Proceedings of the 2005 IEEE
Workshop on Information Assurance and Security
United States Military Academy, West Point, NY, 15–17 June 2005

Towards a Third Generation Data Capture Architecture for Honeynets

Edward Balas and Camilo Viecco
Advanced Network Management Lab
Indiana University

Abstract— **Honeynets have become an important tool for researchers and network operators. However, their effectiveness has been impeded by a lack of a standard unified honeynet data model which results from having multiple unrelated data sources, each with its own access method and format.**

In this paper we propose a new data collection architecture that addresses the need for both rapid comprehension and detailed analysis by providing two data access methods: a relational model based fast path, and a canonical slow path. We also present a set of tools based on this architecture.

I. Introduction

A Honeynet is a network of high interaction honeypots[1]. High interaction honeypots are quite different from low interaction honeypots such as Honeyd [2] for they provide a full operating system and set of software for an intruder to interact with. This high level of interactivity is a desired because it allows researchers the ability to observe the behavior of an intruder in a live system, and not a simulation. As a result, high interaction honeypots are well suited to capture new or unanticipated activity. However, high interaction honeypots collect a larger volume detailed data from multiple data sources making it difficult to manage honeynets and make sense of the collected data.

To help facilitate honeynet deployments and the sharing of information between researchers, The Honeynet Project standardized the GenII honeynet architecture[3]. This architecture includes a specification of Data Capture procedures whose purpose is to *"log all of the attacker's activity"*. The GenII Data Capture procedures specify the collection of three types of data: firewall logs, network traffic and system activity. Figure 2 provides a schematic representation of a typical Gen II deployment. This architecture does not provide any guidance on how to store or access the captured data.

In the standardized architecture, firewall logs are used to provide a summary of the network activity. The "rc.firewall" script provided by the honeynet project allows this by using the Linux IPTables[4] connection tracking capabilities. We feel this logging is counter-intuitive because firewall logs are typically used for policy auditing and in this case they are being used to provide summary accounts of network activity. In addtion, these summaries lack needed detail such as the duration and quantity of network activity

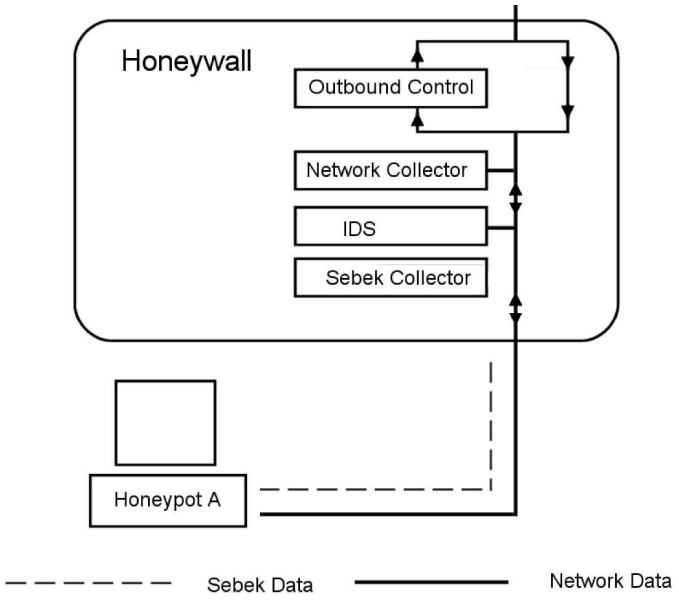

Fig. 1. GenII Honeynet Data Capture.

Network traffic and Intrusion Detection System(IDS) events are captured using the Snort IDS system[5]. For Data Capture, two instances of are executed, one to merely record the raw traffic, and the other to examine the network traffic looking for events that are indicative of misuse or intrusion.

System activity refers to monitoring activity from the perspective of each high interaction honeypot. This type of monitoring includes two types of data: Syslog and Sebek. Syslog data is provided by each honeypot's operating system. Sebek is a tool developed by the Honeynet Project to monitor the behavior of intruder even when the intruder uses session encryption[6]. Sebek operates as hidden kernel module which covertly exports log data to the logging system.

The GenII honeynet architecture gathers very detailed data but it suffers from a number of limitations. We assert that there are three problems with the existing GenII

ISBN 0-7803-9814-9/$10.00

design.

First,the GenII data ontology is very coarse and excessively rigid. The three *"layers"* as defined in the architecture are directly tied to the tools that gather the data rather than to the types of data collected. This poses a problem for tools that collect both system and network data, and makes it difficult to integrate new subtypes of data.

Second, while GenII honeynets do collect deatailed data, this data is largely unstructured within a data source. For instance, while we know that the apparently unstructured packet capture data does have a structure which can be represented using a network flow abstraction, these flows are not explicitly stored in the data and thus need to be rediscovered each time the data is examined.

Lastly, the data storage is unorganized; each data source is stored in a format defined by the tool gathering the data with no relationship to the data gathered by any other tool. As an example, lets assume we want to identify all IDS alerts related to a network connections containing a bidirectional flow of data. To do this, a tool would need know how to read and parse pcap data, then it would have to process the pcap data to recreate the set of network flows. Next, assuming the tool is capable of processing snort IDS logs, the tool would need to take the matching network connections and search through the IDS data looking for matches based on the key attributes of each connection. This lack of unification is a problem for several reasons. First, every time we want to examine the network traffic from a network flow or connection level of abstraction we need to process the raw traffic data, causing an unnecessary inefficiency. Second, because the data is stored in a data format which is implicitly defined by the tool gathering it, the addition of a new tool such as the Bro IDS[7], introduces a new data format which is incompatible with existing analysis tools. This forces every analysis tool to be partially rewritten for each new data source.

To solve these problems we propose a third generation of data capture architecture for honeynets. This new architecture includes a hierarchical ontology, a relational data model based on the new ontology and a consistent data access method.What follows is a description of the proposed architecture and an implementation based on the architecture.

II. Our Approach

The data capture architecture needs to define a data model which is independent of the format of the data source and which reflects the conceptual structures involved. Our approach is based on the following abstractions: hosts, processes, network flows, and files. The relationships between these structures is illustrated in figure 2.

Hosts represent individual instances of a running operating system. Hosts contain Processes and are typically

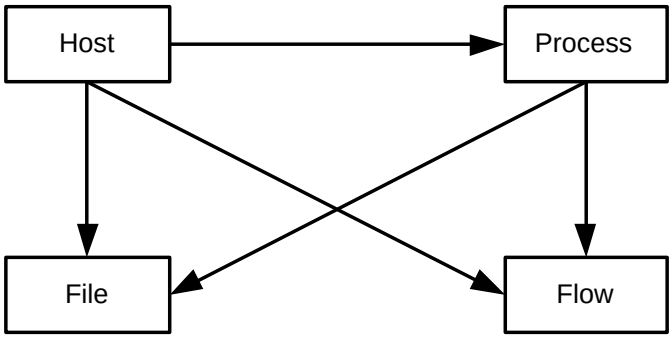

Fig. 2. Data Model Relationships.

associated with an IP address. While the association of IP address to Host is unreliable due to NAT etc, from the perspective of the Honeynet it is sufficient, and since the end point which is a Honyepot is a known entity and this relationship can be reliably discovered. Hosts provide an abstraction around which we can group all other data, network flow data, process data and file data. Data from every data source contributes to the representation of a host.

Processes are an executing instance of a program which interacts with the host using system calls. These system calls include the ability to access files and communicate with other hosts. The process abstraction is suited for organizing system call activity. Such activity includes socket calls which can create or terminate network flows and read, write, and open calls which effect files.

Files are what one would typically think of as data residing on the hard drive of a system. It is a contiguous set of binary data stored as a single unit. They typically have a name, and are referenced within the operating system with some form of identifier such as inode.

For the purposes of our model , we are defining a network flow as a possibly bidirectional communication between two hosts which involve a pair of end points. These endpoints are defined by the IP address, IP protocol number, and depending on protocol, the port number. Flows can be related to a host's process data via the socket system call.

A. Objectives

There are four objectives we wish to address with he new architecture:

The data model should be directly based on this ontology and strive to be independent of data source. In the GenII model, adding a new data capture instrument results in a new data format, and every analysis tool needs to be modified to use the new data source. While it is unlikely that we can eliminate the need to modify tools to fully benefit from a new data source, we feel that we can limit the amount of work by keeping the data access and encoding methods consistent. The need for interoperability between functionally equivalent tools provides another

situation where having a source independent standard data model is necessary.

The data model should be capable of expressing the relationships between the different types of information in the model. As an example, a network flow is going to be related to a host and process, the model should make it easy to identify that relationship. If we were monitoring system call activity on a honeypot we should be able to relate a network flow to a system call made by a process on one of the hosts. This type of monitoring is key to programmatic identification of incident causality[8].

The data should be added to the composite data store on a continual basis as close to real-time as practical. This allows the system to preprocess the data into the data store as the data comes in rather than trying to do it at the moment an analyst asks for data, thus reducing the work required from multiple examinations of an event.

Lastly, it is necessary that a programmatlly consistent data access methods is provided. In order to allow researchers to automatically share data between Honeynets, it is not sufficient to simply standardize how we collect data, but also the methods of representing and sharing data.

B. Fast Path and Slow Path

During the construction of our model we observed that there are two competing needs of the analyst. One is a high level understanding of the behavior of the intruder inside the honeypot. This is the need to have a grasp of the full scope of an intrusion which covers the basic who, what, where and why questions. The other,is the need to recover information at the most detailed level available in order to accurately understand a specific technique or behavior. An example of the latter is the need to understand the exact actions taken within an exploit or the need to reverse engineering a recovered malware tool. To satisfy these competing needs we propose the separation of the data model and access methodology into two data paths.

For the high level comprehension, we provide what is know as fast path data access. Fast path access provides a unified view of events in a relational form, with some degradation in detail. This data guides the analyst toward interesting events which may require more detailed analysis and provides a solid foundation for macroscopic efforts such as trend analysis. The fast path data is stored in a unified relational model.

For detailed inspection of the "interesting" events a slow path data access method is provided. The slow path access provides the level detail needed for forensic type investigations, malware analysis, or packet analysis, where accuracy and detail are paramount. Slow path data is stored in its canonical form. Pcap files are the only canonical form since they not only contain all the information extracted by the data collection tools but they also contain all the data necessary to regenerate all our derived data sources [1].

We propose that this hierarchical model can be represented relationally. We see this as desirable for two reasons: first the composite data model we are proposing is highly relational; second, a relational database system can provide the desired programmatic consistent data access.

III. Our Implementation

The implementation described is the basis for a Honeynet data capture and analysis system under development for the next generation Honeywall[9]. For this paper we provide a brief introduction to the next generation data capture and data analysis capabilities, the heart of which are at the core of this paper. In this section we will briefly cover our enhancements to the GenII data collection architecture, our data fusion methodology, and we will examine analysis capabilities enhanced with this approach.

A. Data Collection

Key to our making use of the proposed model but absent from the GenII design are multiple data types which are not explicitly collected within the GenII architecture. This section addresses how we collect each of these new types of data and why they are important to create a contiguous composite view of intrusion sequence.

Within the GenII architecture IPTables is used to approximate network flow monitoring with its connection tracking capability. However, while providing some aspects of the flow concept,this capability does not provide valuable details such as quantity of data transfered, bidirectionality information or end time of a network connection. Such information is desired to estimate the interest of a connection, for example one might be more interested in a bidirectional flow lasting 10 seconds than a flow lasting under a second and consisting of a single packet.

To improve the collection of flow data, we turned to Argus[10], a tool designed to provide flow monitoring as defined by the IP Performance Measurement Working Group[11][12]. Argus provides network flow records which contain summary detail not provided by IPTables connection tracking, such as the start and end of the flow, number of bytes transmitted in each direction and number of packets transmitted in each direction. Similar in design to snort, Argus collects the canonical network traffic data using libpcap, and it then processes this data to create a derived data source. This data source is used to populate flow related fields in the relational model. Unlike the Connection Tracking logs, this data does describe the quantity of data and duration of each flow.

To augment our understanding of the network activity and the hosts at either side of a communication, we added passive operating system fingerprinting capability,

[1] With the expecption of IDS alerts, since in order to recreate them we also need the rule set used at that particular time

provided by the p0f[13] tool. P0f is also a pcap based monitor that provides an estimate of the operating system(OS) used by host that initiates a TCP connection. This data is useful for two reasons. First, across flows it allows one to see if the apparent host OS is changing for a given IP source providing an indication that the host might be behind a NAT. Second, OS identification can improve the accuracy of IDS events through the process of passive alert verification[14][15]. For instance in a situation where a apache mod_ssl exploit[16] is launched against a non-linux host, the system could detect this discrepancy and treat the alert with a lower priority similar to the approach taken by RNA[14][2].

The addition of the Argus and p0f data to the Snort and packet capture data provides a more comprehensive representation of events than provided in the GenII design. Further this new data can be organized around the concept of a network flow. However additional data sources are needed to bridge the relational gap between the network flows and processes on a host.

To bridge this gap we enhanced Sebek [6] to monitor network activity from the host's perspective. Sebek is a kernel based data capture tool designed to be installed on high interaction honeypots [1]. Balas modified Sebek to monitor socket, process and file activity [17]. These modifications provided three necessary capabilities.

First, Sebek was enhanced to monitor socket activity. Whenever a honeypot accepts or creates a network connection, Sebeck records the IP level attributes as well as the corresponding host, process and inode. This allows us to relate a network flow to the specific open inode and file descriptor used by a process to service the connection. This data is integral to providing a composite view of the incident that transcends flow and host data. Once a network connection associated with an intrusion attempt is observed, we immediately know which inode and process the intrusion was tied to. Using this data we can quickly identify related information such as the keystrokes captured by Sebek.

Second, Sebek was enhanced to monitor process creation. This monitoring allows us to relate one process to another, rebuilding the process tree. This is important in intrusion analysis for it allows us to track the intrusion forward from the point of intrusion identifying all processes created, and any other causally related system activity, such as outbound network connections[8]. The same capability can be used in reverse, if we see an outbound connection on a honeypot, we can back track to identify the point of intrusion.

Lastly, the ability to monitor the opening of files was added. Coupled with the process tree this allows us to identify all files accessed as part of an intrusion. This knowledge can in turn be used to prioritize data analysis efforts. As an example, presume that a specific intruder likes to place his/her files in a unique location in the file system. Once this location is identified, we can quickly search preexisting data for any prior indications of the same intruder's presence. This capability can also be used to create a crude form of Honeytoken[18] where the act of accessing a certain file might be deemed an interesting event requiring further investigation.

[2]p0f can only estimate the OS of the TCP initiator, in this example the OS of the host under attack is known by either manually introduction of the OS by part of the administrator as with a honeypot or through previous TCP connections initiated by the particular host

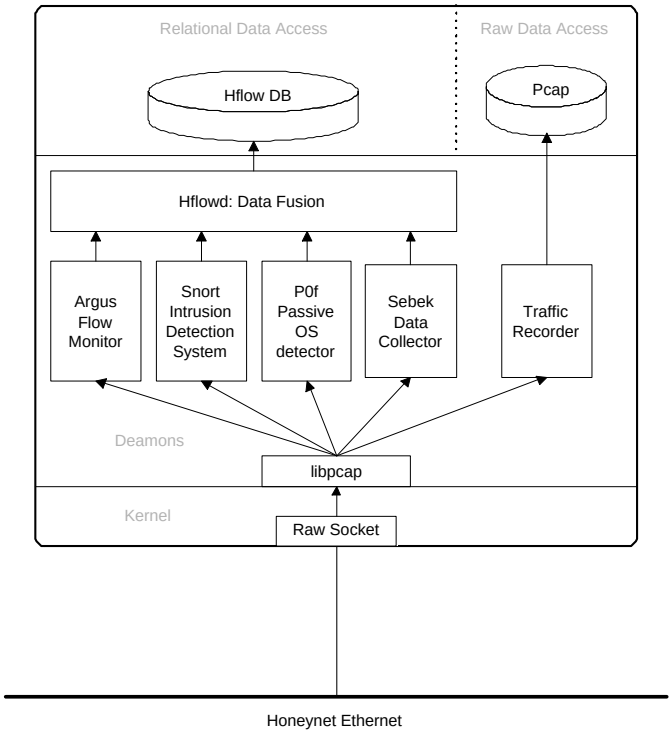

Fig. 3. Data collection and fusion diagram

B. Data Fusion

Hflow was developed to combine each of these data sources into a composite relational model. It continually consumes data from each source, fusing it based on identifiable relationships and it then loads this data into a database.

Hflow receives Argus flow, Snort IDS, p0f OS fingerprints and Sebek data. This data once combined is then inserted into a database.

Flow related data, such as Argus and Snort, are correlated based on corresponding tuples consisting of the IP protocol number, the source and destination IP addresses and if applicable port numbers which fall within the same time period. This is similar in approach to the way an operating system determines which socket a packet is related to.

Process data is gathered by Sebek and organized based on the process ID. For every activity monitored by Sebek both the process ID and parent process ID are recorded. This combination allows Hflow to recover the process tree. One concern with the process ID is that it is a counter that can roll over. While the process ID value eventually rolls over, during the same period in time no two processes on a single host will share a process ID. Further, in the unlikely event that 2 processes share the same process id in close time proximity, it is even more unlikely that the 2 processes will have the same combination of process id and parent process id.

File data is linked to process data by monitoring system calls that provides us with the process ID of the acting process, and the inode number of the file being acted on. By monitoring the open and read system calls, we are able to map process activity to a specific file.

The result of Hflow data fusion is a unified data set with identifiable relationships between the logical categories. This data is exported in the necessary format to be uploaded into a relational database in a continual basis.

However, the database is a reduced representation of the complete system activity record. As a result, we also store the networks full packet capture to provide detailed analysis of process flows and to have a backup of the canonical data source.

To bridge the gap between the slow path packet capture data and the fast path data, the *pcap_api* was developed, this tool allows analysts to download a dynamically generated pcap file using the database ID corresponding to a flow in the Hflow database as input. When an analyst is examining the fast path data and identified a network flow requiring closer examination, the *pcap_api* is used to extract only the raw data that matches in time and IP header info defined by the referenced flow.

The mapping of each data capture tool to the categories of data produced can be seen in table I.

TABLE I
MAPPING CAPTURE TOOL TO MODEL CATEGORIES

Tools	Flow	Host	Process	File
Argus	Yes			
p0f	Yes	Yes		
Snort	Yes			
Sebek	Yes	Yes	Yes	Yes

C. Data Analysis

To demonstrate the utility of of the fast path and slow path data model, we developed *Walleye*, a web based Honeynet data analysis interface. The purpose of the interface is not to be a comprehensive or monolithic analysis platform. It is designed to facilitate intrusion sequence comprehension thought the presentation of a sequence-centric, composite view of the data. By this we mean that the view of the data we provide is a composite of each of the raw data sources. If successful, what the analyst perceives is the greater than any with any single source of data. It is hoped that the composite view provided by Walleye, which spans multiple data source will improve the analyst's capability to quickly perceive the intrusion sequence. This allows the analyst to look not just at an IDS event, but look at that event in the context of the effects of the event on system activity and side effects. This context exists today but requires manual effort to identify. By using the relational model we can automatically identify this context and provide it to the analyst, ideally improving accuracy and efficiency, by removing the need to manually identify relations and compiling the composite event view.

Figures 4 and 5 provide illustrations of how Walleye uses the relational model to generate composite event views. In the first illustration, we see a composite view of different data srouces: p0f, Argus and Snort organized into a single unit or group. In the second we show how we can use data from across groups to provide a comprehensive overview of an intrusion sequence.

Figure 4 shows the current representation of a flow. From the image we can observer several features: (*i*) This is a bidirectional ICMP flow, whose initiator is 10.0.1.13, (*ii*) 2655 different alerts are collapsed into a unified picture (three different alerts launched 855 times each at the right side of the picture), (*iii*) 877 packets where sent in each direction during this flow and (*iv*) the OS fingerprint is unknown for this connection. This chart combines information from three different data sources: Argus, P0f and Snort and presents them to the analyst in a related and aggregated fashion.

Figure 5 shows the tracking of an intrusion sequence across two honeypots. The process tree diagrams are automatically generated by walleye based on the host and process data, and can also use to flow data to relate activity on multiple honeypots. In this visualization, we see 2 types of objects, processes and IDS alerts. Processes identified by the Host and PID columns followed by the list of command names the process executed as. Processes which at some point executed with root privileges have a solid white background, those which are non-root have shaded background on the row headers. IDS alerts are represented as 2 row and typically longer rectangles, which display the class of the alert and the specific alert. Also of note in the graph is the directional arrows, these show the progression of the incident.

The referenced graph was generated as a result of a staged intrusion where we manually broke into our own honeypots, as part of this we also preinstalled a few tools to keep the graph sufficiently simple for illustrative purposes. In the Figure 5 illustration we a intruder gain access to Honeypot *.*.*.25 using a preinstalled bintty backdoor.

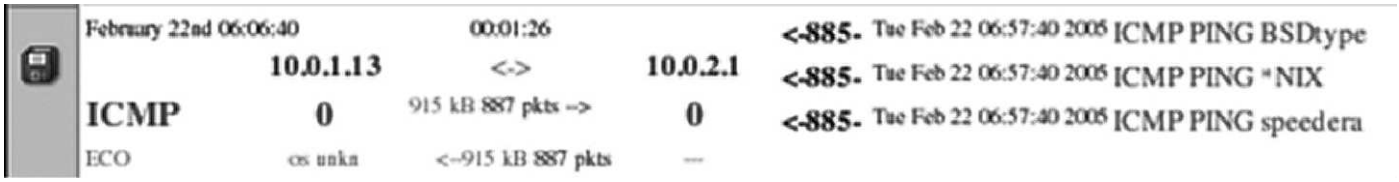

Fig. 4. Flow abstraction as used by Walleye

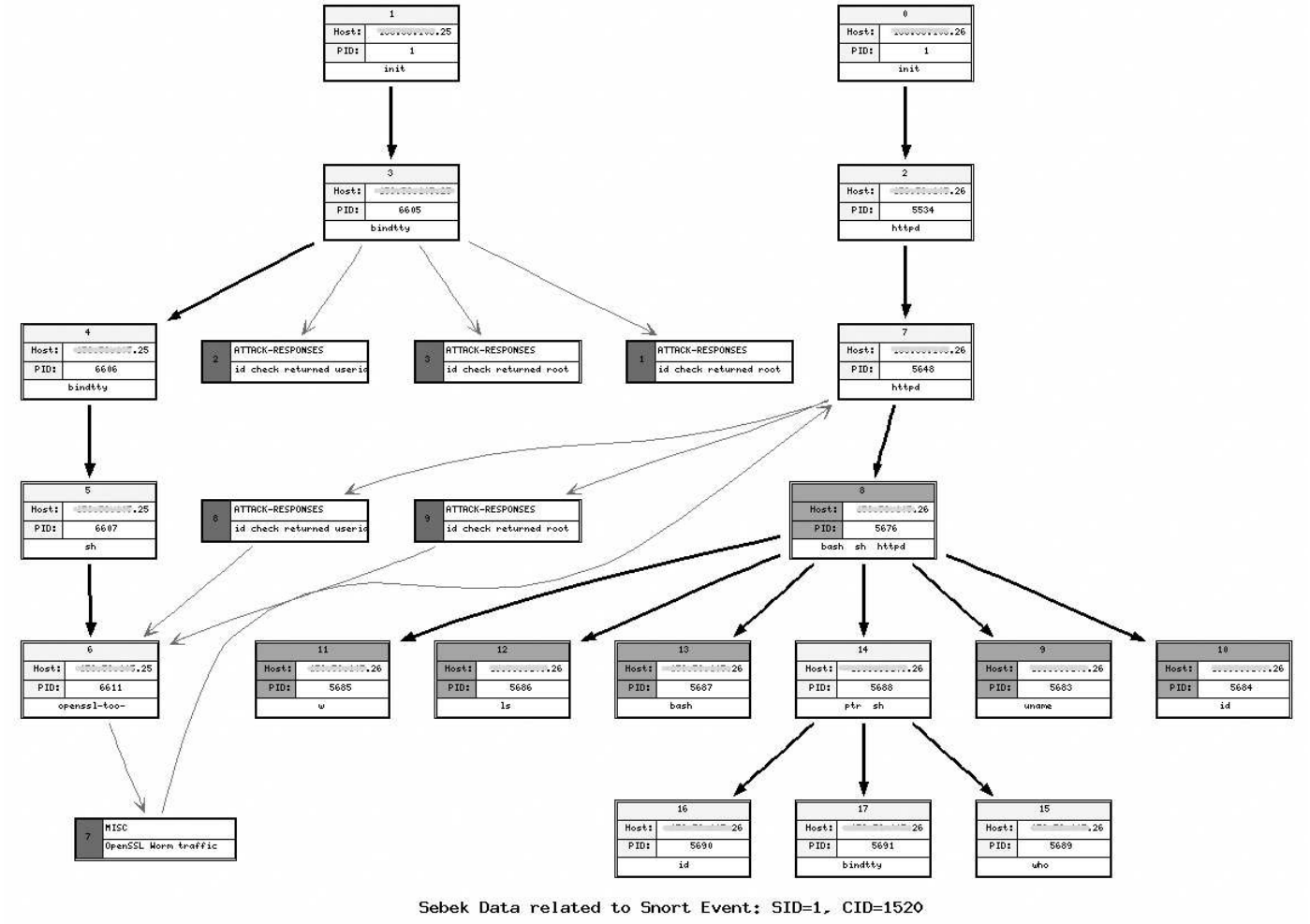

Fig. 5. Example visualization of multi-data source intrusion sequence

The intruder then used the open-too-ssl exploit to attack the *.*.*.26 honeypot. We can see the point where the intruder gained access to *.*.*.26 was from host *.*.*.25 process ID 6811 to host *.*.*.26 process ID 5648 and process ID 5676. Once on the *.*.*.26 host the intruder ran the ptr exploit which took advantage of a ptrace vulnerability to gain root access. Once root access was gained, the bintty backdoor was launched to provide remote access.

This visualization is primarily based on 2 data sources, Sebek and Snort. It shows in a single visualization the sequence of events that occured during the intrusion. From this visualization we can identify the exact point where the intruder gained access to each host, which process was exploited to gain root access and which processes on the system were created by the intruder. While it does not provide the greatest level of detail, we feel that it does provide a composite intrusion sequence view which will improve overall comprehension.

In the Walleye interface this visualization is a clickable image map. The analyst can then click on an element in

the graph to increase the level of detail. For a given process the analyst can click on the icon and would see a summary of system activity for that process, such as a log of all keystrokes.

IV. Limitations

As any with implementation there are some limitations on our apprach we will address them, focusing on the two most important components of our system: Hflow and Sebek.

A. Hflow

The Hflow system is based upon a number of individual components, the data collection tools, the Hflow daemon itself, and the database. Hflow's output is a relational representation of the model and the benefits of this representation come at the price of complexity. Complexity leads to increased failure probability due to the dependency of many interacting parts. To handle the error rate and because not all failures have the same impact, we address the effect of specific component failures.

Table II provides a of component failure impact. In this graph the impact is defined as follows. High impact means that the system has a total loss of ability to observe events. Medium impact means we have lost a significant ability to relate one data source to another, or that we have lost on derived data source. Low impact means we have lost a data source but that the data source was of a supplemental nature. Variable loss means the impact is dependent on the error which is continuous rather than discrete.

TABLE II
Component errors & impact

Comp.	Error	Resulting Loss	Impact
Traf. Rec.	crash	slow path data	High
p0f	crash	OS identification	Low
Snort	crash	Intrusion Detection	Low
Argus	crash	non-IDS flows	Med
Sebekd	crash	All Host Data	Med
		correlation Proc/Flow	Med
Hflow	crash	All fast path data	High
libpcap	pkt loss	accuracy	Variable

Looking back at the Hflow schematic, it is apparent that the Hflow daemon is a single point of failure for the fast path data and the traffic recorder is the single point of failure for the slow path data.

Attack vectors represent scenarios where an intruder can specifically cause a failure in a component. Many of these attacks are essentially Denial of Service type attacks based on resource starvation.

In the case of Hflow, efforts have been made to mitigate the impact of such DoS attacks. Within Hflow there is a buffer of currently active flow data. This buffer is required to account for non-synchronous delivery of data from each of the four data sources. Each time a flow entry is updated a timeout value is updated, as these records go stale they are purged from the buffer based on a defined timeout value. Given that this buffer is inherently limited in size, the most likely attack vector is to fill this buffer.

One approach is to create as many distinct flows in a unit of time, assuming this doesn't crash Argus, this would potentially fill the Hflow flow buffer, to mitigate this risk, Hflow has a parameter that defines a high water mark in the form of number of active flows, when this number is exceeded, Hflow will preferentially drop flows which which do not exhibit a bidirectional flow of data.

As with all of the contributing components, future work will be needed to understand Hflow's operation envelope. The use of connection rate limiting and bandwidth rate limiting provides additional mitigation. Connection limiting or bandwith rate limiting will bound the state creation and thus the buffer needed by Hflow to avoid DoS to can be precisely determined. DoS attacks would then be restricted to libpcap based tools.

B. Sebek

As it relates to Sebek there are two limitations to our approach.

On the current system we cannot observe raw sockets as a result, if an intruder uses a user land IP stack which communicated with the operating system using a raw socket, then we would be unable to correlate the process activity to the network activity.

A second problem we have with Sebek is that we expect the flow of execution to follow the expected process model. For instance if a piece of malware begins to execute within the kernel context we will be unable to monitor the system. Further if a piece of malware is able to jump laterally from one executing process to another[19], our ability to recreate a meaningful process tree have to be reconsidered.

V. Previous Work

A. Previous Models and Data Fusion

Modeling of system behavior by the means of system call monitoring have been studied by Forrest *et al.*[20] and by Provos [21]. These efforts have focused on intrusion detection and/or sandboxing. They provide evidence that system call monitoring can be performed on operational systems which an acceptable level of performance.

Process tree recovery through system call monitoring was first expressed by King *et al.* [22][8] in the CoVirt system. The features added to Sebek for this proposed GenIII architecture are conceptually equivalent to those outlined though the implementation details differ.

The IDS correlation objectives are very similar from ours. As expressed by Hätälä *et al.* the function of cor-

relation is the: (i)Aggregation of related alerts, (ii) Correlation of different information sources and (iii) Context Correlation. In our system we have the three components, but in our case correlation of different information sources and context correlation are the same. The final objective remains clear: we need to provide a better understanding of the states the system went through during an intrusion.

B. Data Analysis

A popular tool for honeynet data analysis is the "Honeynet security Console" [23]. This provides a solid example of the data-centric non-relational honeynet data analysis. The access method is uniform across data types, yet it is lacking the ability relate data from different sources or to create a composite representation of an intrusion event sequence. Although this tool has aided in comprehension, it leaves room for improvement in the area of composite event representation. We anticipate that tools such as this will benefit the ideas outlined in this work.

VI. Future Work/Broader Impact

We feel that the fusion of multiple data sources into a relational representation with a common access method will facilitate a new generation of data analysis techniques. These techniques will benefit from the ability to observe an event in the context of its cause and its impact on the honeynet system, and on the ability to use the relational structure to automatically extract unidentified intrusion sequences from the raw data creating a repository of confirmed intrusions. Within network data, contextual awareness has been used to improve IDS alert reliability and we feel the same type of improvement can be made by providing context awareness which transcends both network, host and process data.

Monitoring data might prove a useful addition to production systems to facilitate incident response. The key value of Sebek is its ability to record volatile system data. This data allows us to identify all network connections file and processes related to an incident, all of which exists temporarily as state information within the operating system. As this information is powerful for improving honeynet data analysis, it should be equally useful for incident response.

For non-research honeynets, this approach should be well suited to increase the level of automation used in honeypots used to generate anti-virus signatures or intrusion signatures. Maybe even create automated incident reports in a open format such as the ones being developed by the IETF-INCH group.

In our current implementation only the Linux version of the Sebek client has been enhanced. we will be working with developers to enhance the remaining version of Sebek to provide he necessary socket and process tracking.

Another direction initially explored by d'Oray *et al.*[24] is the integration of file system monitoring into Sebek. The addition of file system monitoring into Sebek and ultimately into Hflow may provide a link to disk forensics. We feel that with the addition of disk forensics data such as that provided by tools like the Sleuth Kit[25] analysts may be able to use volatile data to optimize the analysis of persistent disk data.

References

[1] T. H. Project, *Know Your Enemy.* Addison-Wesley, 2nd ed., 2004.

[2] N. Provos, "Citi technical report 03-1: A virtual honeypot framework," tech. rep., Center for Information Technology Integration, Univerity of Michigan, 2003.

[3] T. H. Project, "Know your enemy:genii honeynets."

[4] "Netfilter homepage." http://www.netfilter.org, 2005.

[5] M. Roesch, "Snort–lightweight instrusion detection for networks," in *Proceedings of LISA'99 Systems Admistration Conference*, 1999.

[6] T. H. Project, "Know your enemy sebek." http://project.honeynet.org/papers/sebek.pdf, November 2003. Last access: Feb 2005.

[7] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 31, no. 23–24, pp. 2435–2463, 1999.

[8] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen, "Enriching intrusion alerts through multi-host causality," in *Proceedings of the 2005 Network and Distributed System Security Symposium (NDSS)*, February 2005.

[9] T. H. Project, "Know your enemy: Honeywall cdrom." http://project.honeynet.org/papers/cdrom/index.html, Feb 2005. Last access: Feb 2005.

[10] "Argus project," 2004.

[11] "Rtfm: New attributes for traffic flow measurement," 1999.

[12] "Framework for ip performance metrics," 1999.

[13] M. Zalewski, "passive os fingerprinting tool." http://lcamtuf.coredump.cx/p0f.shtml, 2004.

[14] "Real-time network awareness." http://www.sourcefire.com/products/downloads/secured/sf_RNA.pdf, 2004.

[15] R. Gula, "Correlating ids alerts with vulnerability information." http://www.tenablesecurity.com/images/pdfs/va-ids.pdf, 2002.

[16] M, "Cve-2002-0392." http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0392, 2002.

[17] E. Balas, "Honeynet data analysis: A technique for correlating sebek and network data," in *Digital Forensics Research Workshop*, June 2004.

[18] L. Spitzner, "Honeytokens: The other honeypot." http://www.securityfocus.com/infocus/1713, Jul 2003. Last access: Feb 2005.

[19] P. Biondi, "Shellforge g2: Shellcodes for everybody and every platform." http://www.cartel-securite.fr/pbiondi/conf/shellforgeG2_csw04.pdf, 2004.

[20] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of Computer Security*, vol. 6, no. 3, pp. 151–180, 1998.

[21] N. Provos, "Improving host security with system call policies," in *12th USENIX Security Symposium*, USENIX, 2003.

[22] S. T. King and P. M. Chen, "Backtracking instrusions," in *Proceedings of the 2003 Symposium on Operating Systems Principles (SOSP)*, October 2003.

[23] http://www.activeworx.org/programs/hsc/index.htm.

[24] M. d'Orey Posser e Andrade Carbone and P. L. de Geus, "A mechanism for automatic digital evidence collection on high-interaction honeypots," in *Proceedings of the 2004 IEEE Workshop on Information Assurance and Security*, pp. 1–8, IEEE, 2004.

[25] B. Carrier, "The sleuth kit homepage." http://www.sleuthkit.org/sleuthkit/, 2005. Last access: Feb 2005.