Conceptual framework for a Honeypot solution

Christian Döring, M.Sc.
University of Applied Sciences Darmstadt, Department of Informatics (FHD)
cdoering@web.de

Co-author: Prof. Dr. Heinz-Erich Erbs
University of Applied Sciences Darmstadt, Department of Informatics (FHD)
erbs@fbi.fh-darmstadt.de

Abstract. Several Honeypot solutions have been developed since Clifford Stoll described the first use of a computer to trace an intruder [1]. But there is no common framework of deploying Honeypots and especially no common analysis method exists. This causes Honeypot-unfamiliar operators to spend a great amount of time with learning concepts of Honeypots and even more time with interpreting results. This paper explains what a non-experimental Honeypot solution requires and how a ready Honeypot can be used to identify unknown attacks.

1 Introduction

The concept of Honeypots in general is to catch malicious network activity with a prepared machine. This computer is used as decoy. The intruder is intended to detect the Honeypot and try to break into it. The purpose of a Honeypot is to detect and learn from attacks and use that information to improve security. A network administrator obtains first-hand information about the current threats on his network. Undiscovered security holes can be protected gained by the information from a Honeypot.

So far network monitoring techniques use passive devices, such as Intrusion Detection Systems (IDS). IDS analyze network traffic for malicious connections based on patterns. Those can be particular words in packet payloads or specific sequences of packets. However there is the possibility of false positive alerts, due to a pattern mismatch or even worse, false negative alerts on actual attacks. On a Honeypot every packet is suspicious. The reason for this is that in a Honeypot scenario, the Honeypot is not registered to any production system. Regular production systems should not be aware of the presence of a Honeypot. Also the Honeypot should not provide any real production data. This ensures that the Honeypot is not connected by trustworthy devices. Therefore any device establishing a connection to a Honeypot is either wrong configured or source of an attack. This makes it easy to detect attacks on Honeypots.

The value of a Honeypot is determined by its purpose. To increase the benefit of a solution its goal needs to be specified as detailed as possible. This paper introduces a conceptual framework which was developed in [4]. A setup following these

guidelines and implemented with proper definition on purpose and intended usage will help in saving time on analyzing the results. That approach prevents from missing valuable results within the captured data because of having no analysis strategy. In chapter 3 one strategy is described to identify unknown attacks.

2 Required features

The following section describes features required to every Honeypot solution. To form a conceptual framework for any deployment the paragraphs explain what should be implemented and not how. An example of how to implement a solution can be found in the next chapter. The following section is an extraction from [4].

Security

A running Honeypot may not, in any circumstances, touch production machines. This is the highest goal of securing a Honeypot. Otherwise a compromised Honeypot would allow taking over sensible data and machines. Actually a compromised Honeypot should not be able to touch any other machine than the one which infected it, but this would decrease the Honeypot's value to attackers. In some cases this would be sufficient, but Honeypot's are dedicated to watch and learn new threats and this requires outbound connections.

However a production Honeypot with a low level of interaction can be configured with no outbound access. The risk involved in this setup can be neglected. Thus it appears that different setups involve different risks. An implementer has to carefully consider this when planning a configuration.

A good practice is to deny the Honeypot access to any production device and open communication paths to the public network. Additionally the access to the public network needs to be limited. Without this limitation the Honeypot could be used to execute, i.e. a Denial-of-Service or a Distributed-Denial-of-Service-attack or to store stolen software. This requirement can be reached by limiting the number and size of connections to the outside. With more advanced techniques, such as Intrusion Protection System it is possible to filter individual flows matching predefined patterns. This would ban the propagation of worms and separate from advanced attacks.

Cloaking the Honeypot

The ideal solution would be to tap the monitoring device to a hub between Honeypot and the network and capture all traffic. This would hide the presence of the monitor or Honeywall. In case of denying outbound traffic from the Honeypot this would be a good solution.

But this would allow passive monitoring only and lack of controlling. Lance Spitzner the founder of the Honeynet Project mentions in [2] the importance of a controlling mechanism, hence this type of control needs to read every packet, decide

if permitted and either drop or forward it. A firewall seems to be an adequate solution for this case.

Firewalls typically work on OSI layer 3. During the transmit process the IP header is rewritten: the Time-to-live field is reset, the MAC address is changed and the header checksum is re-calculated. An advanced intruder could reveal those changes and fingerprint the Honeywall, which would make the Honeypot fairly uninteresting or even worse the attacker would attack the Honeywall.

Analyzability

The collected data on a Honeypot shows what happened on the wire: scans, intrusion attempts, worm propagation and other malicious activities. After dumping the packets into an analyze tool the investigator is confronted with an enormous amount of data. A method is needed to weed out the informative data from the useless traffic.

TCP connections are easy to track. They provide a sequence number with identifies each packet to a corresponding flow. Packets need to be gathered to flows to reduce the amount of items to be analyzed. This includes bi-directional traffic to and from the target. The challenge is to categorize each flow. It is easy to assign a purpose to a flow by checking the destination port. In most cases this is satisfactory, but in some circumstances a flow to a specified port does not contain valid data or it might be a port scan. Therefore categorizing a flow by its port number is not always valid. Intrusion Detection Systems (IDS) help identifying further.

UDP connections are stateless and do not provide an extra options to relate them to an individual flow. The IDS Snort by Martin Roesch [3], defines a flow as unique when the IP protocol, source IP, source port, destination IP, and destination port are the same.

In the Internet Protocol version 4 (IPv4) there is an 8 bit field called "Protocol", to identify the next level protocol. It is difficult to analyze protocols which are neither TCP nor UDP as most analysis tools, i.e. Snort is based on these protocols.

Accessibility

Well-grounded setups cover security and promote easy and complete ways to analyze data. Further an operator needs quick access to this data. Also a way of notifying of events needs to be implemented.

In order to check data and logs frequently, the operator needs physical or network access. Direct access to the console is provided by any installation, in addition tools need to be installed which provides quick analysis of current data. Without the chance of direct access, i.e. in a hosted environment, the monitoring device should provide an interface for accessing the data. Problem is that access to the monitor causes extra traffic which could lead to reveal the Honeypot's existence. Hence the analysis interface must be accessed over another path. In addition to this the connection should be encrypted that even when discovered, its true meaning must not be exposed.

Alerting

Quick response to attacks requires automatic notification of the operator. An automatic alerting function should be able to send messages when an intrusion was detected. Also it should be possible to send alerts in various ways. In the case an alerting message fails to deliver, a redundant destination path should be available.

The easiest trigger for alerts can be found in the nature of Honeypots. By its nature every connection made to the Honeypot is suspicious and would not occur without the presence a Honeypot. However traffic which is not responded by the Honeypot is not interesting, therefore outbound data should be used to trigger alerts. This includes that any service which sends outbound traffic without user/ hacker interaction, i.e. Browser service on Windows machines, needs to be stopped.

But the outbound trigger can also overwhelm mailboxes. An experiment performed on June 12, 2005, triggered an average of 7 flows per minute [4] the mailbox was soon flooded with mails. This shows that alerting mechanisms need to be adjusted according to the demands of the environment.

Traffic which exceeds outbound limits can also be used as a trigger. This would add more level of detail to the alert. Also protocols other than TCP or UDP should trigger an alert. The protocol value in the IP header provides this, i.e. TCP has the value 6(decimal) and UDP 17(decimal). Values other than those are symptoms of unknown attacks and could be used to bypass firewall rules.

In case of accepted outbound traffic the alert mechanism needs to be trained with patterns of valid traffic. But the other way round when an attacker has found a new way to exploit vulnerabilities which are not recognized, an important alert would be missing. A solution which focuses on detecting patterns only would not be adequate.

3 Using a Honeypot to identify unknown attacks

In this example the Honeypot solution Roo of the Honeynet Project [5] is used. It fulfills all the above requirements and offers a comprehensive platform for identifying unknown attacks. Roo is using one computer for monitoring network traffic, the Honeywall, and one or many computers as decoy. The decoy machines are connected to a dedicated network, the Honeynet, which is connected to a production network by the Honeywall. A paper describing that architecture can be found on [6].

Before implementing a Honeypot an operator needs to gather configuration variables and record them. Changing variables, i.e. new IP from an ISP and the time of change must be captured in a log. This is necessary to allow a setup to be repeated and later analysis to compare with.

A Honeypot which is attacked and compromised by an intruder proposes risk to other network devices. These can be located in part of the own production network or devices on the internet. Allowing an attacker to compromise own production machines might lead to loose or expose critical data. Attacks launched from a Honeypot might direct to legal issues, i.e. a hacker starting a denial of service attack from the Honeypot. To minimize that danger test cases must be created and executed prior to connecting the Honeypot to publicly accessible networks. The test cases must

cover all critical aspects and ensure that the Honeypot does not propose more than the identified risk.

Known attacks versus unknown attacks

Roo uses several components to identify suspicious connections. Most important is Snort. It identifies attacks by pattern analysis. Snort patterns are defined by rules which can be payload, non-payload or post-detections rules. Payload rules scan the content of packets, while non-payload rules concentrate on protocol headers. Post-detection rules are used to perform additional activity after detection.

Every connection made to the Honeypot is listed as a flow with information on protocol, IP addresses, ports, amount of packets etc. A graphical user interface (GUI) shows each flow and offers to download a tcpdump[1] compatible file.

Known attacks are identified by Snort rules and if occurred listed in the flow. Unknown attacks are either identified by a generic alert, i.e. "SHELLCODE x86 inc ebx NOOP", by the notice "unknown signature" or without any alert notice. The shellcode alert in the previous example is not counted as a known attack because it does not apply to a specific vulnerability. The mentioned shellcode is a technique called "Noop-sliding" used in several exploits to gain access (see [4] on pages 43-60 for an in-depth analysis).

Verifying new attacks

The problem with verifying success of an attack is that the Honeywall does not monitor processes on the decoy. Without information what each connection caused it is not possible to state if an attack was successful. Analyzing the responses from the decoy is not sufficient. The attacked machine could send responses which look like the attack was denied but this might deviate from the truth. The only way to determine if an attack was successful is to replay the attack.

For replaying an attack a second Honeypot is needed which matches the requirements and variables of the original. To provide this the operating system is re-installed with the information of the configuration sheet prepared before the first setup. This trial machine is then installed with monitoring tools and a debugger. Sysinternals [7] provides excellent monitoring tools for Microsoft Windows platforms. On a Windows machine it is required to monitor registry activity, file access, process and port usage. The debugger reveals what function calls are made and which external libraries are accessed.

Most important is the creation of processes. When the connection is even able to create a new connection it was successful. The new connection can be used by an attacker to download a binary which provides future access to the infected machine.

With the above described verification it is possible to write a new Snort rule to identify the new attack. Further action should be taken to close the vulnerability on

[1] Tcpdump is a UNIX maintenance command. It prints out the headers of packets on a network interface.

production machines. The trial Honeypot can also be used to verify if a bugfix or workaround provides enough security to close the vulnerability.

Conclusion

Using a conceptual framework for utilizing and analyzing Honeypots improves analysis time and value of results. It is also basis for comparing results. The above described example benefits of repeatability to verify an attack. Without the exact information of setup and variables it would not have been possible to set up a trial Honeypot.

Process creation in correspondence with traffic reveals success of an attack. In the example of this document this is done manually. Hence it would improve analysis time if this could be done automatically. Future development should concentrate on providing process history. The Honeynet Project has developed a solution, Sebek [8], which monitors console activity. It would be worthwhile to implement process monitoring to this application and benefit from the yet existing features.

References

1 Clifford Stoll: The Coocoo's egg, Pocket Books 1990
2 Lance Spitzner: Honeypots, Tracking Hackers, pages 239-240, Addison-Wesley 2002
3 Martin Roesch: Martin Roesch, Snort – Intrusion Detection and Prevention System, http://www.snort.org/, Sourcefire Inc.
4 Christian Döring: Improving network security with Honeypots, pages B34-B36, Master's thesis University of Applied Sciences Darmstadt, Department of Informatics, 2005
5 The Honeynet Project: Research alliance, http://www.honeynet.org, non-profit Honeypot research organization, 1999
6 Honeynet Project: Know your enemy – Learning about security threats (2nd Edition), GenII Honeynets, pages 95-180, Addison-Wesley 2004
7 Mark Russinovich, Bryce Cogswell: Sysinternals, freeware for Windows internals, http://www.sysinternals.com
8 Honeynet Project: Know your enemy - Sebek: A Kernel based capture tool, web publication, http://www.honeynet.org/papers/sebek.pdf, Honeynet Project 2003