonwalks have higher chance of diverging in the presence both causal and repeated infection attempts, and converge slower to causal flows. These results, again, confirm the effectiveness of random moonwalks in capturing the high aggregated traffic rates and the tree-structured traffic patterns, regardless of the attack scanning models.

Hitlist Spreading Worms

The topological sequential worm suggests that with the knowledge of known vulnerable hosts, attacks can spread more effectively with a relative lower scan rate. We examine two different types of epidemic attacks that spread by hitlist: (1) a random hitlist worm that randomly scans hosts from the pre-compiled vulnerable host list, and (2) a coordinated hitlist worm that infects hosts using a divide and conquer strategy, with each infected host performing a maximal of two scans to only un-compromised hosts.

Figure 3.32 lists the properties of the two different hitlist worms we examine. And we plot the flow probability distributions resulted from the attack traffic in Figure 3.33. With a hitlist, attacks can spread slowly in order to evade detection, while still infect all vulnerable hosts in a short period. Both attacks in our experiment spread with equal or even lower rate than an average normal host, and last for 500 seconds only. For the random hitlist worm, we can still detect many causal flows by focusing on the highest ranked flows in the largest eigenvector, since the worm still incurs slight higher traffic rate with scan-like activity. The coordinated hitlist worm

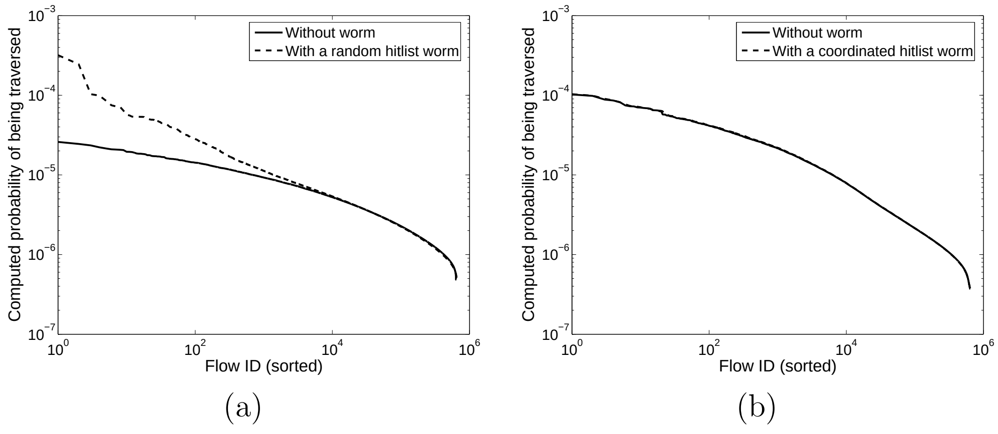


Figure 3.33: Probability distributions with and without hitlist spreading worms. (a) Each infected host randomly scans a pre-compiled list of vulnerable hosts. (b) Infected hosts scan in a coordinated way, with each one performing a maximal two scans to only un-compromised hosts.

pushes the capabilities of our algorithm to a limit, since each infected host contacts only two more hosts in addition to its normal traffic with a very slow rate. However, such type of worm, while conceptually simple, requires both accurate knowledge of vulnerable hosts and coordination strategies among all infected hosts, and is thus harder to implement in practice. For this type of worm, we observe no flow probability distribution change resulted from the attack traffic at all. Even in this case, our algorithm still successfully detects 6 initial causal flows out of the 100 top frequency ones, suggesting the great potential of the algorithm in identifying a wide class of stealthy attacks.

We note that in our experiments, the list of vulnerable hosts are selected randomly from the host population based on a given fraction. Once infected, each coordinated hitlist worm will also randomly select a new vulnerable host to propagate the attack. In reality, certain hosts might be more vulnerable to a specific attack than other hosts. For example, the SQL worm targets at systems running Microsoft SQL Server 2000 or Microsoft Desktop Engine (MSDE) 2000 only. In addition, attackers can strategically infect servers only among the vulnerable host list during its initial propagation. Because servers usually have a large number of incoming flows (i.e., a high fan-in rate), a causal attack flow to a server thus has a higher chance of evading detection due to the random selection of flows among all candidates. Random moonwalks may not be able to identify the initial causal flows for such attacks that leverage the non-uniform host in-degrees on the host contact graph. One direction of future work is therefore to study the practicality of such attacks, and quantify the detection performance degradation.

3.10 Distributed Network Forensics

Given the large number of ISPs and administrative domains (ADs) that make up the Internet, it is likely that Attacker Identification and Attack Reconstruction will have to be deployed in a distributed fashion across the network. Each individual ISPs or ADs may not have the complete host contact graph available. Even inside an AD, traffic auditing and forensics may not be deployed to cover the entire network. Some ADs will have very complete traffic auditing, others will have none, and many will audit packets only at their borders and peering points with other ADs.

We formulate different scenarios in which partial and distributed deployment can take place, identify challenges involved in each case, propose possible solutions, and evaluate the sensitivity of our algorithm to the extent of deployment and cooperation among domains.

There are in general three scenarios where we have only partial information to perform Attack Reconstruction and Attacker Identification, and may need cooperations from distributed domains:

1. Traffic auditing is deployed in a large backbone network such as AT&T and Sprint, but not the access networks connected to the backbone. In such case, we are likely to have the knowledge of a majority portion of the host contact graph with a small portion of data missing. Because our proposed random moonwalk method relies on statistical sampling of the traffic traces, it has the potential to yield robust performance against missing data. The challenging questions are: (1) How does the amount of data missing affect the algorithm effectiveness? (2) What types of data missing will degrade the performance most? The answers to these questions will help the ADs to strategically deploy traffic monitors inside their networks for best performance and costs. We discuss different situations where a small portion of data is missing, and quantify the impact of missing data in Section 3.10.1

2. Two or more peering backbone networks or ADs independently deploy network auditing. Due to trust and privacy issues (which will be further discussed in Section 3.12), data cannot not be shared across administrative domain boundaries. In such scenario, a challenging question is: Can each AD perform forensic analysis independently, and then collaborate with other ADs without exposing private data to each other? We propose and evaluate in Section 3.10.2 a distributed random moonwalk algorithm where no information about network internal traffic will be shared among different ADs.

3. Traffic auditing is deployed at small access networks, such as department or enterprise networks, to identify the attack entry point into the local network.

In the lack of a globally visible attack structure, the task of Attack Identification and Attack Reconstruction becomes more challenging. In such case, we might leverage additional information from network intrusion detection systems (NIDS) to obtain a good starting point for local traffic causality analysis. For example, since the scales of such networks will be relatively small, we may be able to afford more detailed traffic auditing and analysis, such as packet level traces, than the flow level graph that we explore. We may also exploit information such as the file system changes or system calls from host-based intrusion detection systems. Many existing methods on host based causality analysis (e.g., [48, 49]) can also be used to reconstruct partial attack graphs using a bottom-up approach. Thoroughly exploring these issues, however, is beyond the scope of this thesis.

3.10.1 Missing Data Impact on Performance

First, we consider the scenario where traffic monitoring is deployed in a backbone network with a small amount of data missing. Given that the majority of the host contact graph is available, most of the global tree structure of worm traffic will still be present, and potentially be identified by the random moonwalk algorithm.

There are two important questions regarding the algorithm performance in such case. First, how does the amount of data missing impact performance? Intuitively, the more missing data there are, the worse the performance would be. It would be important to understand to what extent the results are still useful for tracing the worm origin. Second, what portions of data missing will affect the algorithm effectiveness most? Certain part of data might play a more critical role than other data in the process of identifying the initial causal flows. Presumably, if most of the initial causal flows themselves are not audited or logged, it will be more challenging to carry further attack investigation.

To answer the above two questions, we perform a set of experiments with the same data set used in Section 3.7. To simulate the partial deployment scenarios, we group campus IP addresses based on /24 prefix subnets, and select a subset of those as the simulated access networks without traffic auditing. Based on the selected subnets, we split the whole network trace into two parts: backbone traffic and subnet traffic. Both parts consist of *internal traffic* and *border traffic*. The internal traffic refers to those flow records, whose senders *and* receivers both reside at the corresponding network(s). This portion of traffic will not be observed outside the network domain. The border traffic refers to those flow records, whose senders *or* receivers only, locate inside the corresponding network, but not both. Both the backbone and the access networks will be able to observe such border traffic, should each side deploy traffic auditing independently. Therefore, the portion of border traffic will be double counted as both the backbone traffic and the subnet traffic.

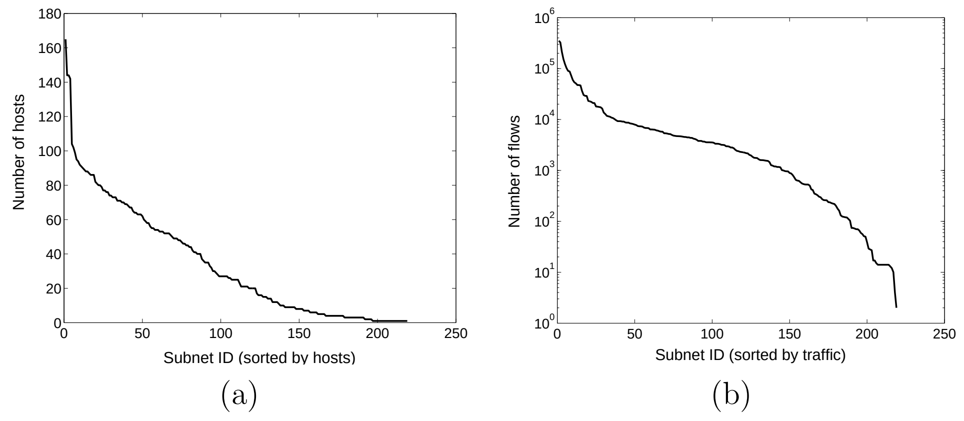


Figure 3.34: (a) The distribution of the number of subnet hosts. (b) The distribution of traffic generated.

For the four hour CMU campus trace with 8040 internal hosts, there are in total 219 /24 subnets. Figure 3.34 shows the the distributions of the number of hosts and the amount of traffic in terms of network flows generated from the 219 subnets. Given the non-uniform host and traffic distributions, we use the following methods to select subnets, assuming traffic monitoring is *not* deployed at the selected subnets to simulate different strategies of partial deployment:

- **By traffic:** Select subnets that generate the most amount of traffic.
- **By host:** Select subnets that have the most number of hosts.
- **By infection time:** Select subnets that contain those hosts who were infected earliest in time. Those hosts correspond to the ones on the initial levels of the causal tree.

Again, we choose the fraction of vulnerable host $F = 0.1$. The normalized worm rate that we introduce into the background trace is 1, with a worm inter-scan duration of 20 seconds. We use the detection accuracy of causal edges as our performance metric. For each method, we first perform a case study by removing only a small number of subnets from traffic auditing. We then gradually increase the number of subnets to remove in order to study the performance degradation verses the amount of traffic missing.

	Backbone	Subnet
Total traffic (%)	78	43
Total host (%)	97	3
Attack flows (%)	99	6
Causal flows (%)	100	6

Figure 3.35: Remove 4 subnets by traffic.

	Backbone	Subnet
Total traffic (%)	70	69
Total host (%)	92	8
Attack flows (%)	99	14
Causal flows (%)	99	14

Figure 3.36: Remove 10 subnets by traffic.

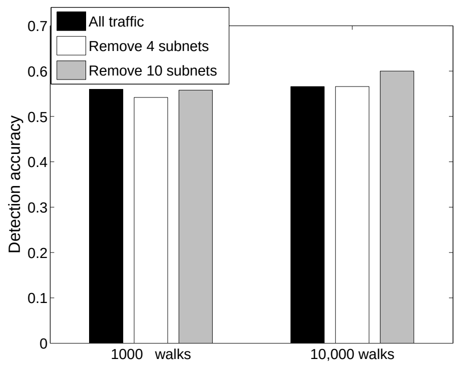


Figure 3.37: Detection accuracy with trace data missing from the subnets that generated most amount of traffic.

Remove Subnets by Traffic

In this case, we select a number of subnets that generated the most amount of traffic as the networks that are not monitored. Figure 3.35 and Figure 3.36 list the statistics about both the remaining backbone traffic and the subnet traffic, where we select 4 subnets and 10 subnets, respectively. Note since the border traffic between the backbone and the selected subnets will be counted in both traces, the sum of the backbone and the subnet traffic will be greater than 100% of the original trace. We also show the percentages of attack flows and causal flows, observed by both the backbone and the simulated access networks, counted in a similar way.

Figure 3.37 shows the detection accuracies of causal flows in such partial deployment scenarios, compared with the case where we have all the trace data available. Interestingly, by removing the small number of subnets that generated the most amount of traffic, we observe almost no performance degradation, though there is a significant amount of traffic reduction in the remaining backbone trace (up to 30% reduction). In the case where we split off the flow records from 10 subnets, the random moonwalk algorithm achieves even slightly higher detection accuracies than the performance with full data. To explain the reason, we examine the percentages of hosts from the missing subnets, as well as the percentages of attack traffic including

	Backbone	Subnet
Total traffic (%)	98	43
Total host (%)	84	16
Attack flows (%)	97	30
Causal flows (%)	98	30

Figure 3.38: Remove 10 subnets by host.

	Backbone	Subnet
Total traffic (%)	92	51
Total host (%)	73	27
Attack flows (%)	92	47
Causal flows (%)	92	47

Figure 3.39: Remove 20 subnets by host.

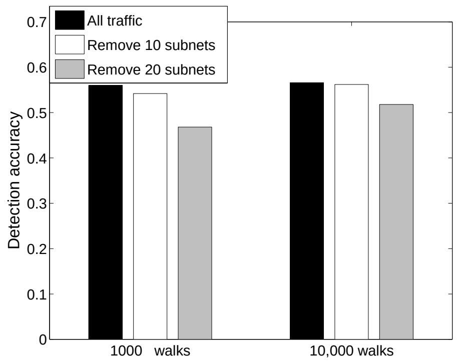


Figure 3.40: Detection accuracy with trace data missing from the subnets that have the most number of hosts.

causal flows left in the remaining trace in Figure 3.35 and Figure 3.36. We note that the selected subnets contain a small number of hosts only (less than 10%). Since our attack model is random scanning worms, most of the attack traffic generated from the small percentage of hosts is directed at the rest of host population located in the backbone network, and thus audited by the backbone as border network flows. As a contrast, the majority of the high volume of internal traffic generated inside these subnets is mostly normal traffic. Removing such normal traffic does not impact the performance significantly, and may even help concentrate random moonwalks toward attack flows and causal flows, resulting in slightly better performance.

Remove Subnets by Host

We then select subnets that have the most number of hosts as un-audited networks. Figure 3.38 and Figure 3.39 show the statistics about the traces where we split off 10 subnets and 20 subnets by host, respectively.

Figure 3.40 shows the performance degradation without the selected subnets. Removing subnets that have the most number of hosts does not reduce the amount of traffic seen at the rest of networks significantly, but has more negative impact on

	Backbone	Subnet
Total traffic (%)	99	15
Total host (%)	96	4
Attack flows (%)	99	8
Causal flows (%)	99	12

Figure 3.41: Remove 4 subnets by infection time.

	Backbone	Subnet
Total traffic (%)	99	20
Total host (%)	92	8
Attack flows (%)	99	16
Causal flows (%)	97	22

Figure 3.42: Remove 10 subnets by infection time.

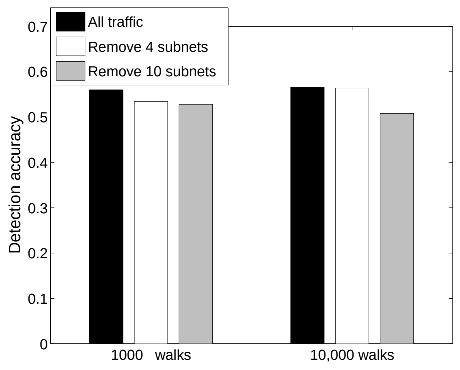


Figure 3.43: Detection accuracy with trace data missing from the subnets that contain hosts who were infected earliest in time.

the detection accuracies. This is again due to the scanning model of worm attacks to infect as many hosts as possible. With more hosts missing from partial backbone deployment, we are more likely to miss a larger fraction of attack traffic generated among the large number of missing hosts. Indeed, from Figure 3.38 and Figure 3.39, we do observe more fraction of attack flows and causal flows missing from the remaining trace, compared with the case where we split off subnets that generated most traffic. Such results suggest that for better detection performance, we should deploy traffic monitoring at networks with more host populations than networks that generate larger amount of network traffic.

Remove Subnets by Infection Time

In this case, we first sort all the infected hosts by their time of infection, and identify those subnets that contain the hosts that were infected earliest, and split them off from the backbone correspondingly. For example, if host H_a, H_b, and H_c are the first three infected hosts, and they belong to subnet S_a, S_b, and S_c, then we split these three subnets off from the backbone and focus on the remaining data.

Figure 3.41 and Figure 3.42 list the statistics about the backbone traces and the subnet traces, where we split 4 subnets and 10 subnets off by infection time, respectively. Figure 3.43 shows the detection accuracies in such partial deployment case, compared against the performance with full data available. We observe that while the backbone trace still keeps the majority of traffic and hosts, including attack flows and causal flows, the performance does degrade slightly. We achieve similar detection accuracies to the case where we remove almost 30% of hosts from the backbone, suggesting that removing subnets that contain the top level infected hosts generates the most impact on performance. Such results also have implications to how future stealthy attacks might evade detection in forensic analysis. If traffic auditing is only partially deployed, then attackers can carefully spread an attack at its initial stage, to within the boundary of a network where no traffic auditing is deployed. After a sufficient number of hosts are infected at this initial stage, the attackers can trigger all the infected hosts to start scanning toward the backbone, in order to launch a full-fledged attack.

Given the reduced detection accuracies with the top level infected hosts missing, an important question is: Will we still be able to find relatively lower levels of the causal tree, which will be used to provide directions and hints for further investigation? If the tree branches that we identify all point to hosts from a specific network, it is then likely to be the location where the attack originates from. Figure 3.44 shows the flow graph constructed from the 42 top-frequency flows after 10^4 walks on the trace where we remove 10 subnets selected based on the top level infected hosts. If a host involved in a flow belongs the subnets that we remove, we identify the host as from the missing subnets. We see there is a large tree branching structure emanating from the missing subnets, providing a strong evidence that the attack might have originated from the subnets without traffic auditing. Such results are encouraging for catching even stealthy attacks, which may topologically spread the infection during their initial phases.

Amount of Data Missing vs. Detection Accuracies

The previous subsections have discussed the cases where data is missing from only a small number of subnets. In this section, we vary the number of subnets that will be removed, and examine how performance degrades with the increasing amount of missing data, using the same three data selection methods.

Figure 3.45 (a) shows the detection accuracy of causal flows by varying the number of subnets that will be removed from traffic auditing. As the number of subnets does not directly reflect the amount of traffic missing, we also plot the detection accuracy vs. the amount of traffic missing from the campus trace in Figure 3.45 (b) correspondingly. Note all the infected hosts belong to the 100 subnets out of the total

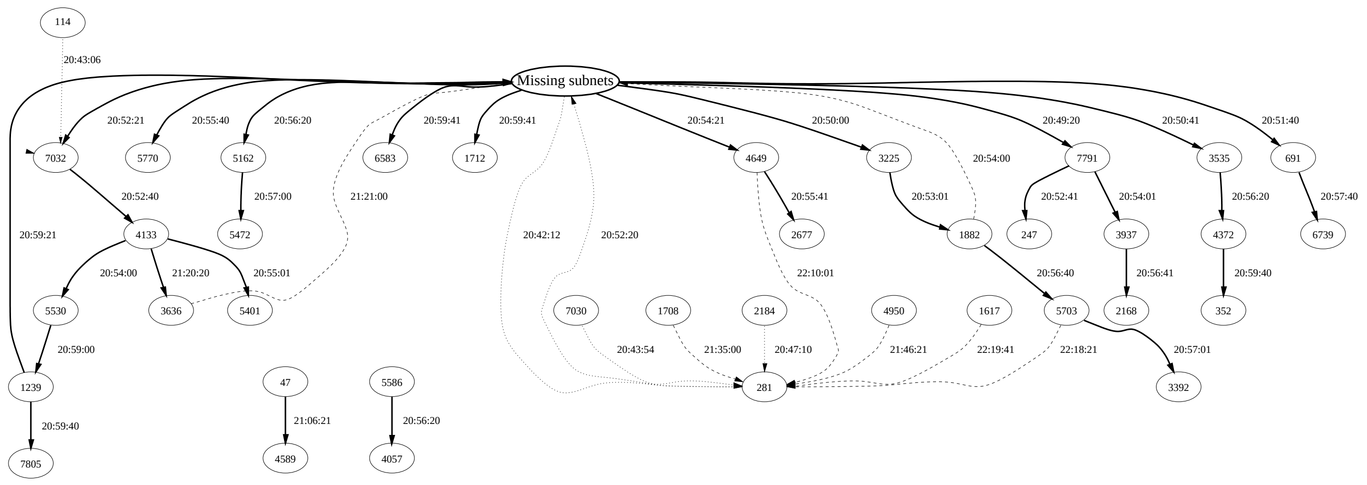


Figure 3.44: Host communication graph with 42 top frequency flows after 10^4 walks on the trace where we remove 10 subnets selected based on the top level infected hosts. Each circle represents a host with the number in the circle indicating the host ID. The "Missing subnets" node represent all the hosts that belong to the subnets without traffic auditing. Solid arrows indicate successful infection flows (i.e. causal edges), while the dashed arrows represent unsuccessful infection attempts. The dotted arrows correspond to normal traffic flows. The number beside each arrow indicates the flow finish time.

219 ones, so there is no data point in the figures after we remove the 100 subnets by infection time.

There are several observations we can draw from both figures. First, the performance degrades linearly with the increasing amount of missing data. Even with partial trace available, random moonwalks can still identify part of the causal tree for out of band investigation. If administrators can predict the amount of traffic monitored or collected, they can estimate the potential performance degradation based on the traffic auditing strategies. Second, removing subnets according to the traffic volume has very small impact to performance overall. From Figure 3.45 (b), we see that by removing subnets based on traffic, there is only 5% of performance decrease even in the case where 60% of traffic is missing. It suggests that traffic auditing can be performed in a strategic way to achieve similar performance with reduced storage cost. On the other hand, as discussed before, removing subnets by host or infection time reduces the detection accuracies more significantly. As a further explanation, Figure 3.46 shows the percentage of causal flows missing with the increasing amount of trace data removed using the different methods. Indeed, removing subnets based on top level infected hosts also eliminates many causal flows for forensic analysis, while the majority flows removed according to the traffic method are normal flows

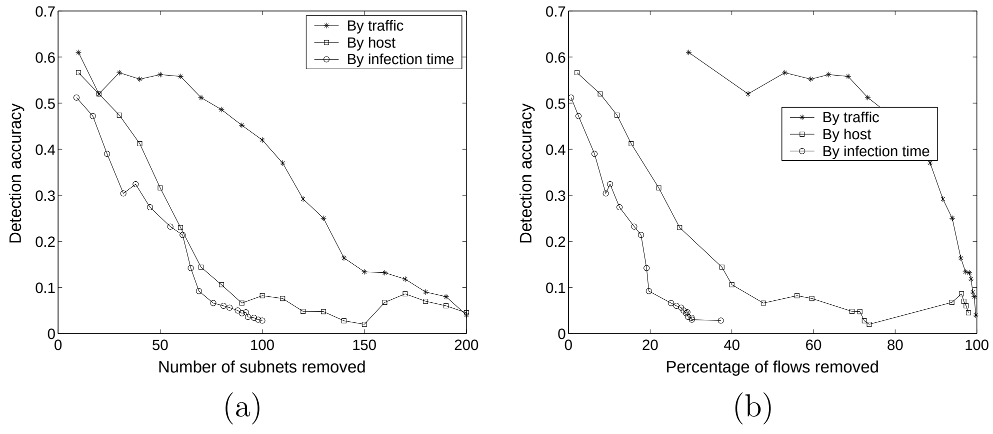


Figure 3.45: The causal flow detection accuracies with the decreasing amount of trace data available. (a) Performance vs. the number of subnets selected. (b) Performance vs. the amount of traffic removed.

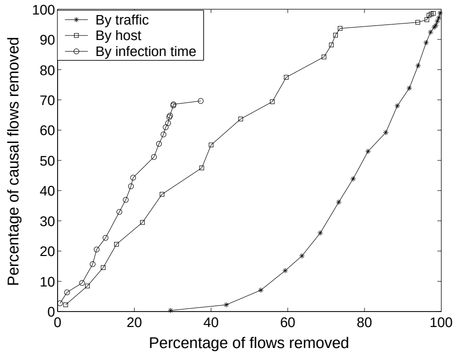


Figure 3.46: Fraction of causal flows missing vs. the total amount of traffic removed using the three different selection methods.

instead. Since our goal is to identify initial causal flows, the number of causal flows in the trace directly impacts the detection accuracies.

Performance Summary

To summarize, with a small amount of data missing, random moonwalks can still achieve similar detection accuracies, with only slight performance degradation. The performance decreases linearly with the amount of data missing, suggesting partial data would still be helpful for forensic analysis. Since our attack model is random scanning worms, it would be more effective to deploy traffic monitoring at networks with large host populations. Stealthy attacks may spread at un-audited networks during its initial stage to degrade the detection performance most. Even in such case, however, random moonwalks can still identify lower levels of the causal tree branches, which point to the networks from where the attack originates for further investigation.

3.10.2 A Distributed Random Moonwalk Algorithm

In this section, we consider the scenario where two or more networks independently perform traffic auditing. For privacy and scalability reasons, data will not be shared, or stored at centralized repositories. Forensic analysis has to be performed collaboratively by all the participating networks in a distributed way.

Our algorithm for worm origin identification can be elegantly adapted to such a distributed deployment scenario, to incrementally perform random moonwalks among multiple networks or administrative domains (ADs). Figure 3.47 illustrates the high level concept of the distributed random moonwalks, where four networks collaboratively perform the algorithm by exchanging intermediate results, as if the random moonwalks are conducted on the entire big network made up by all four networks.

Figure 3.48 illustrates the detailed steps of the algorithm, using an example of two networks, referred as AD_1 and AD_2, respectively. Solid arrows represent the directions of network flows, and dashed arrows represent the directions of random moonwalks (in reverse to the flow directions). α, β, γ, and ω denote four different types of flows. β and ω refer to those internal flows within the domains of AD_1 and AD_2, respectively. α refers to the set of flows originating from AD_1 and terminating at AD_2, while γ refers to the set of flows originating from AD_2 and terminating at AD_1. For each iteration i in the algorithm, both AD_1 and AD_2 keep a counter of how many times each flow f is traversed inside its own domain, denoted as $A_i(f)$ and $B_i(f)$, respectively. Given these notations, the algorithm works as follows.

Step 1: Each AD_k independently starts W_k random moonwalks, with initial steps randomly chosen among flows initiated by hosts inside its own network.

In our example, AD_1 starts W_1 random moonwalks with initial steps selected from the flow set α and β. Similarly, AD_2 starts W_2 random moonwalks from the flow set γ and ω.

Step 2: For each AD, many walks will reach border flows whose senders are from other domains. With only partial knowledge of the next step choices, such walks will be stopped, while others will finish as regular.

In the two network example, AD_1 will stop a walk once it reaches any flow in the set γ, and AD_2 will stop a walk once it reaches any flow in the set of α.

Step 3: After each iteration i, the ADs exchange the frequency counts of border flows. For every flow f that were initiated by a host from AD$_{j'}$ and received by a host from AD$_j$, AD$_j$ shares the frequency count $A_i(f)$ with AD$_{j'}$, so that all the walks that terminated at f prematurely could be continued within AD$_{j'}$ in the next iteration.

In this example, AD$_1$ shares with AD$_2$ the frequency counts of all the flows in the set of γ. For simplicity, we denote the set of counts as $A_i(\gamma)$. Similarly, AD$_2$ shares with AD$_1$ the set of frequency counts $B_i(\alpha)$.

Step 4: At each next iteration $i + 1$ where $i \geq 1$, every AD continues random moonwalks only from border flows that were initiated by hosts inside its own network, and destined to hosts outside. The number of walks starting from each flow equals exactly to the frequency count of the initial flow, which was exchanged during the previous iteration.

In this example, suppose the frequency of a flow $f \in \alpha$ being traversed is $B_1(f)$ at AD$_2$ after the first iteration. Then at the second iteration, AD$_1$ will perform $A_2(f) = B_1(f)$ random moonwalks, all with f as their initial steps. Because the source of f is located within AD$_1$, AD$_1$ now has a global view of all the possible next steps to continue those walks that stopped at f inside AD$_2$ during the first iteration.

Step 5: All the ADs iteratively perform the operations from Step 2 to Step 4, until there are no flows to continue the walks globally. For each AD, the final frequency count of a flow is the sum of of its frequency counts from all the iterations. The ADs can now share the final results for a global reconstruction, or individually perform further investigation using the highest frequency flows as suspicious causal flows.

In our example, there are in total three iterations. For AD$_1$, the final frequency count of a flow f will be computed as $\sum_{i=1}^{3} A_i(f)$. Similarly, for AD$_2$, the final frequency count of a flow f will be $\sum_{i=1}^{3} B_i(f)$.

During the above process, we note that every walk after the first iteration is a continuation of a path explored by other networks from the previous rounds. Thus the distributed algorithm is exactly simulating the global random moonwalks in a distributed fashion. Due to the backward direction of time, it is guaranteed that all the walks will terminate globally, if the ADs agree on a time boundary beforehand. The number of algorithm iterations is bounded by the maximum path length d of all the walks, which can be estimated based on the knowledge of the total trace length, and the sampling window size Δt.

To ensure that the initial steps of the distributed random moonwalks are chosen in a global randomized way, the number of walks, W_k, to be launch at AD$_k$, should be normalized based on the total number of flows that could potentially be selected from AD$_k$. Since the amount of traffic at a network might be used to infer sensitive information about the customer populations or the network topologies, it can not be

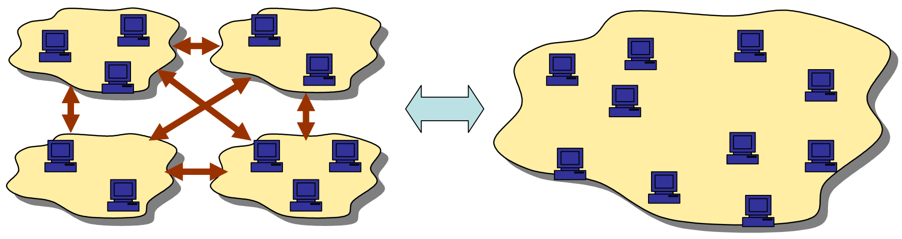


Figure 3.47: An AD graph illustrating the concept of incremental Attacker Identification and Attack Reconstruction.

shared or accessed by other ADs. Instead, all the ADs could agree on a normalization factor η for calculating the number of moonwalks launched from each network, where

$$\text{For each } AD_k,\, W_k = \eta |E|_k$$

.

Here $|E|_k$ represents the total number of flows initiated by hosts inside the network of AD_k. In our previous example, the number of moonwalks launched from AD_1 $W_1 = \eta(|\alpha| + |\beta|)$, and the number of walks performed by AD_2 $W_2 = \eta(|\gamma| + |\omega|)$. Since each AD independently performs random moonwalks within its own domain, W_k will not be released to the public.

The only information to be shared among different ADs, during the distributed algorithm, is the frequency counts of border flows as intermediate results. No information about internal hosts or traffic will be exposed. Due to the randomization process along every step of the walks, a malicious AD should not be able to reverse engineer W_k from the shared counts. In addition, because the records of border flows across two different ADs will likely to be audited and logged by both networks anyway, exchanging the frequency counts of these flows leaks no additional information about either ADs or end users.

Figure 3.49 shows the detection accuracies of the distributed random moonwalk algorithm applied to a two sub-network scenario simulated using the same CMU campus network trace. Each sub-network has approximately equal number of hosts, and Figure 3.50 lists the traffic statistics of the two sub-networks. The attack traffic bounces across both networks by randomly scanning destination hosts among the entire network, with a normalized attack rate of 1 (i.e., each infected host performs one scan per 20 second). Each network starts random moonwalks independently, with a total of 10^4 walks launched from both networks. We observe that the causal flow detection accuracies of both networks increase monotonically as the process goes on, and eventually converges. The final performance is comparable to what is achieved

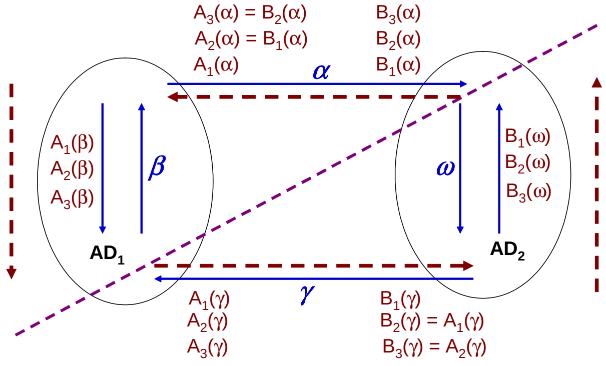


Figure 3.48: The distributed random moonwalk algorithm.

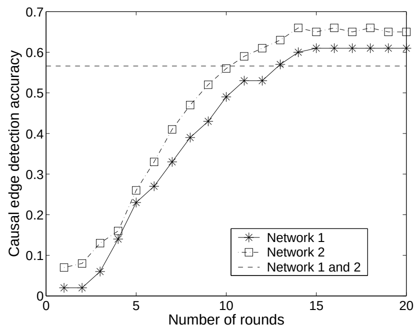


Figure 3.49: The causal edge detection accuracy of the top 100 frequency flows using the distributed random moonwalk algorithm.

	Sub-network 1	Sub-network 2
Total traffic (%)	83	57
Total host (%)	50	50
Attack flows (%)	75	74
Causal flows (%)	77	73

Figure 3.50: Traffic statistics of the two sub-networks for distributed forensic analysis

with a unified global view of the network.

To estimate the communication cost, we consider the worst case scenario where every walk terminates at a different border flow during each iteration. Assume there are in total W walks launched from all the networks, there will be maximally W frequency counts to be exchanged at each round. With a total of d iterations, the overall communication overhead is $W \times d$ counts, distributed among all the participating networks. In addition to the size of data in exchange, other costs include the overhead of setting up secure channels for communication, for example, the cost of authentication and data encryption.

3.11 Random Sampling vs. Biased Sampling

We have shown that, by *randomly* sampling paths of host communication and correlating the traversed flows, we can effectively identify the top level causal tree of a wide class of worm attacks. In this section, we explore variations of our algorithm, where we may strategically guide walks toward certain flows, or incorporate knowledge of compromised hosts output by the local intrusion detection systems (IDSes).

We discussed in Section 3.10.1 that a small number of hosts may generate a significant fraction of total traffic. Consequently, we are more likely to select those normal flows as our initial steps by the "randomness" of the selection, taking more hops to reach the causal tree. Without the knowledge of compromised hosts, a simple heuristics to speedy the convergence of moonwalks is to start the walks from as many hosts as possible, since a potential target of a worm is to infect a large number of hosts eventually. In our implementation, we set a counter for each host on the number of times it being selected as a starting point, and set a maximal threshold to bound the counter. We define this heuristic as the **maximum coverage rule**.

Alternatively, suppose we have the perfect knowledge of the list of the compromised hosts after an attack, then the moonwalks can be guided toward these infected hosts to achieve better performance in potential. We consider the following set of algorithm variations in this case:

- **Initial step rule:** We can start moonwalks from flows initiated by the known compromised hosts only.
- **Next step rule:** When selecting a next step in the path, prefer flows initiated by a compromised host than by a normal host.
- **Rank adjust rule:** After performing the moonwalks, adjust the flow ranks by lowering the ranks of those flows not initiated by a compromised host.

We evaluate the impact of these rules on performance using the same campus network trace, introducing different rates of artificially injected worm attacks. Again, we use the computed optimal Δt to perform 10^4 moonwalks for each case, and repeat each run 5 times.

Figure 3.51 compares the causal flow detection accuracies resulted from the different heuristics. Interestingly, the additional preferences in selecting the initial steps have no substantial impact on performance. The maximum coverage rule outperforms the default random moonwalk algorithm with marginal accuracy increase, suggesting that the default random moonwalks can already cover a large number of hosts quickly due to the random selection of next step hops. Because of the attack's random scanning model, most of the walks are also likely to be directed to infected hosts at some point, resulting in similar effect as starting the walks from compromised hosts only. However, biasing walks toward compromised hosts along the entire paths does indeed improve the detection accuracy significantly. The reason is that by favoring flows initiated from infected hosts, moonwalks will more easily stay on the causal tree and converge to the initial causal flows. Adjusting the flow ranks after moonwalks

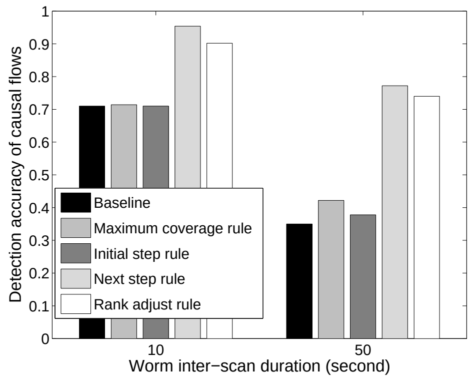


Figure 3.51: Detection accuracy achieved with different heuristics. “Baseline” refers to the default random moonwalk algorithm without heuristics.

can also lead to big performance improvement by removing those high ranked normal flows that are received by hosts on the top levels of the causal tree.

In summary, the knowledge of compromised hosts has a great potential to improve the detection performance. Such information can be obtained from the output of local IDSes to strategically guide the selection of next steps for further enhancing the capabilities of random moonwalks.

3.12 Discussion and Future Work

The previous sections have explored algorithmic issues of identifying the origin of epidemic spreading attacks and the initial causal flows, both in centralized and distributed scenarios. In practice, deploying an Internet-scale forensic analysis system for Attacker Identification and Attack Reconstruction is a non-trivial task. There will be a number of challenges as a result of the sheer scale of the Internet traffic and the need for cooperation between different service providers. Next, we discuss some of the challenges that exists in the system deployment and their implications, and outline future research directions.

a) A tremendous amount of data would be required to represent the complete host contact graph of the Internet. Our proposed method for identifying attack propagation paths relies on a network auditing system to log end host communication records. To bound the amount of audit data a network might observe, let us assume that a major ISP has O(100) POPs, and each POP has 10 Gbps capacity towards the middle of the ISP's network. The total amount data flowing through the network could then be 10^2 POPs/ISP $\times$ 10^{10} bits/sec/POP $= 10^{12}$ bits/sec. Now, supposing an average packet size of 1000 bits and an average flow of 10 packets, that would be 10^{12} bits/sec /(10^1 packets/flow $\times$ 10^3 bits/packet) = 10^8 flows/sec.

So, if we wished to store a record for each flow sent over the network, it would require 10^8 flow records/sec $\times 10^2$ bits/flow record $= 10^{10}$ bits/sec , i.e., 10 Gbps for the ISP or 1% of the overall network capacity (100×10 Gbps). In order to keep one hour of data, the storage obligation for the ISP would be roughly 450 GB to store the compressed data (assuming a 10 fold compression rate), distributed among the POPs.

Thus the sheer volume of the data to be collected, stored and analyzed represents a problem of scalable and efficient data collection and analysis, but the quantities involved are not inconceivable.[2] We also note that by the time a worm infection becomes so pervasive, that the induced traffic potentially outpaces these logging capabilities, the records most important for finding the attack origin, namely those close to the origin, have already been recorded.

b) The complete host contact graph for the entire Internet will not be available. As we discussed in Section 3.10, it is likely that auditing and forensic analysis will be deployed in a piecemeal fashion across the Internet. Our eventual goal is to realize a federated Dragnet system, where each network or AD deploys the Dragnet monitors and agents to cooperatively perform the attack investigation.

On the algorithmic side, our proposed random moonwalk method relies on statistical sampling of the traffic traces, making it intrinsically robust to many kinds of missing data. As shown in Section 3.10.1, even when critical data is systematically missing, the method still produces a partial causal tree that can be used as a basis for further out-of-band investigation. The study about the missing data impact on performance and the distributed random moonwalk algorithm presented in Section 3.10 are only our initial efforts toward this direction. There are also system issues to be solved for a real deployment.

Similar to single-packet IP traceback [94], we also envision an architecture in which distributed collection points log flow records and corresponding timestamps. In addition to the source and destination IP addresses, each flow record contains an identifier for distinguishing two flows between the same source and destination at roughly the same time, for which we can use, e.g., the 13-bit identifier field of the initial packet in the flow in the case of IPv4. Though this is not strictly necessary, it permits us to relax the degree of clock synchronization necessary among collection points and can improve the accuracy of our search. At each individual collection point, we require two causally related flows be logged in their causal order and timestamped with a common clock.

[2]Today, 500 GB of disk space costs \$500 to \$1,000.

With multiple networks or ADs collaborating with each other interactively, we also need architectural support to share intermediate results in a reliable way. Such data exchange could be realized either by a "push" mode, or by a "pull" mode where ADs set up distributed repositories for flow record querying. In either case, only authorized ADs should be able to access records in an authenticated way. And we need mechanisms to detect misbehaving ADs in order to enforce the correctness of the analysis results.

c) Privacy concerns must be addressed. Issues of trust and cooperation between domains raise challenges with respect to protecting both domain proprietary information and end user privacy. It is particularly important to prevent the execution of queries that can either retrieve an arbitrary part of the entire host contact graph recorded by an AD (hence leaking business data about the AD) or read out all flows to or from an arbitrary host (hence violating the privacy of a normal host).

Even though our distributed algorithm requires information sharing/querying on only border flows, misbehaving ADs can collaborate to infer the traffic patterns of a particular victim AD. In addition, the frequency counts of border flows might create a covert channel for inferring network traffic or routing properties, especially in the existence of multiple information channels among different ADs. Future work include exploring the privacy implications of such data sharing, as attackers can carefully craft traffic patterns to create covert channels that may disclose sensitive information.

The creation of a distributed network auditing service will also serve as a practical application of work by ourselves and other researchers to develop techniques that limit the disclosure of private information without compromising the amount of useful information retrieved, and without adding too much communication overhead.

d) The security of the auditing system itself must be maintained. If data is to be useful to law enforcement authorities, the auditing system must be constructed to maintain a "chain of custody" that can convince a jury that the data cannot have been tampered with after being collected. It must also be impossible for an outside attacker to "frame" an uninvolved host by creating traffic that implicates the host. Further, the auditing system itself will likely become a favorite point of attack. It must be defended against attackers who might first DoS attack the auditing system, then launch attacks without running the risk of detection.

e) There are a wide range of other types of attacks. Our approach is effective for the class of attacks that propagate via "tree" structured communication patterns. Future work includes the development of algorithms to perform post-mortem analysis of a larger class of attacks. Our current implementation assumes that the semantic direction of the flow is consistent with the network notion of flow directionality. Attacks may try to obfuscate the notion of causality among network flows. As future work, we can explore ways to make the algorithm robust to such attacks.

Chapter 4

Related Work

This chapter discusses related work in two parts. In the first part, we survey various approaches for leveraging distribution information and correlation mechanisms. While these efforts are in general related with both applications we study, there is also application specific related work, which we will present in the second part.

4.1 Distributed Security Architectures

Spatiotemporal event correlation exploits sequences of events distributed across both time and space. Leveraging information gathered from distributed multiple measurement points, however, is not a new approach itself. Many researchers have noticed the potential of the collective approaches in security, especially for intrusion detection and anomaly detection in different contexts.

A lot of efforts have focused on the architectural issues in system design, so that events and alerts from distributed monitors could be audited and effectively synthesized. EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) [79], DIDS (Distributed Intrusion Detection System) [92], CSM (Cooperative Security Managers) [110], and AAFID (Autonomous Agents For Intrusion Detection) [6] are independently proposed distributed architectures for intrusion detection and response. They all use local monitors or agents to collect and filter suspicious events and anomaly reports from a variety of sources (network packet traces, SNMP traffic, and system logs, etc.), watching multiple network links and tracking user activity across different machines. The architectures provide both communication methods for monitors to exchange locally detected information and coordination methods to orchestrate various components of the systems. In particular, both EMERALD and AAFID aggregate alerts hierarchically, via different service layers or data collection components. In DIDS, local monitors perform data collection and reduction, then the generated alerts are analyzed by a centralized rule-based expert system. As a contrast, CSM uses a peer-to-peer way to communicate among individual hosts without a centralized director. Our works starts with a similar motivation, but more focuses on a specific approach of correlating the distributed events, while these systems target to provide the infrastructures useful for configuring distributed intrusion detectors and combining alarms from them. We envision the spatiotemporal event correlation approach to be complementary to these effort, not as a replacement of the existing architectures that provide global observation sharing.

4.2 Alert Correlation

Correlating different types of audit logs and alerts is another active area to exploit distributed information for security. Abad et al. [3] have proposed to correlate heterogeneous types of logs (e.g., system calls and network traffic) using both signature-based method and data mining method. Compared with using a single type of logs, their system can increase the detection accuracies.

In the same area, many researchers have proposed various statistical learning techniques to correlate INFOSEC (information security) alerts. Valdes and Skinner [103] propose a probabilistic-based correlation method using similarity measures of alert attributes. Follow up work [4] presents a case study of their deployment experience on correlating heterogeneous sensor data with live traffic to reduce alert volumes. Qin and Lee [80] have developed a statistical framework to causally analyze sequences of alerts for constructing attack scenarios, where low level alerts are aggregated and prioritized for time-series analysis.

There are also many other approaches that correlate sequences of alerts for analyzing attack scenarios. In [23, 69, 68], the correlations are performed by defining rules and relating alerts through pre-conditions and post-conditions. The assumption is that the consequences of earlier attacks have a strong connection with the prerequisites of later attacks, since the earlier attacks are usually launched in preparation for the later ones. Noel et al. [70] propose a signal analysis method to correlate sequences of alerts by reasoning distances on attack graphs constructed with additional vulnerability analysis to reduce the online processing overhead. Another approach proposed by Debar and Wespi [27] defines an architecture to correlate alerts through both backward-looking method (to reason about duplicate alerts) and forward-looking method (to reason about consequence alerts), and then to aggregate them with predefined rules into attack scenarios.

Other efforts of correlating alerts include M-correlator [78], where a stream of security alerts are ranked and clustered into a consolidate incident stream for further analysis, and M2D2 [66], where the authors propose a formal data model for integrating different types of information into intrusion detection.

Compared with these alert correlation techniques, our approach tries to correlate system state transitions defined as low level events, rather than the results of local intrusion analysis. As such, spatiotemporal event correlation can potentially identify more anomalous events that look normal from local monitors, and also detect the lack of normal patterns. Secondly, most of these approaches focus on temporally correlating alert sequences (though [3] discusses the advantages of spatial correlation in their paper), while our approach uses correlation techniques to learn both the temporal and spatial pattern of host state transitions globally. Finally, this thesis work currently focuses on correlating information gathered by homogeneous monitors (specifically, the file system change monitors and network flow monitors), rather than heterogeneous types of logs that are exploit by many these existing methods. As future work, we may enhance our work to include different type of measurement data to represent individual host states.

4.3 Related Work to Seurat

In the area of anomaly detection, Seurat uses file system updates to represent a host state change. File system updates have been known to be useful information for intrusion detection. Tripwire [101], AIDE [57], Samhain [85] are well-known intrusion detection systems that use file system updates to find intrusions. Recently proposed systems such as the storage-based intrusion detection systems [77] and some commercial tools [76] support real-time integrity checking of file systems. However, all of them rely on a predefined rule set to detect anomalous integrity violation, while Seurat automatically diagnoses the anomaly using learning based correlation across time and space.

Several learning-based approaches to intrusion detection have been proposed to avoid the tedious task of defining detection rules and to spot new types of attacks. For example, Warrender et al. [108] used various machine-learning techniques to model normal system-call sequences. Ghosh et al. [35] applied Artificial Neural Networks to learn anomaly and misuse detection models for system programs. Ko [51] employed a machine learning method (e.g., Inductive Logic Programming) to model valid operations of a program, and Lee et al. [56] proposed a framework for data mining various system audit data. Significant deviations from the automatically learned model indicates anomaly in those systems, similar to the techniques we used in Seurat. These existing approaches, however,analyze only the temporal behavior of hosts or systems, while overlooking the advantages of spatially located events.

More related correlation approaches to Seurat are [105, 106] proposed by Wang et al. and [25] proposed by Dagon et al., whose authors have also noticed the value of spatial correlation. Wang et al. [105, 106] have applied a collective approach to tackle misconfiguration problems. In their system, a malfunctioning machine can diagnose its problem by collecting system configuration information from other similar and friendly hosts connected via a peer-to-peer network. The work does not target automatic detection of anomalous events or patterns through correlation, but rather it aims at figuring out the cause of a detected problem. Dagon et al. [25] have proposed a system to correlate observed events from honeypot machines for local worm detection. As honeypot machines are not used by actual users, any active network event is suspicious and will be correlated with the same event from other honeypot machines to confirm the existence of worm propagation. Their approach therefore shares similar motivations with alert-based correlation, while in our approach, we correlate both normal and abnormal events from regularly used machines to automatically detect the existence of abnormal patterns.

The specific feature reduction techniques employed by Seurat, namely the wavelet analysis and the principal component analysis methods, have also been used in network security for anomaly detection and diagnosis. Except for being used for disease outbreak detection in [119], wavelet analysis is also used in the domain of traffic analysis to identify anomalous events that signal the onset of an attack [7]. Recently, Lakhina et al [54] have proposed a PCA based method to diagnose traffic volume anomalies, where the high-dimensional space occupied by traffic measurements is separated into disjoint subspaces correlating to normal and abnormal network conditions.

4.4 Related Work to Dragnet

In the area of network forensic analysis, we are not aware of any previous work that can automatically pinpoint the origin of an epidemic attack or the initial causal infection events.

Our random moonwalk algorithm assumes that attack flows do not use spoofed source IP addresses, since in the types of attacks we consider here, attack packets are rarely, if ever, spoofed. The overwhelming majority of attack traffic involved in the propagation is initiated by victims instead of the original attacker, so using spoofed addresses would only decrease the number of successful attacks[1] without providing extra anonymity to the attacker.

If attackers do begin to use spoofed addresses, then traceback techniques [87, 16, 9, 94, 58] could be used to determine the true source of each flow sampled by our

[1]For example, spoofed packets are useless for propagating an infection over TCP-based communications, since the TCP handshake cannot complete, and spoofing addresses for UDP-based attacks in the presence of egress filters [32] results in the attack flows being discarded.

algorithm. Traceback alone, however, is not sufficient to track worms to their origin, as traceback determines only the true source of the packets received by a destination. In an epidemic attack, the source of these packets is almost never the origin of the attack, but just one of the many infected victims. Some method is still needed to find the hosts higher up in the causal tree.

Among these IP traceback techniques, the single-packet traceback system of Snoeren et al. [94] shares our requirement for an auditing system that logs the source and destination of network traffic. While Snoeren's scheme requires logging *per packet* at each router on the packet's path, our approach only requires logging *per flow* with far fewer logging elements, as flows need only be recorded by enough elements to permit a query to find them and determine their causal orderings [55].

The notion of computer forensic analysis is not new, and has been previously studied in other context. King et al have proposed a set of mechanisms to reconstruct a time-series attack events on a local host by observing OS-level objects [48, 49]. Puchholz and Shields have also discussed possible approaches to identify origins of an attack by analyzing process level communication events [17, 18]. Both of their work, aims at local host causality analysis, and is not appropriate for large scaled network forensics.

Recent work [118, 29] has also noticed the value of spatially correlating communication events for detecting propagating attacks. Compared with our work, these approaches try to exploit attack specific behaviors. Because they either require detailed packet-level analysis or host state monitoring, their effectiveness is limited and cannot be applied in our application.

GrIDS (Graph-based Intrusion Detection System) [21] is another closely correlated system that detects attacks by building a graph representation of network activity based on the reports from all the hosts in a network. It correlates TCP/IP activity events between hosts in the network to infer patterns of intrusive or hostile activities based on predefined rules. GrIDS was designed to potentially identify attack propagating paths like Dragnet, but with significant differences compared with our approach. First, to reduce data complexity, GrIDS uses a hierarchy to incrementally filter suspicious events with rules. It thus bears more similarity to alert-based correlation approaches. In addition, GrIDS targets mostly on-line detection of attacks, while our focus in Dragnet is to perform off-line analysis on a tremendous number of network flows automatically.

Other work on traffic causality analysis has mostly focused on detecting stepping stones, which is suggested [93] as a potential solution for worm origin identification together with IP traceback. Just as we discussed that IP traceback cannot be used to trace the origin of epidemic attacks, stepping stone techniques are not suitable for our objectives either.

There have been in general two categories of approaches for detecting stepping stones. The first class of approaches focuses on content-based techniques [96] and thus require very expensive packet payload analysis. More importantly, they cannot track down flows from polymorphic worms or worms that encrypt payloads. The other class of approaches [120, 28] focus on correlating packet-level characteristics (e.g., inter-packet timings) to detect if multiple interactive connections are part of a single attack session. However, using fine-grained packet timing characteristics for establishing causality does not work for worm attacks which typically do not use interactive sessions. Even in the context of detecting causality of interactive flows, such techniques still remain an active area of research especially with respect to the robustness of such timing correlations [12, 107]. In contrast, our work ignores packet-level characteristics and attack signatures, but instead focuses on establishing causal relationships between flows by exploiting the globally visible structure of attacks. Thus the random moonwalk algorithm can potentially be agnostic to specific attack contents, attack packet sizes, or port numbers used.

While our work does not depend on the generation of worm signatures, our approach is complementary to these efforts [52, 47] as well as other efforts in detecting the existence of attacks [65, 42, 44, 113] and traffic anomalies [8].

Finally, our method for correlating random walks is inspired by link analysis [53], where the authors infer correlations among social network entities from their activity patterns.

Chapter 5

Conclusions and Future Work

In this chapter, we summarize our key contributions, discuss limitations, and propose several new directions for future work.

5.1 Contributions

Network security is an arms race where new attacks will be invented trying to outwit existing detection and analysis techniques. A wide variety of malicious attacks today attempt to disrupt the normal functioning of a large number of hosts in the Internet. While many of these attacks are aggressive in their propagation, and their malicious patterns can be identified by known signatures individually, future such attacks can potentially be much more stealthy to evade the detection of traditional detection systems deployed in isolation. Nevertheless, to effectively compromise more hosts eventually, these attacks are bound to display abnormal patterns that can be discerned by correlating locally observed events in a global way. On the other hand, the increasing capacity and performance of networks and storage devices enable the efficient collection and storage of a huge amount of audit data distributed across the networks, making it possible to analyze large quantities of data globally. Our work is motivated by both the trend that attacks are becoming more sophisticated, requiring global coordinated detection and analysis capabilities, and the technology trend of various high performance devices.

The main contribution of this thesis is a general solution to *reliably and effectively capture the abnormal patterns of a wide class of attacks, whose activities, when observed in isolation, may not seem suspicious or distinguishable from normal host activity changes.* To achieve this goal, we present a spatiotemporal event correlation approach that correlates events across both space and time, identifying aggregated abnormal patterns to the host state updates. The key observation behind our approach is that events introduced by malware in a network system often have both temporal and spatial locality. Abnormal events or patterns, which may not seems suspicious locally, will stand out when they are correlated with events taking place at different times and locations.

Based on the same high level concept, we instantiate the spatiotemporal event correlation approach in two important security applications. The first application we look at is anomaly detection, where we develop a pointillist method to detect abnormal file updates shared across multiple hosts (Chapter 2). The use of spatiotemporal event correlation, in our prototype system Seurat, effectively detects both simulated attacks and various manually launched worms and viruses in real computer clusters. The second application we study is network forensic analysis, where we propose a Dragnet framework for collecting and analyzing flow level traffic data (Chapter 3). In this application, the spatiotemporal event correlation approach is exemplified as a random moonwalk algorithm that identifies the initial successful infection flows for tracing epidemic attack origins. Our analysis, simulation based experiments, and real trace study have shown that the algorithm is effective in identifying the causal relationships between initial infection events for both fast propagating worms and a wide class of stealthy attacks. These results together, demonstrate the unique advantage of spatiotemporal event correlation in both reducing false alarms and identifying global abnormal patterns.

Compared with traditional correlation based approaches, a major challenge of spatiotemporal event correlation is the increased data scale and complexity by exploiting aggregated low level events. Yet, such challenge can be effectively addressed through compact event representations and efficient correlation algorithms, as demonstrated by the two applications studied in the thesis. In both cases, we reduce the data volumes by orders of magnitude through statistical sampling and feature reduction mechanisms, demonstrating the feasibility of our approach in practice.

While the high level idea is the same, the two applications utilize different types of event data. The first application focuses on host file system updates, and the second application examines network flows among end hosts. The specific data representation methods and the correlation algorithms we develop in the two examples are also quite different. In anomaly detection, the file update events are represented as feature vectors, and the correlation is performed by clustering vectors in a high dimensional space. For network forensic analysis, we represent communication events with a host contact graph, and the correlation is performed by sampling paths of network flows. All these suggest that spatiotemporal event correlation is a general approach, with no specific constraints on the types of data to be correlated or the particular algorithms to be used. We believe such an approach can be applicable to a wide range of security applications for identifying groups of related events with various semantics.

5.2 Limitations

Because the power of spatiotemporal event correlation comes from an aggregated view of events across both time and space, our approach will be most successful in detecting attacks that result in abnormal changes distributed at multiple hosts and locations. Examples of such attacks include email viruses, worms, DoS attacks, and DDoS attacks. Our approach has similar limitations as many traditional approaches in detecting attacks that succeed only once or target only a few hosts in a the network. For example, attacks to a misconfigured FTP server that requires no root password may not be able to duplicate at other machines. In the lack of global abnormal patterns, the spatiotemporal event correlation may not be as effective to detect these attacks.

In these single attack scenarios, however, spatiotemporal event correlation still has its value by temporally correlating events for anomaly detection. However, human inputs may be required in a feedback loop to eliminate the false alarms caused by unseen normal activities. When applied strategically, spatiotemporal event correlation may also detect single host attacks as outliers in the space dimension, providing starting points for further investigation.

The detection of aggregated malicious behavior suggests that attackers may significantly slow down the rate of propagation, so that their infection events gradually blend into the normal events to evade detection and identification. In particular, when there are only a single or a few malicious events during our detection or analysis window, we may not be able to catch them through spatiotemporal event correlation. One solution is to use a larger detection window, waiting for more attack instances, which also suggests that the detection latency might be longer. However, we note that by slowing down the rate of infection, the degree of attack virulence will also decrease, compromising fewer machines and leaving more time for patching vulnerable hosts. It is interesting future work to understand to what extent we can still leverage the power of aggregates for each specific application.

5.3 Future Work

In this section, we present research directions that are left open by this thesis.

5.3.1 Distributed Data Collection and Correlations

In the current prototypes of both Seurat and Dragnet, we rely on a centralized data repository and analysis engine to store and correlate events collected from distributed monitors. Despite its simplicity, such centralized scheme has both scalability and privacy issues. A distributed architecture to store and query data will potentially solve both problems. Such architecture will consist of the following components: (1) distributed monitors for collecting audit data in a coordinated way, (2) distributed repositories for indexing and storing audit trace, and (3) a distributed query mechanism to access event records efficiently. There are already a few initial attempts [2, 59, 98] from other research groups in this direction, focusing on specific types of data and scenarios.

The correlation algorithm can also be designed to analyze events in a distributed collaborative fashion. In Section 3.10.2, we discussed how we can adapt the random moonwalk algorithm so that multiple administrative domains can collaboratively perform worm origin identification without releasing private data. There are also efforts [60, 67] in sensor networks to aggregate information from distributed monitors. As future work, we can design and implement a generalized framework to support a wide class of correlation algorithms at distributed locations.

5.3.2 Privacy Preserving Data Access Mechanisms

Our current correlation algorithms assume it is possible to directly access all the available data. In a larger scaled system with distributed repositories, event records may not be shared straightforwardly to ensure user privacy. A number of existing solutions such as [105, 95, 116] have proposed encryption-based and anonymity-based mechanisms to share data with privacy preservation, but they only support limited functionalities. It is a challenging problem for future research to effectively support a wide range of correlation operations on event records without violating privacy guarantees.

5.3.3 Incorporation of Heterogenous Types of Data

In our thesis work, we consider mostly homogenous types of events for correlation. Existing work has shown that leveraging different types of audit logs can improve the accuracy of attack detection [3, 66, 70]. In Section 2.5, we also explored a general feature vector space that supports different types of file update attributes, and have shown that the use of additional feature attributes can help detect more types of abnormal events. In the Dragnet system, the combination of intrusion detection results with the host contact graphs can also significantly improve the detection accuracy of causal flows (Section 3.11). These preliminary results are encouraging, and suggest that, for future research, we can more effectively exploit the value of various types of events in the correlation process.

5.3.4 Other Applications

Spatiotemporal event correlation is a general approach that can benefit a wide class of security applications. This thesis work has shown its effectiveness with two examples. As future work, we can further generalize this approach to other scenarios, for example, network-based intrusion detection. In Seurat, we use file system update events to represent host state changes and thus require per-host based monitoring. With the increasing deployment of network traffic monitors, we can easily log both the incoming and outgoing traffic of each host without interfering normal host functions. Such traffic data can be encoded to represent host state changes, and correlated across different machines for detecting malicious scans or backdoor traffic. As another example, spatiotemporal event correlation can potentially be used to detect massive spam emails or the unauthorized use of email forwarding servers, by correlating both the incoming and outgoing emails from a large number of users.

Bibliography

[1] . PlanetLab. `http://www.planet-lab.org`.

[2] S. Seshan A. R. Bharambe, M. Agrawal. Mercury: Supporting Scalable Multi-Attribute Range Queries. In *Proc. of the ACM Sigcomm*, 2004.

[3] C. Abad, J. Taylor, C. Sengul, Y. Zhou, W. Yurcik, and K. Rowe. Log Correlation for Intrusion Detection: A Proof of Concept. In *Proc. of the 19th Annual Computer Security Applications Conference*, Las Vegas, Nevada, USA, December 2003.

[4] D. Andersson, M. Fong, and A. Valdes. Heterogeneous Sensor Correlation: A Case Study of Live Traffic Analysis. Presented at IEEE Information Assurance Workshop, June 2002.

[5] K. Avrachenkov, N. Litvak, D. Nemirovsky, , and N. Osipova. Monte Carlo Methods in PageRank Computation: When One Iteration is Sufficient. Technical report, University of Twente, 2005.

[6] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. An Architecture for Intrusion Detection Using Autonomous Agents. In *Proc. of the 14th IEEE Computer Security Applications Conference*, December 1998.

[7] P. Barford, J. Kline, D. Plonka, and A. Ron. A Signal Analysis of Network Traffic Anomalies. In *Proc. of ACM SIGCOMM Internet Measurement Workshop*, 2002.

[8] P. Barford, J. Kline, D. Plonka, and A. Ron. A Signal Analysis of Network Traffic Anomalies. In *ACM SIGCOMM IMW*, November 2002.

[9] S. Bellovin, M. Leech, and T. Taylor. ICMP Traceback Messages. Internet draft, work in progress, 2001.

[10] M. W. Berry, Z. Drmac, and E. R. Jessup. Matrices, Vector Spaces, and Information Retrieval. *SIAM Review*, 41(2), April 1999.

[11] CERT Advisory CA-2003-20: W32/Blaster worm. `http://www.cert.org/advisories/CA-2003-20.html`, 2003.

[12] A. Blum, D. Song, and S. Venkataraman. Detection of Interactive Stepping Stones: Algorithms and Confidence Bounds. In *Proc. of The Seventh International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2004.

[13] L.A. Breyer. Markovian Page Ranking Distributions: Some Theory and Simulations. Technical report, `http://www.lbreyer.com/preprints.html`, 2002.

[14] A. Z. Broder, R. Lempel, F. Maghoul, and J. Pedersen. Efficient Pagerank Approximation Via Graph Aggregation. In *Proc. WWW04*, 2004.

[15] J. Browne. Probabilistic Design. `http://www.ses.swin.edu.au/homes/browne/probabilisticdesign`.

[16] H. Burch and B. Cheswick. Tracing Anonymous Packets to Their Approximate Source. In *Proc. of USENIX LISA Systems Administration Conference*, 2000.

[17] B. Carrier and C. Shields. Providing Process Origin Information to Aid in Network Traceback. In *Proc. of the 2002 USENIX Annual Technical Conference*, 2002.

[18] B. Carrier and C. Shields. The Session Token Protocol for Forensics and Traceback. *ACM Transactions on Information and System Security*, 7:333–362, 2004.

[19] CERT Coodination Center. Overview of Attack Trends. `http://www.cert.org/archive/pdf/attack_trends.pdf`, 2002.

[20] CERT Coordination Center. Overview of Attack Trends. `http://www.cert.org/archive/pdf/attack_trends.pdf`, 2002.

[21] S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, J. Rowe, S. Staniford-Chen, R. Yip, and D. Zerkle. The Design of GrIDS: A Graph-Based Intrusion Detection System. Technical Report CSE-99-2, U.C. Davis Computer Science Department, January 1999.

[22] F. R. K. Chung. Spectral Graph Theory. 1997.

[23] F. Cuppens and A. Miege. Alert Correlation in a Cooperative Intrusion Detection Framework. In *IEEE Symposium on Security and Privacy*, May 2002.

[24] G. E. Blelloch D. Blandford and I. Kash. Compact Representations of Separable Graphs. In *ACM/SIAM Symposium on Discrete Algorithms (SODA)*, 2003.

[25] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levin, and H. Owen. HoneyStat: Local Worm Detection Using Honeypots. In *The 7th International Symposium on Recent Advances in Intrusion Detection (RAID 2004)*, September 2004.

[26] H. Debar, M. Dacier, and A. Wespi. Towards a Taxonomy of Intrusion-detection Systems. *Computer Networks*, 31(9):805–822, April 1999.

[27] H. Debar and A. Wespi. Aggregation and Correlation of Intrusion-Detection Alerts. In *Recent Advances in Intrusion Detection*. Volume 2212 of Lecture Notes in Computer Science, Springer-Verlag, 2001.

[28] D. L. Donoho, A. G. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford-Chen. Multiscale Stepping-Stone Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay. In *Proc. of The 5th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2002.

[29] D. R. Ellis, J. G. Aiken, K. S. Attwood, and S. D. Tenaglia. A Behavioral Approach to Worm Deetection. In *Proc. of the 2004 ACM workshop on Rapid malcode (WORM)*, 2004.

[30] F-Secure. F-Secure Security Information Center. `http://www.f-secure.com/virus-info`.

[31] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-Law Relationships of the Internet Topology. In *ACM SIGCOMM*, 1999.

[32] P. Ferguson and D. Senie. RFC 2267 - Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing, 1998.

[33] E. Forgy. Cluster analysis of multivariante data: Efficiency vs. Interpretability of classifications. *Biometrics*, 21(768), 1965.

[34] A. Gersho and R. Gray. Vector Quantization and Signal Compresssion. Kluwer Academic Publishers, 1992.

[35] A. K. Ghosh and A. Schwartzbard. A Study in Using Neural Networks for Anomaly and Misuse Detection. In *8th Usenix Security Symposium*, 1999.

[36] C. Gkantsidis, M. Mihail, and E. Zegura. Spectral Analysis of Internet Topologies. In *Proc. of Infocom*, 2003.

[37] D. Gleich, L. Zhukov, and P. Berkhin. Fast Parallel PageRank: A Linear System Approach. Technical report, Yahoo, 2004.

[38] G.H. Golub and C.F. Van Loan. *Mattrix Computation.* Johns Hopkins University Press, 2nd edition, 1989.

[39] T. Haveliwala. Efficient Computation of PageRank. Technical report, Stanford University, 1999.

[40] R. Heady, G. Luger, A. Maccabe, and M. Servilla. The Architecture of a Network Level Intrusion Detection System. Technical report, Computer Science Department, University of New Mexico, 1999.

[41] L. T. Herberlein, G. V. Dias, Karl N. Levitt, Biswanath Mukherjee, J. Wood, and D. Wolber. A Network Security Monitor. In *Proc. of the 1990 IEEE Symposium on Security and Privacy*, 1990.

[42] A. Hussain, J. Heidemann, and C. Papadopoulos. A Framework for Classifying Denial of Service Attacks. In *Proc. of ACM SIGCOMM*, 2003.

[43] I. T. Jolliffe. Principle Component Analysis. *Spring-Verlag, New York*, 1986.

[44] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing . In *Proc. of IEEE Symposium on Security and Privacy*, 2004.

[45] M. Kamber. Data mining: Concepts and techniques. *Morgan Kaufmann Publishers*, 2000.

[46] G. H. Kim and E. H. Spafford. The Design and Implementation of Tripwire: A File System Integrity Checker. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, 1994.

[47] H. A. Kim and B. Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. In *Proc. of 12th USENIX Security Symposium*, 2004.

[48] S. T. King and P. M. Chen. Backtracking Intrusions. In *2003 Symposium on Operating Systems Principles (SOSP)*, 2003.

[49] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen. Enriching Intrusion Alerts Through Multi-host Causality. In *2005 Network and Distributed System Security Symposium (NDSS)*, 2005.

[50] J. Kleinberg. Authoritative Sources in a Hyperlinked Environment. In *Proc. of 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[51] C. Ko. Detecting Intrusions Using System Calls: Alternative Data Models. In *IEEE Symposium on Security and Privacy*, May 2000.

[52] C. Kreibich and J. Crowcroft. Honeycomb – Creating Intrusion Detection Signatures Using Honeypots. In *Proc. of ACM HotNets-II*, 2003.

[53] J. Kubica, A. Moore, D. Cohn, and J. Schneider. Finding Underlying Connections: A Fast Graph-Based Method for Link Analysis and Collaboration Queries. In *Proc. of Twentieth International Conference on Machine Learning*, 2003.

[54] A. Lakhina, M. Crovella, and C. Diot. Diagnosing Network-Wide Traffic Anomalies. In *ACM SIGCOMM*, August 2004.

[55] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21:558–565, 1978.

[56] W. Lee and S. J. Stolfo. A Framework for Constructing and Models for Intrusion Detection Systems. *ACM Transactions on Information and System Security*, 3(4), November 2000.

[57] R. Lehti and P. Virolainen. AIDE - Advanced Intrusion Detection Environment. `http://www.cs.tut.fi/~rammer/aide.html`.

[58] J. Li, M. Sung, J. Xu, L. Li, and Q. Zhao. Large-scale IP Traceback in High-speed Internet: Practical Techniques and Theoretical Foundation. In *Proc. of IEEE Symposium of Security and Privacy*, 2004.

[59] X. Li, F. Bian, H. Zhang, C. Diot, R. Govindan, and G. Iannaccone. MIND: A Distributed Multi-Dimensional Indexing System for Network Monitoring. Under submission, 2005.

[60] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *OSDI*, 2002.

[61] M. Maila and J. Shi. A Random Walks View of Spectral Segmentation. In *AI and STATISTICS 2001(AISTATS)*, 2001.

[62] A. Moore. K-means and Hierarchical Clustering. `http://www.cs.cmu.edu/~awm/tutorials/kmeans09.pdf` (available upon request), November 2001.

[63] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer Worm. *IEEE Security and Privacy*, 1:33–39, July 2003.

[64] D. Moore, C. Shannon, G. M. Voelker, and S. Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. In *Proc. of IEEE INFOCOM*, 2003.

[65] D. Moore, G. M. Voelker, and S. Savage. Inferring Internet Denial-of-Service activity. In *Proc. of 10th USENIX Security Symposium*, 2001.

[66] B. Morin, L. Me, H. Debar, and M. Ducasse. M2D2: A Formal Data Model for IDS Alert Correlation. In *Recent Advances in Intrusion Detection*, September 2002.

[67] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson. Synopsis Diffusion for Robust Aggregation in Sensor Networks . In *ACM SenSys*, 2004.

[68] P. Ning, Y. Cui, and D. S. Reeves. Analyzing Intensive Intrusion Alerts Via Correlation. In *Recent Advances in Intrusion Detection*. Volume 2516 of Lecture Notes in Computer Science, Springer-Verlag, 2002.

[69] P. Ning, Y. Cui, and D. S. Reeves. Constructing Attack Scenarios through Correlation of Intrusion Alerts. In *Proc. of the 9th ACM Conference on Computer and Communications Security*, 2002.

[70] S. Noel, E. Robertson, and S. Jajodia. Correlating Intrusion Events and Building Attack Scenarios through Attack Graph Distances. In *Proc. of the 20th Annual Computer Security Applications Conference*, December 2004.

[71] The Network Time Protocol. `http://www.ntp.org`.

[72] PacketStorm. Packet Storm. `http://www.packetstormsecurity.org`.

[73] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bring order to the web. Technical report, Stanford University, 1998.

[74] K. Park and H. Lee. On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets. In *ACM SIGCOMM*, 2001.

[75] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Proc. of 7th USENIX Security Symposium*, 1998.

[76] Pedestal Software. INTACT$^{\text{TM}}$. `http://www.pedestalsoftware.com/products/intact`.

[77] A. Pennington, J. Strunk, J. Griffin, C. Soules, G. Goodson, and G. Ganger. Storage-based Intrusion Detection: Watching Storage Activity for Suspicious Behavior. In *Proc. of 12th USENIX Security Symposium*, Washington, DC, August 2003.

[78] P. A. Porras, M. W. Fong, and A. Valdes. A Mission-Impact-Based Approach to INFOSEC Alarm Correlation. In *Recent Advances in Intrusion Detection*, September 2002.

[79] P. A. Porras and P. G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proc. of the 20th National Information Systems Security Conference*, October 1997.

[80] X. Qin and W. Lee. Statistical Causality Analysis of INFOSEC Alert Data. In *Recent Advances in Intrusion Detection*, September 2002.

[81] M. Richardson and P. Domingos. Mining Knowledge-Sharing Sites for Viral Marketing. In *Proc. of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

[82] S. Robertson, E. Siegel, M. Miller, and S. Stolfo. Surveillance Detection in High Bandwidth Environments. In *Proc. of the 2003 DARPA DISCEX III Conference*, 2003.

[83] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proc. of USENIX LISA Systems Administration Conference*, 1999.

[84] University of Oregon Route Views Project. `http://www.routeviews.org`.

[85] Samhain Labs. Samhain. `http://la-samhna.de/samhain`.

[86] SANS Institute. Lion Worm. `http://www.sans.org/y2k/lion.htm`, April 2001.

[87] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *Proc. of ACM SIGCOMM*, 2000.

[88] S. E. Schechter, J. Jung, and A. W. Berger. Fast Detection of Scanning Worm Infections. In *Proc. of 7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2004.

[89] B. Schneier and J. Kelsey. Cryptographic Support for Secure Logs on Untrusted Machines. In *The Seventh USENIX Security Symposium*, January 1998.

[90] Bruce Schneier. Attack Trends: 2004 and 2005. `http://www.schneier.com/blog/archives/2005/06/attack_trends_2.html`.

[91] V. Sekar, Y. Xie, D. Maltz, M. K. Reiter, and H. Zhang. Toward a Framework For Internet Forensic Analysis. In *Proc. of ACM HotNets-III*, 2004.

[92] S. R. Snapp, S. E. Smaha, D. M. Teal, and T. Grance. The DIDS (distributed intrusion detection system) prototype. In *The Summer USENIX Conference*, pages 227–233, San Antonio, Texas, June 1992. USENIX Association.

[93] A. Snoeren. Public Review of 'Toward a Framework for Internet Forensic Analysis'. In *Proc. ACM HotNets-III*, 2004.

[94] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-Based IP Traceback . In *Proc. of ACM SIGCOMM*, 2001.

[95] D. X. Song, D. Wagner, and A. Perrig. Practical solutions for search on encrypted data. In *IEEE Symposium on Security and Privacy*, May 2000.

[96] S. Staniford-Chen and L. T. Heberlein. Holding Intruders Accountable on the Internet. In *Proc. of the IEEE Symposium on Security and Privacy*, 1995.

[97] S. Staniford-Chen, V. Paxson, and N. Weaver. How to Own the Internet in Your Spare Time. In *Proc. of 11th USENIX Security Symposium*, 2002.

[98] K. Suh, Y. Guo, J. Kurose, and D. Towsley. Locating Network Monitors: Complexity, Heuristics, and Coverage. In *Proc. of Infocom*, 2005.

[99] Symantec. Symantec Security Response. `http://securityresponse.symantec.com`.

[100] Symantec. Symantec Internet Security Threat Report, Volume VI. `http://enterprisesecurity.symantec.com/content.cfm?articleid=1539`, September 2004.

[101] Tripwire, Inc. Tripwire. `http://www.tripwire.com`.

[102] Trusted Computing Platform Alliance. Trusted Computing Platform Alliance. `http://www.trustedcomputing.org`.

[103] A. Valdes and K. Skinner. Probabilistic Alert Correlation. In *Recent Advances in Intrusion Detection*. Volume 2212 of Lecture Notes in Computer Science, Springer-Verlag, 2001.

[104] D. Wagner and D. Dean. Mimicry Attacks on Host-Based Intrusion Detection Systems. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, November 2002.

[105] H. Wang, Y. Hu, C. Yuan, and Z. Zhang. Friends Troubleshooting Network: Towards Privacy-Preserving, Automatic Troubleshooting. In *Proc. of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2004.

[106] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y. Wang. Automatic Misconfiguration Troubleshooting with PeerPressure. In *Proc. of Usenix OSDI*, 2004.

[107] X. Wang and D. Reeves. Robust Correlation of Encrypted Attack Traffic Through Stepping Stones by Manipulation of Inter-packet Delays. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, 2003.

[108] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting Intrusions Using System Calls: Alternative Data Models. In *IEEE Symposium on Security and Privacy*, May 1999.

[109] L. Wasserman. *All of Statistics: A Concise Course in Statistical Inference.* Springer-Verlag, 1st edition, 2004.

[110] G. White, E. Fisch, and U. Pooch. Cooperating security managers: A peer-based intrusion detection system. *IEEE Network*, 10(1), 1994.

[111] Whitehats, Inc. Whitehats Network Security Resource. `http://www.whitehats.com`.

[112] R. J. Wonnacott and T. H. Wonnacott. Introductory Statistics. *Fourth Edition.*

[113] J. Wu, S. Vangala, L. Gao, and K. Kwiat. An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2004.

[114] Y. Xie, H.A. Kim, D. O'Hallaron, M. K. Reiter, and H. Zhang. Seurat: A Pointillist Approach to Anomaly Detection. In *Seventh International Symposium on Recent Advances in Intrusion Detection (RAID 04), month = sep, year = 2004,*.

[115] Y. Xie, D. R. O'Hallaron, and M. K. Reiter. A Secure Distributed Search System. In *Proc. of the 11th IEEE International Symposium on High Performance Distributed Computing*, July 2002.

[116] Y. Xie, D. R. O'Hallaron, and M.K. Reiter. Protecting Privacy in Key-Value Search Systems. Technical Report CMU-CS-03-158, CMU Computer Science Department, 2003.

[117] Y. Xie, V. Sekar, D. Maltz, M. K. Reiter, and H. Zhang. Worm Origin Identification Using Random Moonwalks. In *Proc. of the IEEE Symposium on Security and Privacy (Oakland 2005)*, 2005.

[118] J. Xiong. ACT: Attachment Chain Tracing Scheme for Email Virus Detection and Control. In *Proc. of the 2004 ACM workshop on Rapid malcode (WORM)*, 2004.

[119] J. Zhang, F. Tsui, M. M. Wagner, and W. R. Hogan. Detection of Outbreaks from Time Series Data Using Wavelet Transform. In *AMIA Fall Symp.*, pages 748–752. Omni Press CD, October 2003.

[120] Y. Zhang and V. Paxson. Detecting Stepping Stones. In *Proc. of 9th USENIX Security Symposium*, 2001.

[121] L. Zhou, L. Zhang, F. McSherry, N. Immorlica, M. Costa, and S. Chien. A First Look at Peer-to-Peer Worms: Threats and Defenses. In *Proc. of Fourth International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.

Chapter 6

Appendices

6.1 Probability Estimation in Section 3.6.2

An edge $e = \langle u, v, k \rangle$ can occur at different steps of a random moonwalk. We use $P_i(e^k)$ to denote the probability of an edge at time k being traversed by the i-th step of a walk. Then we have $P(e^k) = \sum_{i=1}^{d} P_i(e^k)$.

We use $O(v, k)$ to denote the number of concurrent outgoing flows from host v at time k. With $|E|$ edges in the host contact graph, we have

$$P_i(e^k) = \begin{cases} 1/|E| & i = 1 \\ \left(\sum_{j=1}^{O(v,k+1)} P_{i-1}(e^{k+1,j})\right)/I(v,k) & i > 1 \end{cases}$$

where $e^{k+1,j}$ is the jth flow generated by host v at time $k+1$, and $I(v,k)$ is the number of incoming flows into host v at time k. The above equation holds for any host contact graph, without any assumptions.

Under the uniform scanning assumption for both normal and attack traffic, a second order approximation for $E(\frac{1}{I(v,k)})$ is ,

$$\begin{aligned} E\left(\frac{1}{I(v,k)}\right) &= \frac{1}{E(I(v,k))}\left(1+\left[\frac{\sigma_{I(v,k)}}{E(I(v,k))}\right]^2\right), \text{ from [15].} \\ &\approx \frac{1}{E(I(v,k))} = \frac{1}{I(k)} \end{aligned}$$

The above approximation holds for large enough $|H|$ and A, since $I(v,k)$ is binomially distributed.

Under the simplified assumptions discussed in Section 3.6.1, if $e_m^k = \langle u, v, k\rangle$ is a malicious-destination edge, we have $O(v, k+1) = A$, otherwise, $O(v, k+1) = B$. Using the approximate form for $1/I(v,k)$ above, for an edge at time k we have:

$$\begin{aligned} P_2(e_m^k) &\approx \frac{1}{I(k)}\sum_{j=1}^{A} P_1(e^{k+1,j}) = \frac{A}{|E|I(k)} \\ P_2(e_n^k) &\approx \frac{1}{I(k)}\sum_{j=1}^{B} P_1(e^{k+1,j}) = \frac{B}{|E|I(k)} \\ P_3(e_m^k) &\approx \frac{1}{I(k)}\sum_{j=1}^{A} P_2(e^{k+1,j}) = \frac{(B+R)T_{k+1}}{|E|I(k)I(k+1)} \\ P_3(e_n^k) &\approx \frac{1}{I(k)}\sum_{j=1}^{B} P_2(e^{k+1,j}) = \frac{B \times T_{k+1}}{|E|I(k)I(k+1)} \end{aligned}$$

By induction, we can easily show that $\forall d'$ $(4 \leq d' \leq d)$,

$$\begin{aligned} P_{d'}(e_m^k) &\approx \frac{(B+R)T_{k+d'-2}}{|E|I(k)I(k+1)} \\ P_{d'}(e_n^k) &\approx \frac{BT_{k+d'-2}}{|E|I(k)I(k+1)} \end{aligned}$$

Taking the sum of all $P_i(e)$ $(1 \leq i \leq d)$, we have

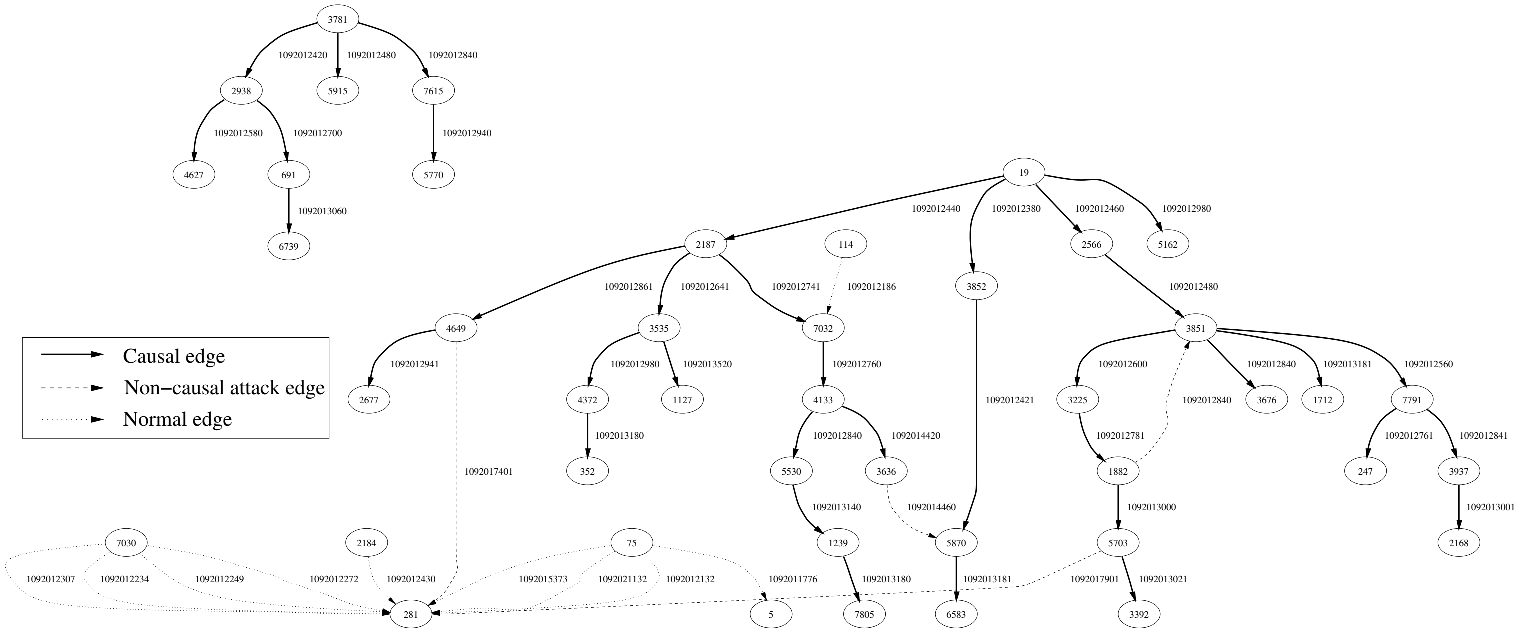


Figure 6.1: Host communication graph with 50 top frequency flows after 10^4 walks on Trace-20 (Figure 3.12). Each circle represents a host with the number in the circle indicating the host ID. Solid arrows indicate successful infection flows (i.e. causal edges), while the dashed arrows represent unsuccessful infection attempts. The dotted arrows correspond to normal traffic flows. The number beside each arrow indicates the flow finish time.

$$P(e_m^k) \approx \frac{1}{|E|}\left[1+\frac{A}{I(k)}+\frac{(B+R)\times\sum_{i=1}^{d-2}T_{k+i}}{I(k)I(k+1)}\right]$$

$$P(e_n^k) \approx \frac{1}{|E|}\left[1+\frac{B}{I(k)}+\frac{B\times\sum_{i=1}^{d-2}T_{k+i}}{I(k)I(k+1)}\right]$$

6.2 Host Communication Graph of Top Frequency Flows

Using real background traffic trace, we show a host communication graph in Figure 6.1 with 50 top frequency flows from Trace-20 (See Figure 3.12) after 10^4 random walks. Each walk has no upper bound limitations for maximum path length d with a $\Delta t =$

800 seconds.

Among the 50 flows, there are in total 36 successful infection flows (i.e., causal edges), and the host 3781 was the actual worm origin. We observe that there are tree branching structures below both host 3781 (the source) and host 19 which is the first host infected by 3781. It turned out that host 19 is a Syslog server and has a large number of incoming flows. Thus when random walks trace back along the tree to host 19, they diffuse among these normal incoming flows. Although random walks did not converge from host 19 to host 3781, both of the hosts provide starting points for further investigation given the automatically identified tree structures. For example, we can search through the traces to extract all flows with timestamps between these two hosts. Alternatively, based on the selected top frequency flows, we can extract traffic characteristics for successful infection flow augmented with more information such as port numbers and flow size. Similar to Figure 3.16, we also observe quite a few incoming flows to host 281, who is a real infected host (by some variant of Blaster worm) perform aggressive scanning.