Towards a Next Generation of Data Capture Architecture of Honeynets

Yong-Kyung Oh, Inhyuk Kim, and Young Ik Eom
School of info. and Comm. Eng., Sungkyunkwan University
email: kenshin579@skku.ac.kr, {kkojiband, yieom}@ece.skku.ac.kr

Abstract

Honeynets have become one of essential tools in system and network security. As the importance of security has increased over the years, many researchers try to improve the overall Honeynet architecture. Due to their efforts, the Honeynets have evolved up to the third generation. However, the GenIII architecture has some limitations. In this paper, we address some of the limitations and provide solutions by redesigning the framework of data capture of Honeynets.

1. Introduction

Traditionally, when it comes to internet-based attacks, network system administrators who believe there is malicious activity on one of his networks will simply go through each log such as server, firewall, or system to find any problems. It is almost impossible for system administrators to distinguish malicious connections from legitimate activity due to large amount of collected data in the system. They have to go through different security tools manually to detect attacks and further they have to make some changes to prevent those attacks that might be occurred again in the future. Honeynets make all these processes much easier because all those tedious jobs such as collecting and analyzing the data are done by a computer.

A Honeynet is a type of honeypot[8]. Specifically, it is a high interaction honeypot designed to capture extensive information on threats. In a high interaction honeynet, the host systems have real operating systems, applications, and services for attackers to interact with, as opposed to low interaction honeypots such as Honeyd which only provides emulated services and operating systems[1]. Basically, a Honeynet consists of many honeypots which are networked together. Any interaction with Honeynets implies malicious or unauthorized activity has occurred as opposed to interaction with Honeypots. As a result, the Honeynets are well suited for capturing the unauthorized activity. However, Honeynets do collect a hugh amount of detailed data which makes difficult to maintain and extract the useful information.

* This research was supported by the MKE(Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment) (IITA-2008-C1090-0801-0027)

Due to the difficulties of Honeynet deployment and management, the Honeynet Project's Honeywall[6] has been developed. The Honeywall is a linux distribution which contains all of the security tools and functionality necessary to quickly create, easily maintain, and effectively analyze Honeynets data. The current Honeywall uses GenIII architecture which is proposed by Balas and Viecco[2]. Figure 1 shows proposed scheme which provides a central way to gather and combine each of these data sources into a composite relational model. For the data analysis part, they also developed Walleye, a GUI web-based interface, to improve the analyst's capability to quickly perceive the intrusion sequence.

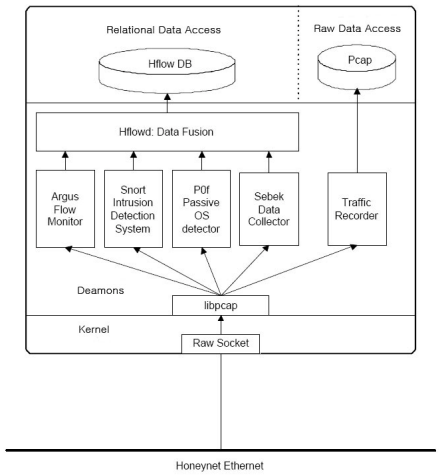

Fig. 1. The GenIII Data Collection Architecture

The GenIII Honeynet architecture has improved a lot over GenII architecture, but it suffers from a number of limitations: First, the GenIII has a single point of failure for both Hflowd and the traffic recorder. There is significant impact on the operation of the Honeynet if Hflowd daemon fails, because it is the core component to collect and combine each data generated from different security tools. The second limitation is that the failure of traffic recorder. Without the traffic recorder, full analysis of network packets would not be available. Lastly, Hflowd daemon does not detect and solve any failures of components. If one of components fails, then analyst would be looking at the flawed process data and may not be able to detect any attacks.

In this paper, solutions to the listed problems in GenIII architecture are presented and the remainder of this paper is organized as follows: Section 2 introduces some related work. Section 3 presents our solutions to the GenIII limitations by introducing new design of our proposed architecture. Finally, we give our conclusions on proposed architecture and discuss future work in the last section.

2. Related Work

The Honeynet project is a collaborative project which has bundles of open source data analysis tools for honeynet/ honeypot data. There are many other related research and tools related to the work presented in this paper.

Honeysnap[5] is another tool from the Honeynet project. It is a modular python application that can parse and perform data analysis on honeynet data. This tool has very minimal dependency on third party executables like tcpflow, and is able to provide analysts with a starting point for more detailed forensic analysis.

There is also a commercial version, HSC (Honeynet Security Console)[4], to analyze honeynet data. The console is only available for Windows 2000/XP platform, and gives the user ability to view events from Snort, TCPDump, Firewall, Syslog, and Sebek logs. It also gives a complete view of the attacker's actions by correlating each of these data types. It has a nice GUI interface and it is free, but no longer maintained by the company.

Like the Argus Flow Monitor, King and Chen[9] developed Backtracker tool to help system administrators to be able to easily analyze intrusion by reconstructing a timeline of events that occur in an attack. The tool can identify files and processes that could have affected, and display the chains of events as a dependency graph like in Figure 2, leading to a quicker identification of the vulnerability.

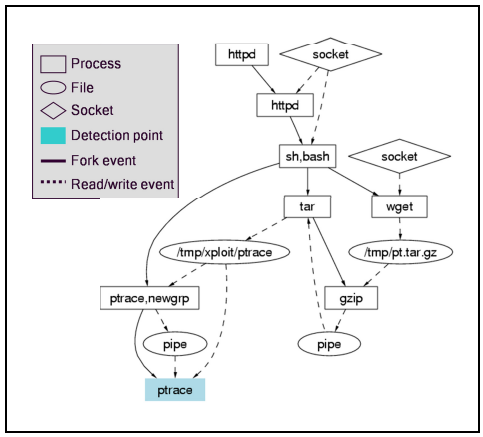

Fig. 2. Backtracker: a Dependency Graph

Typical system loggers do not log sufficient information to recreate or understand all attacks. Due to its lack of completeness and integrity of current system loggers, ReVirt has been proposed by Dunlap[3]. ReVirt logs enough information below the virtual machine which is able to replay the complete, instruction-by-instruction execution of the virtual machine. Using this type of replay, system administrators can easily analyze intrusions.

3. Our Proposed Architecture

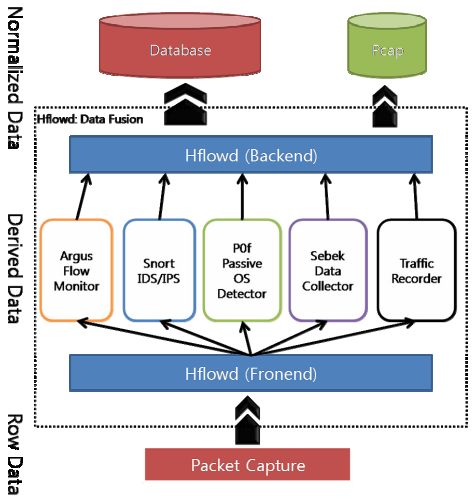

Fig. 3. Next Generation of Data Collection Architecture

Figure 3 provides a revised version of the GenIII architecture. In the diagram, Hflowd is placed below and above the security tool, as opposed to only placed once like in the previous version. In fact, only one Hflowd daemon is running, but this is just to illustrate the fact that new version takes care of both ends, front and back. The Hflowd (frontend) initiates and starts all security components instead of each component starting on its own. Additionally, the frontend checks status of each component to make sure that all the dependent component processes are up and running. It prevents analyzing the inconsistent data. As in the previous architecture, the backend receives Argus flow, Snort IDS, P0f OS fingerprints and Sebek Data, and then these data sources are combined and inserted into a database. The backend also receives traffic flow and stores full packet capture in pcap file. p*cap_api* tool is modulated within Hflowd daemon and it is used to extract raw data dynamically using the database ID corresponding to a flow in the Hflow database as input. The tool extracts the captured data based on time and IP header information.

The following table shows the matching between capture tools and different data categories.

Table 1. Mapping between Capture Tools and Data Categories

Tools	*Flow*	*Host*	*Process*	*File*
Argus	Yes	–	–	–
P0f	Yes	Yes	–	–
Snort	Yes	–	–	–
Sebek	Yes	Yes	Yes	Yes

The GenIII architecture has a single point of failure for both parts, one in traffic recorder and the other in Hflowd. Even looking at our proposed version, it is apparent that Hflowd daemon still has a single point of failure. However, our proposed version has reduced one single point of failure by making traffic recorder part of Hflowd daemon. It seems that it has not made any differences, however, the use of connection counting in the IPTables features[7], we can protect both components from many of attacks essentially attacks like Denial of Service (DoS) type. The connection counting limits the number of outbound connections a honeypot can initiate within a period of time. Once the threshold is reached, the new outbound connections are denied. For different protocols, the connection limits can be set by reconfiguring the rc.firewall script like in Figure 4.

```
SCALE="day"
TCPRATE="15"
UDPRATE="20"
ICMPRATE="50"
OTHERRATE="15"
```

Fig. 5. rc.firewall Script

Here, from a single machine, 15 outgoing TCP connections are allowed per day. If the connection limit is reached, then 15 more connections will be allowed in the next 24 hours.

Unfortunately, the connection counting has a tradeoff between getting more valuable information and hardening the system. If more outbound connections are allowed, identifying a system as the honeypot will be harder, but an attacker has more ways to abuse the honeypot.

We can use Snort-Inline[10] to secure even further, shifting the system from Intrusion Detection System (IDS) to Intrusion Prevention System (IPS). Snort-Inline accepts packets from iptables, via the use of a kernel module named ip_queue and tells iptables to drop or accept the specific packet like in Figure 5.

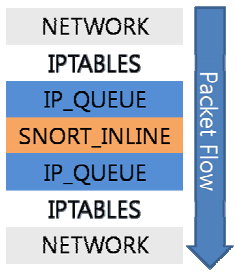

Fig. 5. Packet Flow of Snort_Inline

Snort-Inline is used to detect malicious traffic and take different actions against that traffic. Snort-Inline can either drop or disable known attacks, and it can also modify the contents of the actual attacks, disable the exploit. By using IPTables with Snort-Inline, we can improve the protection of the GenIII architecture.

4. Conclusion and Future Work

In this paper, we pointed out three limitations of Honeynet GenIII architecture and gave solutions by modifying the architecture. First, we added the extra work of Hflowd

daemon at the front guarantees that each component can generate each data and provided good analysis. In addition, with the help of connection limiting feature in IPTable, we protected the single point of failures that existed in GenIII.

In the future work, we plan to implement our proposed version and demonstrate that our solution gives full protection of the attacks that may have existed in the GenIII architecture.

References

[1] Developments of the Honeyd Virtual Honeypot, http://www.honeyd.org
[2] E. Balas and C. Viecco, “Towards a third generation data capture architecture for honeynets,” Proc. of the 6th Information Assurance Workshop, IEEE, 2005.
[3] G. Dunlap, S. King, S. Cinar, M. Basrai, and P. Chen, “ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay,” Proc. of the 2002 Symposium on Operating Systems Design and Implementation (OSDI), 2002.
[4] Honeynet Security Console, http://www.activeworx.org/Programs/HoneynetSecurityConsole/tabid/61/Default.aspx
[5] Honeysnap, https://projects.honeynet.org/honeysnap
[6] Honeywall, https://projects.honeynet.org/honeywall/wiki
[7] Know Your Enemy: Honeywall CDROM Eeyore, http://www.honeynet.org/papers/cdrom/eeyore
[8] N. Provos. “A Virtual Honeypot Framework,” Proc. of the 13th USENIX Security Symposium, 2004.
[9] S. King and P. Chen, "Backtracking Intrusions", Proc. of the 2003 Symposium on Operating Systems Principles (SOSP), 2003.
[10] Snort-inline, http://snort-inline.sourceforge.net