CIDS: Causality-based Intrusion Detection System

Samuel T. King, Z. Morley Mao, and Peter M. Chen
Department of Electrical Engineering and Computer Science
University of Michigan
{kingst, zmao, pmchen}@umich.edu

***Abstract*— This paper presents a new style of intrusion detection system, CIDS, that links network and host based intrusion detection systems together using causal dependencies. This combination provides a number of benefits. First, combining alerts reduces false positives for both network and host based approaches. Second, host based intrusion detection systems have significantly less data to process since they only act on processes and files that are causally linked to a suspicious network packet. Third, alarms are associated with a specific network service, allowing system administrators to act accordingly after detecting a compromise.**

To show the utility of this new system, we describe how several existing intrusion detection systems benefit from using our platform, and we introduce a new host based intrusion detection system that leverages the full power of causal dependency tracking.

I. Introduction

Significant time and resources are spent making software more secure, yet the Internet continues to be plagued by malicious activity. As a result, even the most diligent system administrators must cope with the reality of computer break-ins. One way system administrators combat attacks is by using intrusion detection systems (IDS). There are a number of intrusion detection systems available at both the network and host level, and each type balances different advantages and disadvantages, but none are perfect. Network IDS has the advantage of a network-wide view for detecting anomalous traffic patterns. Host IDS has detailed knowledge of application behavior.

We propose combining network based intrusion detection systems (NIDS) with host based intrusion detection systems (HIDS) to increase the effectiveness of both. Specifically, when a NIDS detects a suspect packet, we track the effects of that packet on the host by following the causal dependencies that result from an application reading or writing the packet. Tracking causal dependencies results in a set of processes and files on the host that are affected by that network packet. Then, host based intrusion detection techniques are applied to the set of files and processes found to determine if the packet did in fact lead a successful break-in. We call this new method of intrusion detection CIDS.

CIDS provides a number of benefits when compared to existing methods. First, CIDS significantly reduces the number of false positives generated by both signature and anomaly based intrusion detection systems. Second, CIDS cuts down the amount of data HIDS have to examine by only highlighting the subset of processes and files that are affected by a specific network packet. Finally, CIDS aids in determining the exploited network service after detecting an attack.

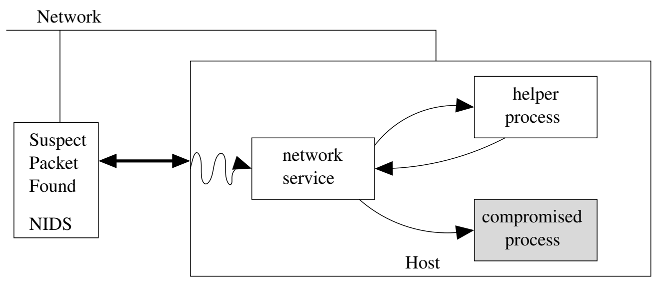

Fig. 1. Overall CIDS architecture. The NIDS queries the host upon detecting a suspicious network packet. The host then follows the causal dependencies resulting from the packet and applies HIDS rules to each of the files and processes in the set. The process deemed bad by the HIDS is highlighted.

This paper describes the benefits of combining host and network intrusion detection systems. Then, we discuss how causal tracking on the host works, followed by a description of several possible combinations of host and network intrusion detection systems. Finally, we describe a novel host anomaly detector, our prototype implementation, and CIDS performance for several attacks encountered on a honeypot system.

II. CIDS Architecture

CIDS combines network and host based intrusion detection systems using the causal tracking available via BackTracker [11] and ForwardTracker. The NIDS identifies suspect packets, then queries the host that read the packet to determine if it led to a break-in (Figure 1). For now, packets are tracked using the TCP sequence numbers and IP addresses, but [20] outlines methods suitable for tracking general network packets in our system in the presence of IP spoofing.

CIDS tracks both incoming and outgoing network packets. Suspect incoming network packets are indicative of an attack on an existing network service. The effects of that packet are followed through the host and malicious activity is detected. In addition, suspicious outgoing network packets might show that a node has already been broken into. For example, if the NIDS detects a node launching attacks on other computers, CIDS is used to determine where the malicious outgoing packets came from.

This architecture is suitable for both real-time and forensic detection of intrusions, but there are trade-offs between the two methods. Real-time detection can stop an attacker before they leave lasting effects on the system. However, an attack that is detected at the network level, but has a delayed effect on the host, might be difficult to detect. Forensic, or post-intrusion analysis does not have trouble with delayed attacks.

However, sufficient amounts of network and host data must be stored to analyze the full attack.

III. Benefits of CIDS

There are a number of benefits to combining NIDS and HIDS. In this section we evaluate the benefits of CIDS compared to stand-alone network based intrusion detection systems, host based intrusion detection systems, and the current efforts to combine the two.

A. NIDS

The CIDS architecture provides network based intrusion detection systems with an unparalleled level of semantic information. By querying the host, NIDS detect exactly which processes and files are directly affected by a specific network packet. Several different NIDS attempt to infer this information based solely on network data, where as CIDS actually monitors it directly at the host. This alleviates the need for inferring which network services handles a request and eliminates an entire class of NIDS evasion techniques. As a result, CIDS helps network based intrusion detection systems in a number of ways which we elaborate in Section V; one of the most promising is reducing false positives.

False positives are a problem with both signature and anomaly based NIDS. Signature based NIDS generate alarms regardless if the attack actually succeeded and anomaly based NIDS trigger false alarms by design. For example, we ran snort, an open source signature-based NIDS, on a /24 network in our department. After only three hours, we received alarms for every active IP address. None of the XXX machines that we were able to check had actually been broken into. Furthermore, we ran snort for 48 hours on the departmental web / MySQL server. Again there were 11956 alerts generated, none of which succeeded. In addition to tests run on our local network using snort, [16] discusses problems with commercial NIDS and false positives in detail.

These false alarms create an overwhelming amount of data for a system administrator to process and can eventually lead to a false sense of security. Although it appears that NIDS are difficult to use, they do provide some benefits. All systems had alarms associated with them, but only XX% of the packets were deemed bad. This significantly reduces the amount of data that host based intrusion detection systems must evaluate when using CIDS.

CIDS provide important feedback regarding the accuracy of the signature regardless of whether it is behavior based or signature rule based knowing the success of the attack. We propose to continuously refine the network signatures in NIDS using the host response information provided from HIDS.

B. HIDS

CIDS reduces the number of false positives generated by host based intrusion detection systems. Any processes or files examined by the HIDS directly result from a specific network packet that has been deemed bad. Combining the two alerts gives system administrators higher confidence that a break-in actually occurred.

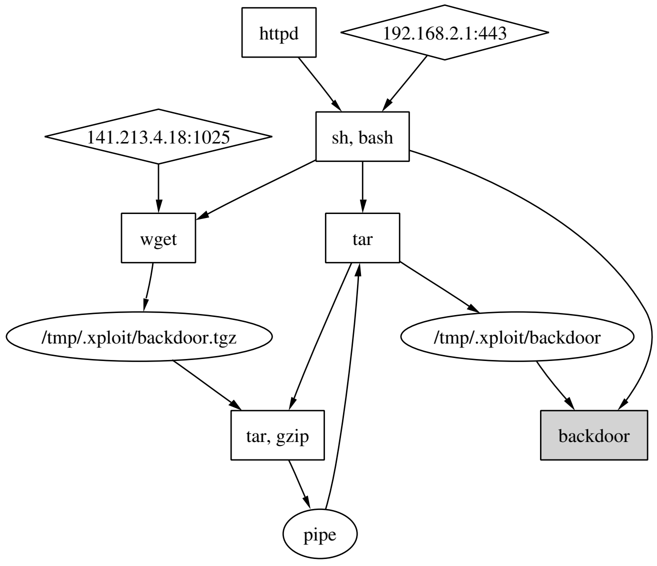

Fig. 2. Dependency graph for `httpd` attack. Processes are shown as boxes (labeled by program names called by execve during that process's lifetime); files are shown as ovals; sockets are shown as diamonds. BackTracker can also show process IDs and file inode numbers. The detection point is shaded. Note: `httpd`'s parent processes (`initlog, rc, init, swapper`) are not shown in the interest of space. Please refer to [11] for examples of full BackTracker graphs.

In addition to reducing false positives, CIDS presents the HIDS with significantly less data to process. CIDS allows slower HIDS to be effective since it only highlights the subset of files and processes affected by the network packet. This subset is significantly smaller than the total activity on the system.

Finally, CIDS connects host alarms with specific network services. Some HIDS, like tripwire which periodically scans binaries looking for Trojans, detect intrusions but cannot connect the alarm with the actual break-in. CIDS provides this connection since actions detected are associated with a network packet, and thus a network service. Determining the exploited network service is important for system administrator so they can take action and either apply a patch to the service, or shut it down all together to avoid future compromises.

C. Combine IDS

Some commercial IDS vendors have combined network alerts with host alarms. However, they seem to simply combine alarms into a single administrative console. Because there is nothing other than IP address linking the alarms together, the false positive problem could become even worse since the administrator would view alarms from both the host and the network.

UCLog[14] links alarms using temporal data. However, the benefits from this are limited; it is possible to have two unrelated alarms close in time and delayed attacks would not be linked. Also, nothing links the alarm to a specific network service, leaving the possibility for future attacks.

IV. BackTracker and ForwardTracker

BackTracker and ForwardTracker used within CIDS start from incoming and outgoing network packet respectively, and provide a graph of all relevant objects. In this section we describe how BackTracker and ForwardTracker use causal dependencies to link various operating system objects together. First, we describe BackTracker and causal dependencies. Second, we talk about ForwardTracker and how it varies from BackTracker. Please refer to [11] for a more detailed discussion of BackTracker.

A. BackTracker

BackTracker automatically identifies potential sequences of steps that occur in an intrusion. Starting with a single detection point (e.g., a suspicious outgoing network packet), BackTracker identifies files and processes that could have affected that detection point and displays chains of events in a dependency graph.

1) Tracking Dependencies: Dependency graphs are created by tracking various operating system events (system calls) and objects (files, processes). Events create causal dependencies between different operating system objects. For example, a `write` system call results in a causal dependency from the process that does the writing to the file that is written to. Also, `fork` system calls result in a causal dependency from the parent process to the child. Using causal events, BackTracker reduces the number of objects in a graph by focusing on events that are likely to be part of the attack. For example if a NIDS identifies a packet that is likely to be part of an attack, BackTracker starts with that packet and traverses backward in time beginning with the process that wrote the packet. BackTracker adds that process to the list of tainted objects and continues examining past events until the dependency graph is complete.

Figure 2 shows a BackTracker graph for an attack on a web server. The detection point for this particular attack is the `backdoor` application, which triggered an alarm from the HIDS being used. After detecting the intrusion, BackTracker uses this process as the starting point for the graph. Starting at the `backdoor` process and moving backward, BackTracker shows that the application runs using the executable file */tmp/.xploit/backdoor*. This file is written by `tar` which unpacks an archive downloaded using `wget`. All of these applications launch from a `bash` shell which spawned from the `httpd` process as the result of a successful attack. By using causal dependencies, BackTracker filters out significant amounts of non-related events and objects present on the system during the attack, resulting in orders of magnitude less data to examine [11].

B. ForwardTracker

In addition to following causal dependencies backward in time, we modified BackTracker to track forward. In essence, rather than answering the question, "how did this get here", ForwardTracker answers the question "what effect did this have". Conceptually, ForwardTracker is similar to BackTracker and they both use the same causal events. However, ForwardTracker starts from an *incoming* network packet, rather than an outgoing network packet, and tracks the effects of the process that read the packet.

V. Network Sensors

In this section, we describe in more details the state of the art of NIDS and the gained improvement using CIDS. Network intrusion detection inherently suffers from problems of false positives due to the impossibility of complete emulation of host behavior and the lack of full knowledge of which packets will actually be seen by the host. This is due to the fact that without external knowledge, NIDS does not typically know how hosts and applications are configured, how exactly applications will interpret a sequence of bytes, and how internal networks between the NIDS and the hosts are structured to determine reachability. These fundamental limitations have long been pointed out by Ptacek and Newsham [19]. In this section, we first describe several related efforts to remedy this fundamental flaw of NIDS, problems associated with these approaches, and how our approach of CIDS correct some of these problems. We use the term *monitor* to refer to the network intrusion detection system that observes the traffic to and from hosts and may alter the traffic as needed to reduce false alarms. We show that CIDS is complementary to existing approaches but enhances traditional NIDS with the ability to accurately disambiguate application protocol behavior. Given the ability to disambiguate relatively cheaply by the host to reduce false positives, CIDS is a promising new approach to improve NIDS by constructing a broader class of signatures to also lower false negatives.

A. Existing Approaches for Disambiguation

One existing proposal to deal with ambiguities at transport and network layer is traffic normalization [8]. It eliminates potential ambiguities before traffic is seen by the monitor to reduce evasion opportunities. For instance, it inserts a TCP "keep-alive" message when a RST packet is seen to ensure proper synchronization of the two TCP peers in the case of an open connection and to eliminate the ambiguity whether the TCP session is indeed closed. There are several semantic and performance problems associated with the approach of normalization. It undermines end-to-end protocol semantics; in effect it is still guessing how traffic *should* be seen by applications. This approach is very expensive, as it requires examining each packet possibly resulting in end-to-end performance slow down. The normalizer sits on the forwarding path and thus needs to be extremely reliable. It is susceptible to attacks such as stateholding and CPU overload attacks. Finally, traffic normalization is limited to ambiguities at only the network and transport layers; it can't remove all ambiguities, e.g., those of applications semantics, which CIDS can address without any ambiguities. In effect, CIDS also encompasses the functionality of the traffic normalization without any of the above disadvantages, since the host sensors have direct access to protocol state above the network and transport layers.

Bifurcation analysis [17] is a related technique that splits the analysis context int multiple threads, one for each possible interpretation. Analysis proceeds separately for each context in parallel. This can easily lead to the explosion of the number of analysis contexts, and is thus unscalable as a general disambiguation method.

A recent related work, Shield [25], uses host-based network filters targeting specific application vulnerabilities before software patches are properly installed. Positioned between the transport layer and applications, it filters traffic based on known vulnerabilities independent of exploits, thus greatly reducing false positives. The HIDS part of our CIDS framework is similar to Shield, but CIDS additionally makes use of alarms generated by NIDS to first identify suspicious traffic. Moreover, the ForwardTracker and BackTracker further disambiguate the behavior of the applications to reduce both false positive and false negatives of Shield which is complementary to our framework.

Traffic normalization relies on the specified transport and network protocol behavior as published by standards such as RFCs for the purpose of disambiguation. It may disrupt end-to-end protocol semantics, as not all implementations follow the standards and some ambiguities are inherently difficult to resolve due to differences in the semantics of application using TCP. For example, there exist application level parameters affecting the TCP/IP stack. The use of TCP "urgent" pointer can result in data delivered either via a signal or inline to the user process. Another related approach to deal with ambiguities or false positives is Active Mapping [21] which refers the technique of keeping a database cataloging the detailed behavior of individual end system protocol implementation and network topology. Based on the information in the database, the network monitor can resolve some but not all ambiguities at the transport and network layer. However, this approach is expensive and would require automatic update to the database, so that the information does not ever become out of date. Even if automation is possible, it is difficult to make it timely triggered by any changes relevant to the information in the database. Furthermore, for a large enterprise network, mapping individual hosts do not scale well. Information collected can be quickly out of date due to new software, OS upgrades, and network topology changes. Similar to traffic normalization, Active Mapping also only focuses on TCP/IP based ambiguities. It requires active probing. Arguable the information can be gathered via passive monitoring, but it would take much longer and may be less accurate. Other shortcomings of active mapping include user-controlled parameter in tcp stack, new semantics, non-deterministic packet drops, NAT, DHCP, TCP Wrappers, and potential attacks on the active mapper for the purpose of evasion.

Another proposal to reduce false positives of traditional NIDS is contextual signatures [22] focused on improving traditional string-based signature matching by augmenting the matching process with low-level context using regular expressions and high-level context. Examples include matching requests with replies, step-wise attacks, exploit scans. These types of signatures can make use of dependencies between existing signatures in so-called interdependent signatures. The disadvantages of these techniques include tedious manual instrumentation of signatures to consider additional context. Such a tedious process can miss some important contexts and thus do not completely eliminate false positives. A signature consisting of a request and a reply for example can be easily evaded if the attacker manages to the compromise the machine the single request and make the reply to evade the signature matching.

B. CIDS Advantages for NIDS

Given the shortcomings of the existing approaches to deal with the false positive issue of traditional NIDS, we now classify a set of attacks that would benefit from combining host and network IDS. One type are those OS or application version specific attacks that are difficult to generalize without trigger large number of false positives. These attacks depend on the OS or application configurations. To encompass all such attacks, the signatures or behavior-based alarms in the NIDS are required to be general, fundamentally resulting in many false positives. Active mapping can help deal with some of these problems; however, it is difficult for the database to be exhaustive. CIDS can be considered as a way to automatically update the Active Mapping database, whose entries should be timed out regularly to prevent staleness.

CIDS can handle those attacks much better than NIDS if attacks are more easily described by specifying which process is invoked in response to the sequence of network packets and host's subsequent behavior. For example, polymorphic shellcode detection [2] is difficult due to the changing exploit signatures. The state of the art of recommended detection techniques [23] suggest to set NOP_NUMBER to detect no-effect instructions to be between 80 and 90 to avoid too many false positives. Such a setting is truly ad-hoc and can also result in false negatives. Attackers knowing these threshold settings can easily evade the detection. Using CIDS, a general signature for detecting NOPS section is used and the host sensor can subsequently eliminate false positives by examining the host behavior. Similarly, attacks such as cross site scripting attacks [4] where malicous code is contained in HTML Tags in client Web requests are more easily diambiguated using CIDS given the inherent difficult of interpreting host's reponse to a sequence of code.

Another type of attacks that CIDS deal better with are those targeted at services using non-fixed ports. For example, for services using port knocking to obscure the port numbers by providing a stealthy method of authentication and information transfer to a networked machine that has no open port, it is difficult to identify which application protocol will receive the observed network packets. With the assistance from the host sensors, the process and application is easily identified and NIDS can perform retrospective network-level detection to analyze the traffic when needed.

NIDS themselves are also inherently vulnerable to attacks. Adversaries can structure network traffic that require NIDS keep significant state, e.g., record information on multiple data packets in succession. This imposes much overhead on the Network IDS, making it vulnerable to resource exhaustion and crash attacks. NIDS can offload some of such state-intensive analysis to host sensors in order to improve robustness and accuracy. Even when NIDS are overloaded and overlooks some of the suspicious traffic, it could transition in the default case of marking all traffic as potentially suspicious, leaving host sensors for further analysis. This will prevent attackers from successfully exploiting overload attacks.

One major advantage of CIDS is that we can now identify which attacks are actually successful. Those successful and failed attacks are invaluable for upgrading network signatures to refine the signature database of NIDS over time. It also provides a way to automatically generating context for contextual signatures.

VI. Host Sensors

We now examine from the perspective of host intrusion detection systems the benefit of using NIDS alarms. There are numerous types of HIDS used to detect compromises. Tripwire [10] detects modified system files, LIDS [9] prevents various system files from being modified, Sandboxing tools look for unusual system calls [18][5][1][7] or system call sequences [6], and [24][13][15] detect injected code attacks, just to name a few.

In this section, we introduce three example services that combine existing HIDS with causal tracking. These services show how existing HIDS can be enhanced by using CIDS. Then, we discuss a novel HIDS anomaly detector, CausalTrace, that leverages the full power of causal tracking rather than simply combining it with an existing HIDS.

A. *Static Analysis*

Static Analysis [24] statically determines all possible sequences of system calls an application can go through by forming a number of different models of the application. Static analysis is especially promising because it traverses all code paths at compile time and does not rely on profiling or system administrator configuration. However, the static models incur significant runtime overhead, making them infeasible for system-wide operation or on production level systems.

CIDS helps mitigate the runtime overhead of running static analysis. When the network component of CIDS highlights either an incoming or outgoing suspect network packet, ForwardTracker or BackTracker returns a set of causally linked processes and files. This set is several orders of magnitude smaller than the total number of objects on the system [11] and even separates out individual server instances. Static Analysis can then be applied to the subset of relevant objects, reducing the amount of data it must process. This reduction of data allows Static Analysis to perform system wide analysis or even work for enterprise servers despite the runtime overhead incurred for checking each individual process instance.

B. *Sandboxing HIDS*

There are currently a number of different sandboxing HIDS [18][5][1][7][6]. Sandboxing HIDS vary widely, but all share one common theme: detect unexpected system calls. Profiles of known good behavior are either specified by hand or through a training period, but each system monitors how applications and children processes interact with the underlying operating system.

This style of anomaly detector defines good behavior and triggers an alarm when bad behavior is encountered. Defining good behavior allows sandboxing HIDS to detect unknown attacks, but also allows for false positives. For example, if not all code paths are traversed while profiling, an alarm triggers when a new, yet correct, code path is taken for the first time.

CIDS reduces the number of false positives resulting from Sandboxing HIDS by linking the alarm with a network alert. If a suspect network packet causes a NIDS alarm, the effects of the packet are tracked on the host and the resulting set of files and processes are evaluated by the Sandboxing HIDS. Because BackTracker and ForwardTracker separate out only the processes and files directly affected by the specific network packet, system administrators have higher confidence that the suspect network packet actually lead to an intrusion and that the host alarm is a true positive.

C. *LIDS*

LIDS hardens the Linux kernel and grants root users fewer privileges than a standard root user in UNIX systems. LIDS restricts system wide activities and can be configured to disallow non-trusted kernel modules and prevent modifications of system binaries. This level of protection prevents attackers from installing Trojan binaries, even when they get root access to the machine. However, the detection occurs *downstream* from the intrusion. For example, if an attacker breaks into a web server, spawns a bash shell, and attempts to install a root kit using a cron script, LIDS will prevent the root kit from being installed but cannot determine which network service was exploited. Fortunately, CIDS is able track the effects of a network packet and can link the attempted root kit installation with an exploited network service.

D. *CausalTrace*

In addition to using existing methods of host intrusion detection, we developed a new host anomaly detector based on BackTracker and ForwardTracker graphs called CausalTrace. BackTracker and ForwardTracker graphs provide a system-wide view of the attack and capture information that is not available to other HIDS. For example, a typical HIDS would most likely allow network services to write files in /tmp. However, if an attacker corrupts data in /tmp that is used by other applications, typical HIDS do not follow the causal link. Fortunately, BackTracker and ForwardTracker graphs capture the tainting information and use it as a means for detecting suspicious activity. The main observation is that attacks on network services interact differently with the operating system than typical network services.

Typical network services tend to have specific interactions with the underlying operating system. For example `ftpd`, `smbd`, and other file servers read and write files; `httpd` reads files, launches CGI scripts, and communicates with databases; and instant messaging clients relay text messages to various servers. These interactions are captured by CausalTrace and used to constrain attackers. Figure 3 shows the ForwardTracker graph for a typical instance of a ftp server. The ftp server writes files and uses helper processes that pipe information back to

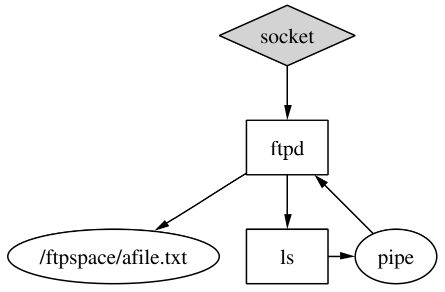

Fig. 3. ForwardTracker Graph for a typical instance of the `wu_ftpd` ftp server.

the main server process. As a result, a successful attack on the ftp server would be constrained to performing only the actions shown on this graph. Typical exploits are constrained to begin with (e.g., limited stack space for buffer overflows), so limiting attackers further helps prevent hackers from gaining full control of the system, even after a successful attack.

In addition to constraining hackers, CausalTrace detects attacks against CIDS itself. One possible way a hacker could attack CIDS is by forcing everything on the system to be tainted. However, this is not typical behavior for network services and is easily detected by CausalTrace.

1) Prototype Implementation: To implement CausalTrace, we use a set of policy rules for each process in a causal dependency graph. These rules define allowed causal events for each process and are used to enforce the constraints of a BackTracker or ForwardTracker graph. For example, the `ftpd` process shown in figure 3 would only be allowed to call `fork`, write to */ftpspace/afile.txt* and read from a pipe. Also, the forked process could only call `exec` on `ls` and write to a pipe. As a result, constraints placed on the causal graph are almost identical to full graph matching, but algorithmically more efficent to compute.

To support CausalTrace, we implemented policies that are predicated based on the fork chain of each individual process. Applying different rules to children processes is similar to Sandboxing HIDS like MAPbox [1] and BlueBoX [5]. Child-boxes from MAPbox and inherited policies for BlueBoX apply different rules to children processes, however, our approach allows for even more fine grained control. For example, if a Web server uses `bash` for both a CGI script and a helper process for a different CGI script, the two `bash` instances have distinct privileges.

Furthermore, fork chains can have various root processes depending on the causal relationship. Most processes will have the specific network service for the root process in the fork chain. However, if a process that is not a direct descendant of the network service is causally related to the network service, it will have a different root process. Having a range of root processes allows CausalTrace to profile processes based on the full causal relationship.

As a result, existing processes within the causal graph are constrained using rules predicated on the location within the graph. If the behavior of these processes varies, an alarm is raised. Also, any unknown fork chains are flagged. This style of sandboxing is a good approximation of a full BackTracker or ForwardTracker graph. Since files do not have rules applied to them, there is the possibility for ambiguities. However, there are still significant additional constraints placed on the system and it is effective at detecting attacks.

VII. Remaining Challenges

CIDS is a working prototype that is effective for many different cases. However, several unanswered research questions remain. First, we have to address network service interaction. For example, a single computer may double as a web server and a ftp server. Users upload web files using ftp, then the web server reads these files using the local file system. This scenario may lead to significant tainting of the system for certain network packets. One simple approach is to isolate various network services using virtual machines. Virtual machines add little or no overhead [12][3][26] and can be used to isolate the actions of network services from one another. This is just one possible solution, other methods might also be effective.

Many network services we examined (`ftp`, `smbd`, `named`, and `httpd`) have specific interactions with the operating system that can be summarized by a ForwardTracker or BackTracker graph. One exception is the `sshd` network service. `sshd` is a general purpose secure remote login application. Making an accurate ForwardTracker CausalTrace signature requires allowing behavior that is so general, it would be trivial for an attacker to hide within it. Fortunately, `sshd` is the only network service we encountered with this level of generality. There are heuristics that could allow `sshd` to benefit from the added constraints imposed by CIDS, but this is the topic of ongoing research.

VIII. Conclusions

Both host and network based intrusion detection systems are prone to having false positives. By using CIDS, network and host alarms are causally linked, giving system administrators more confidence that intrusions have actually occurred. NIDS are able to query the host and have access to a level of semantic information not previously available. HIDS have significantly less data to evaluate, opening the door for more sophisticated detection algorithms. By bridging the gap between network and host based intrusion detection systems, CIDS introduces a promising new technique in the defense of computer systems.

References

[1] A. Acharya and M. Raje. Mapbox: Using parameterized behavior classes to confine untrusted applications. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[2] K. H. at The SANS Institute. What is polymorphic shell code and what can it do? . `http://www.sans.org/resources/idfaq/polymorphic_shell.php`.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the 2003 Symposium on Operating Systems Principles*, October 2003.

[4] C. C. Center. CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests. `http://www.cert.org/advisories/CA-2000-02.html`.

[5] S. N. Chari and P.-C. Cheng. Bluebox: A policy-driven, host-based intrusion detection system. *ACM Trans. Inf. Syst. Secur.*, 6(2):173–200, 2003.

[6] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In *Proceedings of 1996 IEEE Symposium on Computer Security and Privacy*, pages 120–128, 1996.

[7] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer. A Secure Environment for Untrusted Helper Applications. In *Proceedings of the 1996 USENIX Security Symposium*, pages 1–13, July 1996.

[8] M. Handley, C. Kreibich, and V. Paxson. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics . In *Proc. USENIX Security Symposium 2001.*

[9] X. Huagang. Build a secure system with LIDS, 2000. http://www.lids.org/document/build_lids-0.2.html.

[10] 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.

[11] 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.

[12] S. T. King, G. W. Dunlap, and P. M. Chen. Operating System Support for Virtual Machines. In *Proceedings of the 2003 USENIX Technical Conference*, pages 71–84, June 2003.

[13] V. Kiriansky, D. Bruening, and S. Amarasinghe. Secure Execution Via Program Shepherding. In *Proceedings of the 2002 USENIX Security Symposium*, August 2002.

[14] Z. Li, J. Taylor, E. Partridge, Y. Zhou, W. Yurcik, C. Abad, J. J. Barlow, and J. Rosendale. UCLog: A Unified, Correlated Logging Architecture for Intrusion Detection. In *Proceedings of the 12th International Conference on Telecommunication Systems - Modeling and Analysis (ICTSM)*, 2004.

[15] G. C. Necula, S. McPeak, and W. Weimer. Ccured: type-safe retrofitting of legacy code. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 128–139. ACM Press, 2002.

[16] D. Newman, J. Snyder, and R. Thayer. Crying wolf: False alarms hide attacks, 2002. http://www.nwfusion.com/techinsider/2002/0624security1.html.

[17] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, December 1999.

[18] N. Provos. Improving host security with system call policies. In *Proceedings of the 12th USENIX Security Symposium*, August 2003.

[19] T. H. Ptacek and T. N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, Inc., 1998.

[20] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Network support for ip traceback. In *ACM/IEEE Transactions on Networking, 9(3)*, June 2001.

[21] U. Shankar and V. Paxson. Active Mapping: Resisting NIDS Evasion Without Altering Traffic. In *Proc. IEEE Symposium on Security and Privacy*.

[22] R. Sommer and V. Paxson. Enhancing Byte-Level Network Intrusion Detection Signatures with Context. In *Proc. ACM CCS 2003*.

[23] N. G. S. Technologies. Polymorphic Shellcodes vs. Application IDSs. `http://www.cgisecurity.com/lib/polymorphic_shellcodes_vs_app_IDSs.PDF`, 21 Jan 2002.

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

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

[26] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of the 2002 Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.