Enriching intrusion alerts through multi-host causality

Samuel T. King, Z. Morley Mao, Dominic G. Lucchetti, Peter M. Chen

Department of Electrical Engineering and Computer Science
Computer Science and Engineering Division
University of Michigan
http://www.eecs.umich.edu/CoVirt

Abstract

Current intrusion detection systems point out suspicious states or events but do not show how the suspicious state or events relate to other states or events in the system. We show how to enrich an IDS alert with information about how those alerts causally lead to or result from other events in the system. By enriching IDS alerts with this type of causal information, we can leverage existing IDS alerts to learn more about the suspected attack. Backward causal graphs can be used to find which host allowed a multi-hop attack (such as a worm) to enter a local network; forward causal graphs can be used to find the other hosts that were affected by the multi-hop attack. We demonstrate this use of causality on a local network by tracking the Slapper worm, a manual attack that spreads via several attack vectors, and an e-mail virus. Causality can also be used to correlate distinct network and host IDS alerts. We demonstrate this use of causality by correlating Snort and host IDS alerts to reduce false positives on a testbed system connected to the Internet.

1. Introduction

Intrusion detection systems (IDSs) are used to alert system administrators to possible attacks. IDSs monitor events at various levels in a system and attempt to alert administrators to events that match known attacks (signature-based IDS) or do not match normal behavior (anomaly-based IDS). Host-based IDSs monitor events on a single host, such as system calls or file accesses. Network-based IDSs monitor packets that are sent or received.

One factor that limits the accuracy and usefulness of IDS alerts is the lack of accompanying context. IDS alerts point out suspicious events without relating them to other events and state in the system. For example, Snort [1] highlights suspicious messages but does not describe how the receiving system reacted to these messages or what caused the sending system to send the suspicious message. Tripwire [6] alerts administrators to modified system files but does not show what caused the modifications or how the modified system files were used afterward. Additional context may help an administrator see the connections between different alerts or find other hosts that have been compromised.

This paper describes how to enrich an IDS alert with causal information about the state or events detected by the IDS. Informally, causal information shows which events led to the state/event detected by the IDS and which events were affected by the state/event detected by the IDS. Formally, this causal information is defined by Lamport's *happens-before* relationship [8], which results from messages between hosts and interprocess communication within a host. For example, a causal dependency is created from process A to process B if process A forks process B, if process A sends a message to process B, or if process A writes a file that is later read by process B. By iteratively analyzing these causal dependencies, we compute the transitive closure of causal relationships in the form of a multi-host causality graph. A causality graph with an IDS alert as a root node shows which events were affected by the state/event that generated the alert. A causality graph with an IDS alert as a leaf node shows which events led to the state/event that generated the alert.

Causality graphs can improve the effectiveness of IDS alerts, as in the following scenarios:

- A worm exploits a vulnerability in a company's public web server, then starts infecting machines that are behind the company firewall. An IDS on one of the internal machines detects anomalous activity, but no IDS alerts occur on the company's public web server. A causality graph can trace a worm from the machine that detected the intrusion back to the company's public web server that allowed the worm into

the network. Figures 1–3 demonstrate this capability for a testbed of machines infected by the Slapper worm.

- An intruder logs in as a privileged user and schedules a job (e.g., through the root's crontab file) for that night that will scan slowly for other vulnerable hosts. While logging in as a privileged user and issuing a slow port scan are both suspicious, neither is suspicious enough on its own to trigger an alert to the administrator. A causality graph can show that the login session that wrote the crontab file indirectly led to the slow port scan. Figure 6 demonstrates how a causality graph linked two IDS alerts during an attack on one of our testbed computers.

The contributions of this paper are (1) to propose causality graphs as a way to enrich IDS alerts; (2) to show how to enhance the BackTracker system [7] to create bi-directional, multi-host causality graphs; (3) to demonstrate how causality graphs can help track the propagation of multi-host attacks within an administrative domain; and (4) to demonstrate how causality graphs can help correlate otherwise disconnected IDS alerts.

The following is an overview of the paper. Section 2 describes how the original BackTracker system finds and displays objects and events from a single host that causally precede a given intrusion detection point, and Section 3 describes how we expand the scope of BackTracker to find and display events from a *network of hosts* that causally precede *or causally follow* a given intrusion detection point. We call this extended system BDB, which stands for Bi-directional, Distributed BackTracker. BDB represents a mechanism that administrators can use to enrich IDS alerts; administrators can then use various policies on top of this mechanism. Sections 4 and 5 describe several scenarios that demonstrate how an administrator might use the information provided by BDB to track multi-hop attacks (worms, manual attacks, and e-mail viruses) and correlate disconnected IDS alerts. Section 7 describes some limitations of our approach, and Section 8 relates our work to prior research.

2. BackTracker

We introduced the idea of using causality to analyze intrusions with the BackTracker system [7]. BackTracker uses a modified Linux kernel to log system calls that form dependencies between operating system objects. BackTracker analyzes dependencies between three types of objects: processes, files, and filenames. For example, a process⇒file dependency is formed when a process writes a file, and a file⇒process dependency is formed when a process reads a file. Files are identified by inode number to allow BackTracker to track a file across rename operations and through symbolic links. A process⇒process dependency is formed when one process creates another process. Bi-directional dependencies are also possible, such as when two processes share memory with each other (process⇔process). See [7] for more details on the objects and events analyzed by BackTracker.

BackTracker's analysis starts from a suspicious object or event detected by an IDS. BackTracker is independent of which IDS is used, as long as the IDS can identify a suspicious operating system event or object. BackTracker analyzes the causal relationships formed between objects to construct a graph of the objects and events that led to the specified intrusion detection point. We call this a *backward graph* because it looks back in time from a given IDS alert. BackTracker then filters out the dominant sources of noise in the backward graph according to default filtering rules. For example, the login program reads and writes the file /var/run/utmp, which makes it appear that each new login session is affected directly by all prior login sessions. BackTracker's default rules seek to filter out events that generate a lot of noise in the backward graph but enable only a low degree of control between objects. New rules can be added easily by an administrator, and we use this to add 1-3 simple rules for each network service used in this paper (web, FTP, SMB). For example, we filter out reads and writes to the list of processes maintained by the FTP server (/var/run/ftp.pids-all). See [7] for a fuller discussion of the use of filtering rules.

[7] showed that combining causal analysis and a set of six simple rules was very effective. We analyzed multiple attacks on a machine we had set up as a honeypot, and in each case the resulting backward graph highlighted effectively the source of the intrusion and the path between the initial compromise and the intrusion detection point.

Figure 2 shows BackTracker's output for a compromise caused by the Slapper worm [2]. After an IDS detected a suspicious process (`update`), BackTracker generated the backward graph that led to this detection point. This graph shows that the worm compromised the Apache web server (`httpd`) and caused it to execute a bash shell. The bash shell downloaded and unpacked a uuencoded tar file, which contains the Slapper executable and the source code for another program (`update.c`). The bash shell compiled this program, then ran the slapper process, which ran the newly compiled update program.

3. Bi-directional, multi-host causality

The original BackTracker tool helps system administrators determine how an attack occurred by showing the chain of events that led up to the point at which an intrusion was detected. In this section, we show how to generalize BackTracker's use of causality in two ways: track-

ing causal effects forward as well as backward, and using backward and forward causality across multiple hosts on a network. We then describe new filtering rules to prioritize the causal paths most likely to describe an intrusion as it traverses across multiple hosts.

3.1. Forward tracking

BackTracker shows which events preceding an intrusion detection point could have contributed to the modified state or event that was detected. We would like to generalize this approach to analyze in the forward direction. Analyzing causality in the forward direction answers the question "What events and state were affected by the intrusion detection point?". Consider a scenario in which an attacker replaces the `/bin/ls` executable with a program that sends a user's private files to a collection site. Once an IDS detects that `/bin/ls` was modified, forward tracking will show which files of which users were leaked as a result; this could help limit the damage done by the intrusion (e.g., by informing the user which credit cards should be canceled).

Forward causal analysis is a straightforward extension to backward analysis. Forward tracking uses the same log of system calls and objects logged by BackTracker. Whereas BackTracker adds events to a graph if they affect an object before that object's time threshold, forward tracking adds events to a graph if they were affected by an object after that object's time threshold.

3.2. Multi-host causality

BackTracker's analysis is limited to events that occurred on the same host as the state or event that was detected by the IDS. Because many attacks propagate via the network, we would like to generalize this approach to track intrusions as they infect multiple hosts. This will allow us to leverage one host's IDS alert to find other compromised hosts upstream (using backward causality) and downstream (using forward causality). As with host-level causality, this type of tracking is limited to machines under our administrative control.

To track causality across multiple hosts, we extend BackTracker's logging to also track network sends and receives. If process 1 on machine A sends a packet to process 2 on machine B, this forms an inter-host causal dependency from process 1 to process 2. Connecting a send event with its corresponding receive event requires identifying each packet (or connection). There are several ways to identify packets. The simplest is to use information in existing network headers, such as the source and destination IP address, port number, and sequence number (for TCP messages)[1]. Another approach is to supplement messages with additional information [10], either at network routers or using our modified kernel. Finally, one could identify packets by storing a hash of their contents. We currently identify packets by their source and destination addresses and sequence number; this simple approach is sufficient for the TCP-based attacks evaluated in this paper.

3.3. Prioritizing packets

Tracking the causal relationships caused by network communication can lead to extremely large graphs. Consider the following scenario: an intruder attains a login session on an internal computer (A). He then uses that login session to compromise an internal SMB server (B). From the SMB server, he browses the departmental web server, then launches a worm. Using one of the outgoing worm messages as a starting point, an administrator can generate a multi-host, backward causality graph. This graph will include the key link in the attack, which is the message from the login session (A) to the SMB server (B). However, the graph will also include the irrelevant messages from the departmental web server. To make matters worse, the graph may also include other clients that happened to affect the execution of the departmental web server before that server sent pages to the SMB server.

Another example scenario is if a network service handles a series of requests in a single process. If this network service is compromised, it will causally depend on all prior incoming requests. Fortunately, network services are often built to create a new process to handle each (or a few) incoming requests. Creating a new process helps to limit the set of incoming packets that causally precede an intrusion.

In order to counteract the tendency of graphs to explode in size, we must prioritize which packets to include in multi-host causality graphs. This is analogous to filtering host events that are less likely to be critical steps in an intrusion. Prioritizing packets leads to the same tradeoffs inherent to any kind of filtering. Even objects or events that are unlikely to be important in understanding an intrusion may nevertheless be relevant, and filtering these out may accidentally hide important sequences of events.

There are numerous methods one could use to prioritize which packets to follow in backward or forward causal analysis. The first method is a simple heuristic that works well for today's simple worms. A common pattern of today's worms is to connect to a network service, compromise it (e.g., with a buffer overflow), then immediately start a root shell or a backdoor process. For this pattern, the best process to follow when performing backward causality process is the highest (i.e., earliest) pro-

[1] The gateway that receives messages from outside the network can use ingress filtering to check that these messages do not spoof an internal network address [5]

cess in the backward graph that received a network packet. The best packet to follow for this pattern is the most recent packet received by the highest process. We call this heuristic *highest process, most recent packet*.

A more general and robust method for prioritizing packets is to choose packets that are related causally to another IDS alert. For example, an administrator may use forward tracking from a host that has detected a backdoor process. The forward causal graph from this process will show all hosts that received a message from the backdoor process, as well as the subsequent actions on those hosts that were caused (directly or indirectly) by receiving that message. Some of these messages are likely innocent or ineffective, while others may be malicious and effective. Messages that are malicious and effective are more likely to lead causally to other IDS alerts and should therefore be prioritized. These other IDS alerts may be from a network anomaly detector that points out suspicious messages directly, or they may be from a host IDS that detects suspicious host activity that results indirectly from these messages.

Another method of prioritizing packets leverages the fact that most worms repeat their actions as they traverse hosts. They usually propagate from host to host using the same exploit, or perhaps using one of a few exploits. They may also perform the same activities on each host they compromise, such as installing backdoors, patching vulnerabilities, or scanning for private information. These repeated actions form a pattern that characterizes the worm. As we causally follow an intrusion alert from host to host (forward or backward), we can learn this pattern and use it to prioritize packets that cause similar patterns on other hosts.

A pattern for a worm may be characterized in many ways. It could be the set of files or processes that causally follow from a received packet. More generally, one could characterize a worm by the topology of the causal graph resulting from the received packet. In fact, one can view the topology or membership of a packet's forward causality graph as a signature of a worm and raise an IDS alert whenever one sees this signature. Similarly, one can view the topology or membership of a network service's causality graph as a profile of that network service and raise an alarm whenever one sees an anomalous causality graph.

Finally, in some cases one can take advantage of application-specific knowledge to identify more precisely which incoming packet causes a given action. For example, a mail client may be able to inform the causality tracking system which downloaded mail message contained the attachment that is being viewed. If the viewer is compromised while viewing that attachment, the causality tracker can focus on the appropriate message.

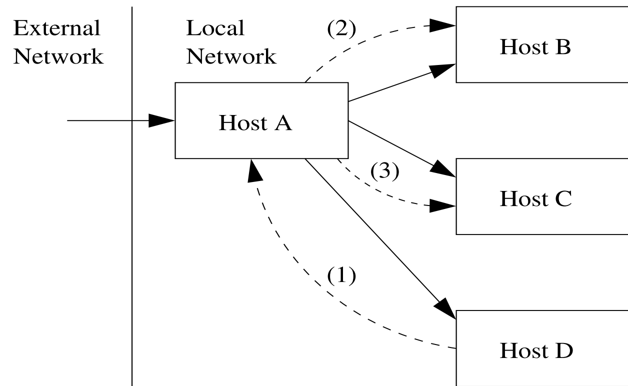

Figure 1. Inter-host propagation of the Slapper worm. The worm first infects Host A, then launches attacks against Hosts B, C, and D. After an IDS detects the attack on Host D, BDB tracks the attack backward to Host A, then tracks it forward to Hosts B and C. The solid lines depict the attacks, and the dotted lines depict the order of BDB's analysis.

4. Tracking multi-hop attacks

The prior sections introduced the mechanisms used by BDB to connect multiple alarms together. In the next two sections, we discuss and demonstrate several ways to use these mechanisms. In this section, we track attacks that traverse several hosts within an administrative domain.

4.1. Overview

Many organizations place most of their computers and services on an intranet behind a firewall, with only a few computers and services exposed to the public Internet. Relying too heavily on this type of perimeter defense has well-known flaws: if an attacker breaches the firewall (such as through an infected laptop, email virus, or bug in one of the public network services), he gains access to the computers on the local network, which are often less secured. To clean up after such a breach, the administrator must first find all the computers that have been compromised. It can be quite difficult to find which computers have been compromised during an attack, because the attacker may break into a system, steal or corrupt data, then clean up by removing telltale signs of his attack.

In this section, we show how BDB can track attacks that traverse multiple hosts, even when the intrusion is detected initially on only a single host. After an intrusion is detected on a single host, BDB tracks backward to determine what led to the intrusion on that host. Tracking

back continues across nodes until it reaches the point at which the attack entered the intranet. Finding the point the attack entered the intranet is useful in securing the network against future attacks. If the attack entered via an infected laptop, the administrator can chastise and educate the user; if the attack entered via a buggy network service, the administrator can disable that service or apply a patch [13]. After backward tracking is complete, BDB performs forward tracking to find other compromised hosts. As BDB finds compromised hosts during backward and forward tracking, it provides the opportunity to learn about the attack. The information it learns can be used to develop network and host signatures of the attack that can be used to avoid future compromises or scan other networks for compromised hosts.

As discussed in Section 3.3, prioritizing which packets to follow can be done in a variety of ways. In Section 4.2, we track the Slapper worm backward using the *highest process, most recent packet* heuristic. During the forward tracking phase, we prioritize which packets to follow by leveraging the signatures learned about the worm while tracking it backward. In Section 4.3, we prioritize the packets by following those flagged as suspicious by the Snort network IDS. In Section 4.4, we use application-specific instrumentation to identify which e-mail message led to later events on the host.

4.2. Slapper worm

We first demonstrate the ability of BDB to track attacks across multiple hosts by releasing a modified version of the Slapper worm on a local worm testbed (Figure 1). We modified the spreading pattern of Slapper to look for new victims on the local network before using its default method of searching for vulnerable machines among randomly generated IP addresses.

Our testbed contains four vulnerable web servers: one accessible from both the public Internet and the local network, and three accessible only from the local network. To make it harder to track the worm, we add background noise by running the SPECweb99 benchmark between the web servers. Two machines acted as SPECweb99 servers, and the other two machines acted as SPECweb99 clients. This workload generated 95,943 operating system objects and 1,628,937 events spread out over the four machines. The test was run for 20 minutes, with the worm being released 10 minutes after the test began.

The initial detection point occurred on Host D when an IDS detected a suspicious process named `update`; Figure 2 shows the causality graph on Host D that led to this process. The `update` process is used by Slapper to spread the worm to other machines. This process on Host D is the starting point of our backward traversal. The causality graph shows that `update` is compiled from source code, which is downloaded by a shell over a socket. The shell is started by a compromised web server.

Using the *highest process, most recent packet* heuristic, BDB identifies a packet received by the web server process as likely to be part of the attack. That packet originated from Host A and is the starting point for backward tracking on that host. Tracking the packet on Host A results in sequence of events similar to those found on Host D. However, the *highest process, most recent packet* heuristic identifies a packet that originated from an external source, so BDB's traversal backward stops.

While tracking the worm backward from Host D to Host B, BDB collects information about the worm in the form of a backward causality graph. These graphs are signatures that describe how the worm behaves on a computer. BDB uses this information to go forward and detect other instances of the worm.

The forward traversal begins with the `httpd` process on Host A since that is the starting point of the worm (Figure 3). BDB initially considers all activity resulting from this point. In addition to communicating with Host D, the worm also contacts Hosts B and C. BDB uses forward tracking to examine the effects of each outgoing packet on the receiving hosts. To determine if the worm had infected these machines, BDB searches the forward graphs for the causality signatures found while traversing backward. In both cases, the signature is found, and the hosts are deemed exploited.

BDB tracks the Slapper worm effectively using the *highest process, most recent packet* heuristic and matching host worm signatures found during the backward traversal phase. Despite the heavy load on the network service that was exploited, BDB filtered out irrelevant events and highlighted only the actions of the worm. The backward and forward graphs of all of the infected nodes contain a total of 171 objects and 251 events, which is 2-3 orders of magnitude less than the total activity on the system.

4.3 Multi-vector manual attack

Although the *highest process, most recent packet* heuristic is effective for the Slapper worm and many other existing exploits we examined, an attacker that knows about this heuristic could evade detection. For example, an attacker could implicate an innocent request by receiving it between the initial exploit and the next phase of the attack. Another difficult scenario occurs when there are multiple network services at different roots of the backward graph; in this case, there is no unique highest process.

One way to address these shortcomings is to use a network IDS to prioritize which packets to follow. While a network IDS may not be accurate enough to track attacks

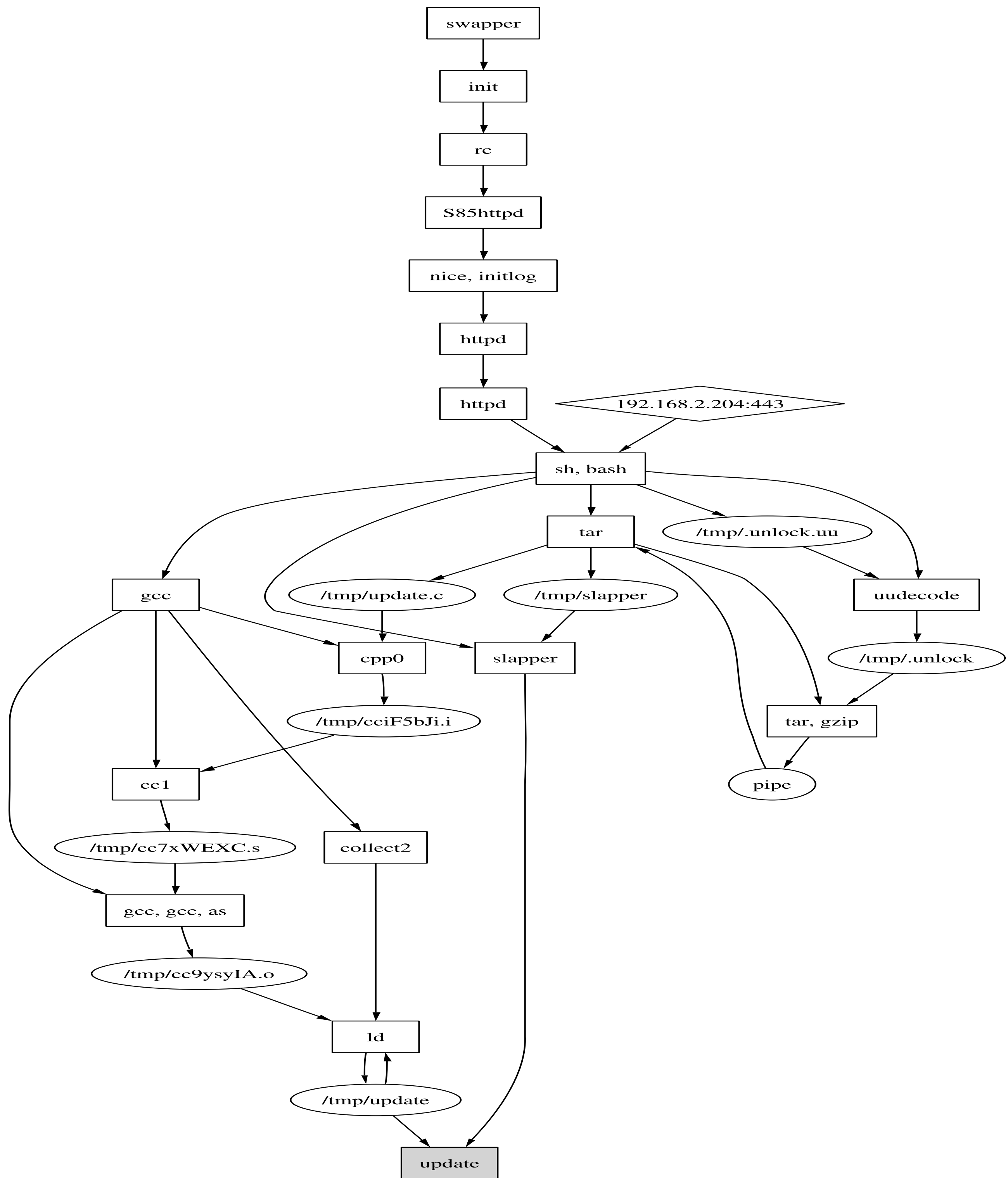

Figure 2. Backward causality graph for Slapper worm for Host D. Processes are shown as boxes (labeled by program names called by execve during that process's lifetime); files are shown as ovals; sockets are shown as diamonds. BackTracker can also show process IDs, file inode numbers, and socket ports. The intrusion detection point is shaded.

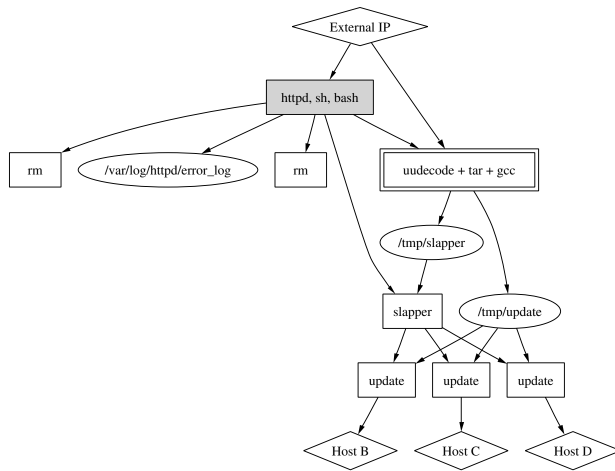

Figure 3. Forward causality graph of Host A for the Slapper attack. The subgraph formed by executing uudecode, tar, and gcc is represented by the box with double lines.

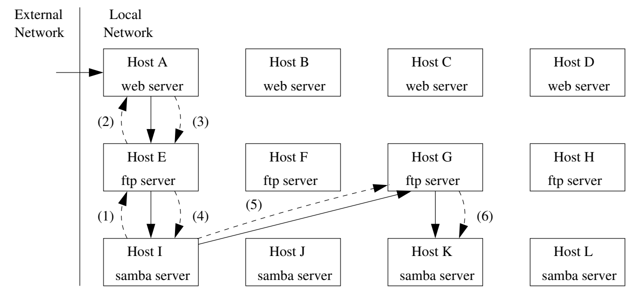

Figure 4. Inter-host propagation of the multi-vector attack. We carried out an attack on five of the 16 testbed hosts (A, E, I, G, K) using multiple attack vectors (solid lines). After an IDS detects the attack on Host I, BDB tracks the attack backward to Host E and A, then tracks it forward to Hosts E, I, G, and K.

by itself, it may be accurate enough to prioritize which packets BDB should follow. In this paper, we use the network IDS Snort [1] to prioritize packets for BDB. Snort matches messages against a number of known signatures; for this test, we use rules that find commonly used shellcode instructions. For example, Snort detects consecutive no-op instructions, which are used often in buffer overflow attacks to compensate for variations in process address spaces. Although this type of network anomaly detector is prone to false positives, it is effective enough to prioritize which causally-related network packets to examine first.

To evaluate whether a network IDS can prioritize packets effectively, we carried out an attack on our local network that exploits several network services on various nodes within our system (Figure 4). We simulate a stealthy attacker by compromising one host at a time, rather than scanning the network and attacking all vulnerable hosts. We start by breaking into a publicly accessible web server. From there, we compromise succeeding hosts through vulnerabilities in the ftp and samba servers. Each attack results in an interactive shell, and we use that shell to download tools and break into the next host.

Our testbed includes 12 hosts: four web servers, four ftp servers, and four samba servers. Each of the nodes also doubles as a client. To make it harder to track the attack, we add background noise through artificial users who are logged into each computer. Each user mounts all four samba servers in his file space. The users download source code from a randomly selected web or ftp server, then unpack this source code to a randomly selected samba mounted directory and compile it. The entire process is repeated for the duration of the test. These activities result in large amounts of noise; across all twelve nodes there are over 2 GB of network traffic, 6,589,526 operating system events and 814,262 operating system objects. The test was run for 20 minutes, and the intrusions occurred 10 minutes after the test started.

The attack is detected initially on Host I, when the attacker launches a backdoor process that opens a raw socket. BDB generates a backward graph using the `backdoor` process as the detection point, then sees which of the causally related incoming packets were flagged as suspicious by Snort. In this case, one of the incoming packets in the backward graph had been flagged as suspicious by Snort. The suspicious packet came from the ftp server on Host E. Using the suspicious packet as the starting point for another BackTracker iteration, BDB again found that one of the causally related packets on Host E's backward graph had been flagged as suspicious by Snort. This packet led us back to the public web server on Host A. All causally related incoming packets on Host A are from external connections, so the backward traversal ends here.

Starting from the external web server, BDB uses Snort to prioritize among the causally related outgoing packets, with the goal of finding other compromised hosts. The forward analysis led us back to Host E and then again to Host I. From Host I, BDB found that the attacker broke into Host G and then Host K.

In the end, BDB found all of the infected hosts and highlighted 420 operating system objects and 19 network packets of the 814,262 objects and 2 GB of network data on the entire system. BDB's resulting causal graph is small enough that an administrator who wants to understand the attack in more detail can examine each object and packet by hand.

Although BDB found all compromised hosts, there were some false positives in our analysis. In particular, Snort flagged as high priority seven packets that unsuccessfully attempted to use the samba exploit. These did not affect our analysis because none of the unsuccessful break-in attempts generated any extraneous network activity, and all of the attacked hosts were eventually broken into. To reduce these false positives, BDB could prioritize network packets further by using additional IDSs; for instance, it could see which suspicious network packets led to other host IDS alerts on the receiving host.

4.4. E-mail virus

Attacks often propagate through a network via e-mail viruses. E-mail viruses typically spread by fooling a user into running a suspicious application or by exploiting a helper process used to handle an attachment. Unfortunately, the structure of e-mail handling programs makes it difficult for BDB to track the causal relationship between incoming messages and subsequent actions. E-mail servers that receive messages and e-mail clients that display them for users tend to be long-lived and to read multiple messages at a time. Because BDB tracks causality at the granularity of a process, it assumes conservatively that all messages that have been read affect all subsequent actions. Tracking causality at a more precise granularity requires program-specific instrumentation, and this section demonstrates how one can enhance BDB in this way to track e-mail viruses.

We made two application-specific changes to support e-mail tracking across multiple hosts. First, we modified the exim e-mail server to link the network messages it receives with e-mail messages IDs. Second, we modified the pine e-mail client to inform BDB of the "current" e-mail message ID being read at a given time. We assume that the saving of an attachment or launching of a helper process is caused by the current e-mail message being read by the user. These changes allow us to track (backward or forward) the effects of an e-mail message as it is received by the server, transferred to the client, and viewed by a user. Resulting actions on a host, such as writing files, starting new processes, and sending messages, are tracked via BDB's normal mechanisms.

We tested the ability of BDB to track an e-mail virus by installing our instrumented e-mail server and client on our testbed, then releasing a virus. Figure 5 shows how BDB's resulting causality graph is able to track the virus backward from the initial detection on Host A (Figure 5a), back to when the virus was received by the exim server (Figure 5b), back to the upstream Host B (Figure 5c).

5. Correlating multiple alerts within and across nodes

In this section, we discuss the benefits of correlating multiple IDS alerts using causality, and we show how correlating alerts with causality can reduce false positives on a testbed system.

5.1. Overview

A major problem with many IDSs is false positives. One approach to reduce these false positives is to combine multiple host or networks IDS alerts into a single, higher-confidence alert. Prior approaches connect distinct alerts through statistical correlation on various features of the alert, such as the destination IP address or time of the alert. BDB makes it possible to correlate alerts in a new way by revealing which alerts are related causally.

The main benefit of relating alerts causally is its potential for increased accuracy—statistical correlation may suffer from coincidental events, whereas causal relationships are determined by chains of specific OS and network events.

In addition, using causality to correlate alerts can reduce dramatically the amount of data each IDS needs to process. For example, in Section 4.3, we showed how BDB revealed which outgoing network packets were causally related to a host IDS alert. BDB can thus narrow the search for suspicious network activity to a handful of packets, even in the midst of a busy network. In the same way, BDB can reduce the amount of data that a host IDS must examine. For example, a host IDS need examine only those processes and files that are related causally to packets that are flagged as suspicious by a network IDS. This allows one to use intensive host IDSs that would otherwise be too slow [12].

Two IDS alerts may be related causally in a variety of ways. One alert may causally precede or follow the other. Or, two alerts may share a common ancestor, such as when a single shell process executes two child processes that each perform a suspicious action. It is up to higher-level policies to determine how to score these different causal relationships, and we leave this to future work. We speculate that alerts are more likely to be correlated if one is the direct descendant of the other than if they simply share a common ancestor.

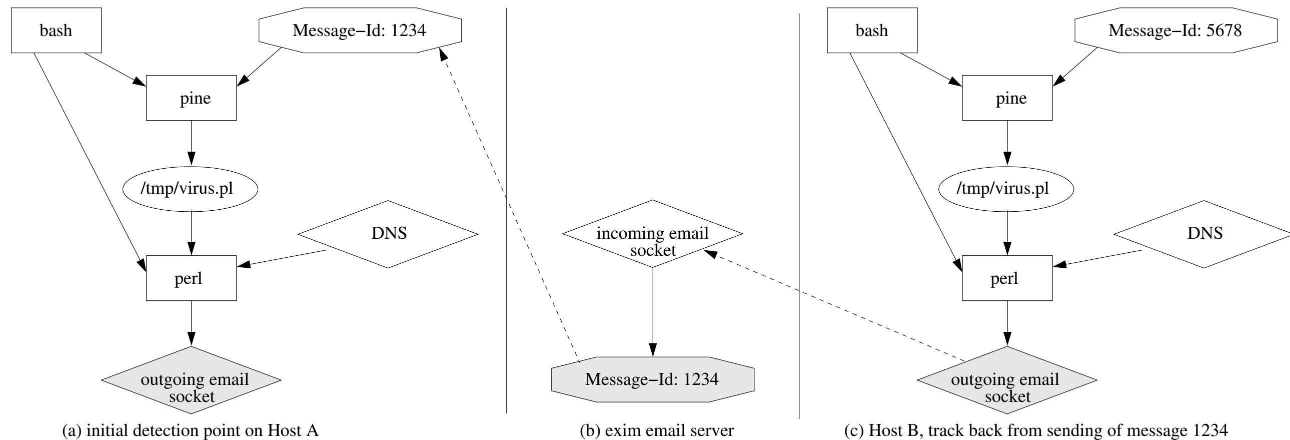

Figure 5. BackTracking an e-mail virus. In (a), BDB uses its normal tracking plus an instrumented e-mail client to track back from an outgoing network connection to the causally related incoming e-mail message. In (b), BDB uses the instrumented e-mail server to link the suspicious e-mail message with the incoming network message that delivered that e-mail message. This procedure identifies the host that sent the e-mail virus, where the backtracking can be repeated.

5.2. Results

To test how effectively causality can be used to correlate alerts, we ran BDB on two test systems that were exposed to the Internet. One system was running the default RedHat 6.2 installation; the other was running the default RedHat 7.0 installation. Both systems had several vulnerable servers that were accessible from the Internet, including the bind, ftp, and web servers. We configured the system to use a network IDS and a host IDS. We used Snort with its default rules as a network IDS, and we used a host IDS that flagged as suspicious any process that runs as root. While both IDSs are expected to generate many false positives, correlating alerts from these two IDSs using causality can increase our confidence that alerts result from actual compromises. To correlate alerts, we started with Snort alerts and generated causal graphs on the host for each suspicious network message. We then reported any processes in the resulting causal graphs that ran as root.

We ran the system for two days. [2] During this period, Snort generated 39 alerts, and the host IDS detected numerous root processes. Of these alerts, BDB detected two that were connected causally: an outgoing message flagged by Snort and two root processes. Figure 6 shows that the alerts were connected causally through a common ancestor (the httpd process), We confirmed by hand that these two alerts were indeed the result of a successful attack. Snort's alert resulted from a suspicious outgoing packet (triggering the "uid=" rule that indicates the output of the id command) that was sent by a shell process executed by a compromised web server. The shell process also downloaded various tools and gained root access through a local `modutils` exploit, which triggered the host IDS alert.

We examined the other Snort alerts by hand to see if there were any other true positives. We found one other true positive, in which the attacker compromised the web server but did not gain root privilege. This illustrates the tradeoffs involved in correlating IDS alerts. Reporting only correlating alerts reduces the number of false positives, but it may mask some true positives as well. Correlating Snort alerts with a more sophisticated host IDS may have reported both true positives.

Overall, BDB effectively reduced the number of false positives on our testbed. It also reduced administrative overhead by allowing us to use Snort's default rules, rather than customizing the set of rules by hand to reduce false positives.

[2] We also used BDB on one of our desktop computers to correlate Snort and host IDS alerts. Although we were not attacked successfully over this period, BDB was effective at filtering out the numerous false positives generated by Snort.

6. Prototype implementation and performance

BDB is a working prototype implemented for the Linux operating system. Causal tracking is implemented as two

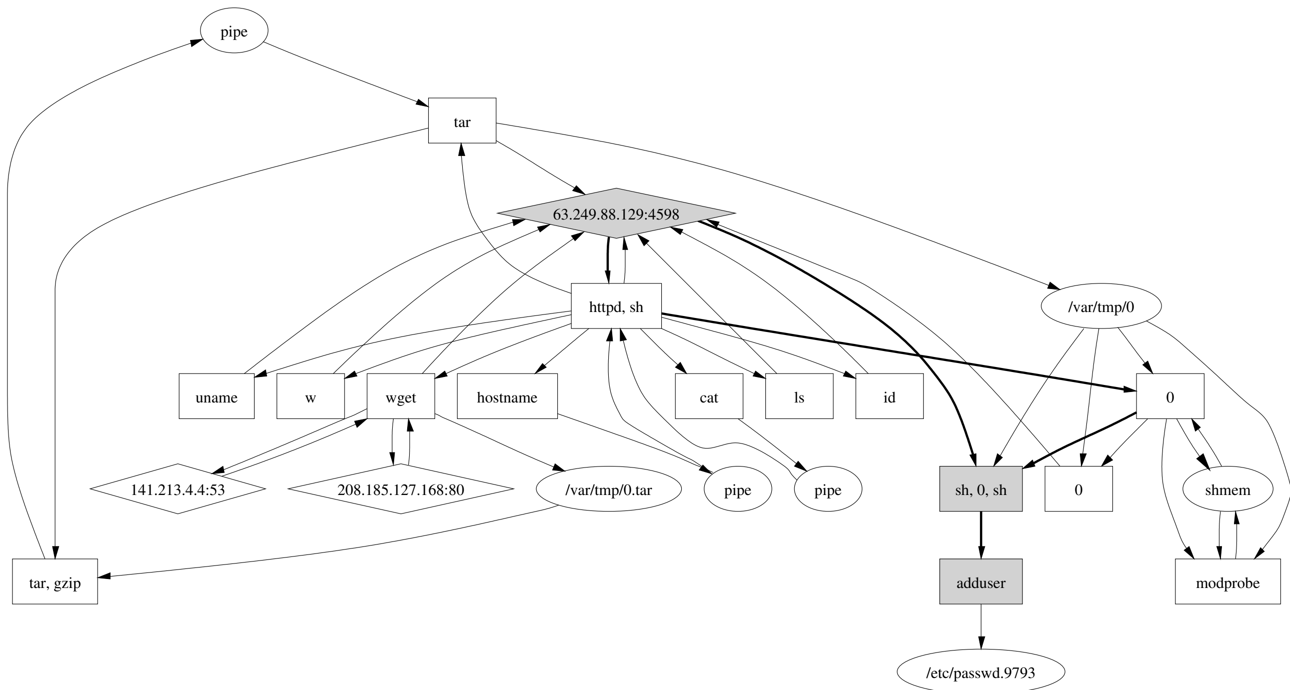

Figure 6. Correlating alerts. This figure shows how a causal graph can connect two distinct IDS alerts. The two alerts are shown as shaded boxes. One alert is generated by Snort, and the other is generated by a host IDS that detects root processes. The bold lines highlight the events that most directly relate the flagged IDS alerts.

separate modules: one for logging the specific causal events and another for generating forward and backward graphs. The logging requires modifications to the Linux operating system; the resulting logs can be written to a local file or streamed out over the network. The graph generator then parses the logs to create the graphs. BDB stores logs in a MySQL database; this allows it to generate graphs quickly even for large logs. For example, generating a graph from a log containing 2,187,963 objects and 55,894,869 events takes about 26 seconds. The time to generate graphs is proportional to the number of objects in the final causal graph rather than the total number of objects in the log. As a result, the multi-hop example discussed in Section 4.3 took less than 45 seconds to perform the complete analysis over all 12 nodes.

7. Limitations

As with most security enhancements, using causal tracking to enrich IDS alerts has its limits. First, BDB can track events only on hosts that are using its kernel modifications. Both backward and forward tracking stop when they reach a host that is not running BDB. For maximum effectiveness, an administrator should run BDB on all hosts in her administrative domain.

Second, BDB tracks only a subset of the many events that form dependencies, and events outside this set lead to covert channels. For example, BDB ignores the dependency that is formed if one process creates a file in a directory and another process lists the contents of that directory. [7] describes why it is difficult for an attacker to use these types of covert channels to carry out an attack without being tracked by BDB, but it is still possible. An attacker may also create a covert channel by leveraging an unmonitored host. BDB cannot track causality on hosts that lack instrumentation, and an attacker can break the causal chain by traversing such a host.

Another style of attack against BDB is to deliberately create noise in the causal graphs, either by generating a large number of events, or by implicating innocent processes and hosts. However, these actions may make it easier for an IDS to notice the attacker.

8. Related work

Other research projects have used the idea of causality to identify the effects of an intrusion. The Repairable File System [16] identifies potentially tainted files by tracking the flow of information across operating system events. While the Repairable File System performs forward causal analysis, it does not track dependencies across the network or filter the resulting graph to prioritize the likely paths of an intrusion. In the database area, work by Ammann, Jajodia, and Liu tracks the flow of contaminated transactions through a database and rolls data back if it has been affected directly or indirectly by contaminated transactions [3].

In addition to our approach of using causality, other techniques have been used to track multi-hop attacks. Work by Zhang and Paxson detects "stepping stones" used to carry interactive communication across multiple hops by seeing which packets have correlated size and timing characteristics [15]. Wang and Reeves extend this approach by manipulating the timing of packets to more easily correlated packets across multiple hops [14]. Our approach has advantages and disadvantages compared to prior approaches. The main disadvantage is that our approach requires one to monitor each host in the chain. Hence our approach is suitable mainly for communication within a single administrative domain. The main advantage of our approach is that it is independent of timing because it can track actual cause-and-effect chains, rather than relying on less robust characteristics such as timing. Hence our approach can track non-interactive multi-hop attacks, such as worms (even stealthy ones).

Many prior researchers have sought to correlate IDS alerts to reduce false positives [11] [4] [9]. Most prior projects correlate IDS alerts based on shared features, such as source or destination IP addresses of packets or the time at which the IDS alert occurred. Other projects use prior knowledge about sequences of actions to connect IDS alerts together into an attack scenario. Our work adds a new way to connect IDS alerts, by tracking actual cause-and-effect chains to connect prior alerts with later ones.

9. Conclusions and future work

Causality provides a new mechanism for enhancing IDS alerts. Instead of seeing each IDS alert in isolation, one can use causality to see which events and state led up to the IDS alert, and which events and state were influenced by the state detected by the IDS. BDB follows causal relationships both forward and backward and follows causal relationships across multiple hosts.

The mechanism of causality can be used in a variety of ways to increase the accuracy and reach of IDS alerts. We demonstrated how causality was able to track the Slapper worm as it spread within a local network. We also showed how causality tracked a manual attack that spread via different attack vectors, and we showed how causality could be enhanced with application-specific knowledge to track an e-mail virus. Finally, we showed how causality can correlate distinct network and host IDS alerts to reduce the number of false positives.

In the future, we plan to explore how the causal graph generated by BDB can be used as a standalone host IDS. An administrator can specify or profile a set of causal graphs for each network service, then restrict the system to generate an alert if it sees a causal graph outside of this set. We also plan to study what types of causal relationship are most likely to connect related IDS alerts.

In summary, we believe that causality is a useful way to gain additional information from existing IDS alerts. By leveraging true positives found by existing an IDS, BDB can find other hosts that were also compromised and can correlate distinct alerts to form fewer, higher-confidence alerts.

10. Acknowledgments

The ideas in this paper were refined during discussions with Michael Bailey, Evan Cooke, David Watson, and Farnam Jahanian. The anonymous reviews provided valuable feedback that helped improve the quality of this paper. This research was supported in part by ARDA grant NBCHC030104, National Science Foundation grants CCR-0098229 and CCR-0219085, and by Intel Corporation. Samuel King was supported by a National Defense Science and Engineering Graduate Fellowship. Any opinions, finding, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Snort: The Open Source Network Intrusion Detection System. http://www.snort.org/.

[2] Common Vulnerabilities and Exposures CAN-2002-0656, July 2002.

[3] P. Ammann, S. Jajodia, and P. Liu. Recovery from Malicious Transactions. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1167–1185, September 2002.

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

[5] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing, January 1998. RFC 2267.

[6] G. H. Kim and E. H. Spafford. The design and implementation of Tripwire: a file system integrity checker. In

Proceedings of 1994 ACM Conference on Computer and Communications Security (CCS), pages 18–29, November 1994.

[7] S. T. King and P. M. Chen. Backtracking Intrusions. In *Proceedings of the 2003 Symposium on Operating Systems Principles (SOSP)*, pages 223–236, October 2003.

[8] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[9] P. Ning, Y. Cui, and D. S. Reeves. Constructing Attack Scenarios through Correlation of Intrusion Alerts. In *Proceedings of the 2002 ACM Conference on Computer and Communications Security (CCS)*, pages 245–254, November 2002.

[10] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *Proceedings of the 2000 ACM SIGCOMM Conference*, pages 295–306, August 2000.

[11] A. Valdes and K. Skinner. Probabilistic Alert Correlation. In *Proceedings of the 2001 International Workshop on the Recent Advances in Intrusion Detection (RAID)*, October 2001.

[12] D. Wagner and D. Dean. Intrusion Detection via Static Analysis. In *Proceedings of 2001 IEEE Symposium on Computer Security and Privacy*, 2001.

[13] H. J. Wang, C. Guo, D. R. Simon, and A. Zugenmaier. Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits. In *Proceedings of the 2004 ACM SIGCOMM Conference*, August 2004.

[14] X. Wang and D. S. Reeves. Robust Correlation of Encrypted Attack Traffic Through Stepping Stones by Manipulation of Interpacket Delays. In *Proceedings of the 2003 ACM Conference on Computer and Communications Security (CCS)*, October 2003.

[15] Y. Zhang and V. Paxson. Detecting Stepping Stones. In *Proceedings of the 2000 USENIX Security Symposium*, August 2000.

[16] N. Zhu and T. Chiueh. Design, Implementation, and Evaluation of Repairable File Service. In *Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN)*, pages 217–226, June 2003.