Implementing and testing an Intrusion Detection Honeypot

Jonathan Werrett
School of Computer Science & Software Engineering
University of Western Australia
35 Stirling Highway, Crawley, WA, 6009
werrej01@csse.uwa.edu.au

2nd June 2003

Abstract

In this paper a Honeypot is implemented and tested in a controlled environment. Eventually it is hoped the Honeypot configuration that has come out of this paper will be used in a research type situation on a large, operating computer network. For this reason it was important to thoroughly test the Honeypot. The configuration of the various component software was explored and recorded. Tests aimed at replicating what the Honeypot would experience in a production environment were proposed and run. Valuable data was collected on the computer resource requirements, software stability, appearance and the behaviour of the Honeypot when faced with probes and attacks over a network.

Keywords: honeypots, intrusion detection, computer security

Introduction

Honeypots are a relatively recent development in the field of Intrusion Detection. Intrusion Detection is concerned with the detection of attacks and other anomalous uses of computers and computer networks. The main concern of Intrusion Detection is being able to identify and separate "anomalous" computer events (such as attacks) from "normal" events. Intrusion Detection Systems (IDS) can be classified into two categories: Network-based Intrusion Detection Systems (NIDS) and and Host-based Intrusion Detection Systems (HIDS).

A Honeypot is defined as being a "security resource whose value lies in being probed or compromised" [1]. They can be either host and/or network based, but are more often than not network based as all interaction is typically performed over a network

connection. A honeypot's main utility comes from the fact that it simplifys the Intrusion Detection problem of separating "anomalous" from "normal" by having no legitimate purpose, thus any activity on a Honeypot can be immediately defined as anomalous. The characteristics of Honeypots make them well suited to the monitoring of malcious activity on networks, as well as being a valuable research tool in Computer Security.

Honeypot concepts are not particularly new, they can be traced back to early Computer Security papers such as Clifford Stoll's *Stalking The Wily Hacker* [2], however the study of Honeypots has recently been formalised. With this formalisation has come a focus on research of attackers' methods and motivations [3], which has led to Honeypots being instrumental in the discovery of new security vulnerabilities [4].

I intend to implement and test a Honeypot in a "test-bed" type situation, on a small Local Area Network (LAN). This implementation will be done for the purpose of gaining familiarity and experience with running a Honeypot, as well as other related technologies, in a controlled environment. This will be done as a precursor to installing a Honeypot in a production environment, namely the Computer Science Department's network at the University of Western Australia (UWA). By initially implementing and testing a Honeypot in a controlled environment I will be able to evaluate my particular Honeypot solution and gain a clearer picture of it's suitability to the eventual production environment. This is essential in minimising any potential disruption to the department's network.

As well as gaining a clearer picture of my Honeypot's suitability to the production environment I will be able to test the Honeypot by running a number of attack tools against it. This testing will provide valuable data on how the Honeypot performs when attacked, including what it logs and what types of attacks it can and can't detect. The testing will also give an indication of what computer resources the Honeypot will need in the production environment (for example, the amount of disk space needed for different levels of logging and what speed CPU is required). On top of these benefits running the Honeypot in a controlled environment will help to iron out any initial implementation problems with related technologies that are needed (such as firewalls and packet loggers). By running these tests against the Honeypot initial data will also be generated that can be used later in my Honours project.

There are number of different types of Honeypots that can be used, ranging from slightly modified "production" machines to commercial software solutions such as ManTrap [5]. The different types of Honeypots have different advantages and disadvantages and are typically suited to different situations. For example ManTrap is a Solaris based, high interaction (meaning an attacker has a "full" operating system to interact with) and logs on a host and network level, this makes it well suited to both Intrusion Detection research and as a production IDS [3].

For this Honeypot implementation I have chosen the Honeyd software Honeypot [6]. It is a low-interaction Honeypot, providing only limited network service emulation (for example, SMTP), however this is acceptable as we are more interested in detecting probes than tracking attackers after a compromise. Honeyd has the unique

feature of being able to emulate multiple operating systems at the network level, using a database of common operating systems' TCP/IP "fingerprints". Honeyd is also easily integrated with running production networks as it can use ARP spoofing to take control of unused IP addresses within a network.

To test the Honeypot I will run a series of common automated attack tools from one machine (the "attack" machine) against a network. The network will have a small number of real machines and I will emulate a number of machines of different types using ARP spoofing. By reviewing the resulting logs I will be able to gain a good picture on how this specific Honeypot setup will perform in a production environment, including what attacks it detects, how it classifies various types of attacks, what attacks (if any) it is unable to detect and how the system as a whole performs. This will be by no means an exhaustive test of the Honeypot's detection capability, but will give a good indication of what to expect when running a Honeypot in a production environment.

Method

I implemented the Honeypot on a small private Ethernet-based LAN, that was disconnected from the Internet. Initially the network consisted of three separate computers, two running Linux and a third running FreeBSD, connected via a hub (see Figure). I made one of the Linux machines the "Honeypot" by installing Honeyd on it and assigned the job of "Attacker" to the other Linux machine.

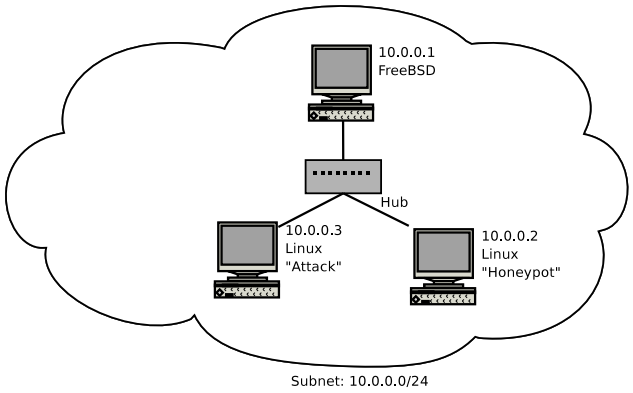

Figure 1: The physical structure of the LAN.

In conjunction with Honeyd, various other pieces of software must be used. Arpd [7] must be used to take control of the unused IP addresses on a network through ARP spoofing. A firewall (in this case Iptables [8]) is typically used to restrict access to the physical Honeypot machine running Honeyd. Also a packet logger (such as Snort [9]) is also used in combination with Honeyd, as Honeyd only logs rudimentary data upon a probe to one of the emulated machines (for example, machine address, port and protocol type). Therefore for more detailed data logging, including identification of probes a packet logger must be used. Honeyd also fails to log probes utilising protocols other than TCP, UDP or ICMP, having a separate

packet logger overcomes this. Arpd was configured to spoof the unused IP addresses for the entire subnet (10.0.0.0/24). The firewall was configured to only allow through traffic to specific machines (either the actual physical machines or the ones emulated by the Honeypot) and reject any other traffic. This firewall configuration sped up the attacks launched, some of which took lengthy times to execute, by quickly rejecting any probes to non-existent machines. Snort used a very similar configuration to that recommended by the Honeynet Project for Honeypots [10]. For further details on the software used and their parameters see Appendix A.

Honeyd was configured to emulate a Cisco router, 2 Windows 2000 machines and 4 Linux machines. These were arranged in a flat network topology. The Honeyd configure file for this setup is listed in Appendix B. Using this configuration Honeyd, in conjunction with the previously listed software, changes the logical structure of our test subnet to that shown in Figure , while the physical structure remained the same as in Figure . It is this new logical structure that would be "seen" by an attacker while probing our network.

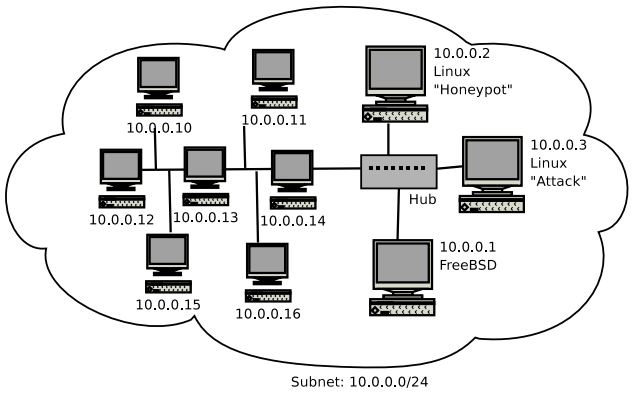

Figure 2: The logical structure of the LAN.

The first probe launched against our test subnet from our Attack machine was a port-scan using Nmap [11]. While technically not being an "attack" it is a common probe [12] employed by attackers to gather information on a target network's structure and to get some idea on what services, and hence software, is being employed. After running a port-scan a list of machines on a target network is gained, along with the associated services and possible the Operating System types. The next step an attacker typically makes is to launch various attacks against the specific services. For this reason being able to detect port-scans is an important feature. Nmap has numerous settings with which scans can be tweaked with the aim of avoiding detection. For the initial run the default stealth SYN scan was used in combination with OS TCP/IP fingerprinting.

The second probe launched from the Attack machine used the Xprobe2 [13] OS fingerprinting tool. Once again this is not an attack as such but still falls under the category of malcious activity. The Xprobe2 tool, like Nmap, falls within the top 75 most popular security tools as listed by Insecure.org's most recent survey [12], and thus it is important to be able to detect and recognise easily.

The next round of tests against the Honeyd intrusion detection system was a com-

plete security audit using the Nessus Security Vulnerability scanner [14]. This is more a tool used by Administrator's auditing their own networks for security holes than attackers as it takes an unsubtle approach and is bound to set off many security alarms/alerts. Nessus tests for various security holes based on what services are open on a target host and is thus a useful tool for testing Snort's pattern-based intrusion detection.

For the fourth round of probes I went back to Nmap and explored some of the more advanced port-scan features that were aimed at avoiding IDS systems. First I tried using the advanced timing options leaving a 5 minute gap (using the -T0 option) between separate probes to ports, still using stealth SYN scans however. I then tried the more advanced scan types including FIN (using the -sF option), Xmas (-sX), NULL (-sN), and Window (-sW) scans. See the Nmap documentation for the specifics of these types of scans [15]. As well as using the different types of scans Nmap was also run in its fragmentation mode. In this mode probe packets are fragmented to the minimum possible size in an attempt to avoid detection.

Results

The default stealth SYN port-scan performed by Nmap was detected well by our Honeypot setup. The Honeyd log files clearly show a port-scan "sweeping" across our network looking for open services and the Snort log files correctly label the original ICMP ping to the various hosts as "ICMP PING NMAP" and detect a stealth scan occurring. Sometimes, however, Snort misidentifies the scan type as either Xmas or NULL.

Nmap was successful in identifying the OS of the emulated machines. Xprobe2 on the other hand correctly identified the Windows machines that were being emulated by the Honeypot, but incorrectly guessed the kernel version of the Linux machines. Xprobe2 returned the Linux 2.2 series kernel as being the most probable, while it was actually meant to be emulating the behaviour of the 2.4.0 and greater kernels. As an aside Xprobe2 left much less of a footprint in the Honeypot logs, than the Nmap attempts to fingerprint. Snort and Honeyd correctly identified the Nmap probes, while the Xprobe2 probes simply showed up as unusual ICMP packets.

The scan performed by Nessus resulted in a large audit trail, as predicted. The Honeyd logs clearly show the preceding port-scans that were performed and the following vulnerability probes. While Snort does not actually detect a Nessus scan itself occurring it does correctly label the various vulnerability probes (for example the Denial of Service attacks).

Changing the timing of the Nmap port-scans in the fourth round of probing did little to "throw off" our Honeypot. The Honeyd logs clearly show the scan sequentially connecting to each port and various hosts. If anything the extended timing made the resulting traffic appear more suspicious as there was exactly five minutes left between new connections. The specific scan types also did little to avoid detection, with Snort correctly identifying them all (once again however with some "false-

Scan Type	Priority 1	Priority 2	Priority 3	Total
Nmap SYN	0	518	0	518
Nmap Fingerprint	0	13	4	17
Xprobe2	0	0	20	20
Nessus	41	6580	433	7054
Nmap FIN	0	518	0	518
Nmap Xmas	0	517	0	517
Nmap NULL	0	521	0	521
Nmap Fragment [1]	0	0	4	4

Table 1: Alerts in each priority level generated by Snort during testing.

positives"). The fragmented packets were also correctly identified, however they also caused Snort to crash. Refer to the Discussion section for further details.

Overall the Honeypot implementation did well at detecting, labeling and prioritising probes launched against it in the testing phase. Snort using it's own probe priority scheme (each detected probe is given a priority from 1 to 3) did a reasonable job at separating probes out based on their potential severity. Table 1 lists the various probes that were launched against the Honeypot and the number of resulting alerts in each priority level. As is expected the Nessus scan tops the table in all three levels and Xprobe2 was the least intrusive. Surprisingly the Nmap scans that were aimed at avoiding IDS's (for example FIN and Xmas) set off the most alerts.

Discussion

The first benefit to come out of the testing process was the detection of a subtle software bug in Honeyd and one of it's network libraries. This software bug caused the Honeyd process on the Honeypot machine to crash when certain types of aggressive probes (for example, Nmap SYN scans) were launch against the entire subnet. This problem would have gone unnoticed without extensive testing of the Honeypot prior to implementing it in a production environment. Thankfully this bug was restricted to the Honeyd version that was included in the Linux distribution being used and was fixed by compiling the very latest version of Honeyd from its website [6].

Early testing of the Honeypot also lead to some tweaks to the Honeyd configuration. Originally there was no default behaviour specific for the unused addresses that did not have an emulated machine attached to them. In this situation Honeyd caused all ports to respond as being open and accepting connections. This lead to significant slow downs in attack probes and a network displaying this strange behaviour would be very undesirable in a production environment.

Other more minor tweaks to the Honeyd configuration that were highlighted in the testing process included a need for careful crafting of the emulated machine uptimes. The Nmap scans when performing OS fingerprinting also report the uptime

[1] Nmap Fragment scan not completely logged as it caused Snort to crash.

Scan Type	Connections	Log Size (Kilobytes)
Nmap SYN	11560	2224
Nmap Fingerprint	3306	468
Xprobe2	15	64
Nessus	10002758	446032
Nmap FIN	11592	6664
Nmap Xmas	11429	6663
Nmap NULL	11541	6524
Nmap Fragment [2]	11315	804

Table 2: Logs generated compared to the connections made to the Honeypot.

of the probed hosts. Having a large number of hosts with identical OS fingerprints and identical up-times would be suspicious to any potential attackers. Also the initial fingerprint that was chosen for the emulated Windows machines ("Windows 2000 SP3") was not being correctly identified in the Nmap scans. A fingerprint that was able to be identified was important as we want to provide targets for would be attackers that are as desirable as possible.

The results of testing the Honeypot with the Nmap and Xprobe2 tools were close to what was expected. This, however, is somewhat unsurprising as Honeyd is specifically aimed at fooling these two attack tools, and even utilises some of their configuration files as fingerprint databases. While they are certainly two of the more common attack tools, testing with other less common tools would ensure that the Honeypot presents a realistic looking facade to any potential attackers.

While testing the Honeypot implementation a serious bug was found with the version of Snort used (2.0.0). It was discovered that Nmap, when fragmenting its probe packets into the smallest size possible, will crash Snort. A thorough search of the Web and the Snort mailing lists did not turn up any reference to this problem. Snort is an Open Source application (as are most of the other tools used to build the Honeypot) and a bug was filed against it to its development team on the 21st May [16]. After some correspondance the development team was able to track the bug down and fix it in the current development version of Snort, the next official release will thus have this problem fixed. In a production environment this would represent an unacceptable weakness and hopefully the new version is available before I deploy the Honeypot on the Department's network.

Testing common probes against the Honeypot gave us a valuable insight into the resource requirements of the Honeypot. The computer the Honeypot was installed on (a Pentium III 800MHz, with 512M of RAM) was more then capable of running all the required software, with it at no stage using more than 10 percent of the systems resources. Table 2 demonstrates the rising data storage requirements of the Honeypot as the number of connections made increases. This data, in combination with sample data on how many connections will be made in any time-frame in the production environment, will help us estimate how much hard-drive space will be

[2]Nmap Fragment scan not completely logged as it caused Snort to crash.

required for the eventual deployment.

Conclusion

Implementing the Honeypot in a simple "test-bed" style network enabled me to test the Honeypot in a controlled fashion. By running common attacks against the Honeypot I was able to gain an idea as to how it would appear to any potential attackers. This gave me valuable feedback on how to improve the configuration so as to make the Honeypot appear and behave in a more realistic manner. Not only that but I was able to correspond the attacks to the traces that they left on the Honeypot, giving me data on what attacks will look like when the Honeypot is deployed in the field.

The Intrusion Detection capabilities of the system, where well demonstrated. Most of the time the IDS was able to correctly identify attacks that where being launched against it, and even sometimes what tools where being used to launch these attacks. This was not exhaustive however, and I was restricted to using well known attacks to test the system. It is unknown how it will perform in detecting new or novel attacks.

Also during testing of the Honeypot two software bugs were discovered that would potentially of caused serious problems with data collection if the Honeypot was being deployed in a production environment. One bug was able to be worked around by downloading the latest version of the Honeyd software. The other bug, the one in Snort, was more novel and was the first time I could find reference to it being encountered. I was able to report this bug to the maintainers and the next version of the Snort software released should fix it.

Finally during testing I demonstrated that the computer resources I was using in testing the Honeypot were more than sufficient to run the required software. I was also able to gain a good idea of how much data is generated by the Honeypot and hence how much data storage I will need when implementing the Honeypot in a production environment and the amount of data I will need to analyse.

References

[1] Lance Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley, Boston. 2002.

[2] Clifford Stoll. Stalking the Wily Hacker. *Communications of the ACM*. pp 484-497. 1988.

[3] Honeynet Research Alliance. Project Honeynet Website. Retrieved May 16th 2003 from the World Wide Web: http://project.honey.org

[4] Computer Emergency Response Team. dtscpd Exploit Advisory. *Advisory CA-2002-01 Exploitation of Vulnerability in CDE Subprocess*

Control Service. Retrieved May 16th 2003 from the World Wide http://www.cert.org/advisories/CA-2002-01.html

[5] Recourse Technologies. ManTrap. Retrieved May 16th 2003 from the World Wide Web: http://www.recourse.com

[6] Niels Provos. Honeyd. Retrieved May 16th 2003 from the World Wide http://www.citi.umich.edu/u/provos/honeyd/

[7] Dug Song and Niels Provos. Arpd. Retrieved May 16th 2003 from the World Wide http://www.citi.umich.edu/u/provos/honeyd/

[8] Netfilter Core Team. ptables. Retrieved May 16th 2003 from the World Wide Web: http://www.netfilter.org

[9] Martin Roesch. Snort. Retrieved May 17th 2003 from the World Wide Web: http://www.snort.org

[10] Honeynet Research Alliance. Honeynet Project Snort configuration file. Retrieved May 17th 2003 from the World Wide Web: http://www.honeynet.org/papers/honeynet/tools/snort.conf.

[11] Insecure.com LLC. Nmap. Retrieved May 19th 2003 from the World Wide Web: http://www.insecure.org

[12] Insecure.com LLC. Top 75 network security tools. Retrieved May 19th 2003 from the World Wide Web: http://www.insecure.org/tools.html

[13] Fyodor Yarochkin and Ofir Arkin. Xprobe2. Retrieved May 19th 2003 from the World Wide Web: http://www.sys-security.com/

[14] Renaud Deraison. Nessus. Retrieved May 19th 2003 from the World Wide Web: http://www.nessus.org

[15] Insecure.com LLC. Nmap documentation. Retrieved May 20th 2003 from the World Wide Web: http://www.insecure.org/nmap/nmap_documentation.html

[16] Martin Roesch. Snort bug #741138. Retrieved May 21st 2003 from the World Wide Web: https://sourceforge.net/tracker/?func=detail&atid=103357&aid=741138&group_id=3357

Appendices

Appendix A

```
# Start Aprd spoofing any unused IP's in our subnet.
arpd 10.0.0.0/24

# Start Snort logging any traffic seen.
# snort.conf provided by Project Honeynet.
```

```
# 10.0.0.2 is the Honeypot.
snort -D -c snort.conf -l logs/  not src host 10.0.0.2

# Start Honeyd logging and emulating machines.
# honey.conf listed in Appendix B
# nmap.print, xprobe2.prints nmap.assoc all provided in the Honeyd package.
honeyd -f honey.conf -p nmap.prints -x xprobe2.prints -a nmap.assoc
-l logs/honeyd 10.0.0.10-10.0.0.16
```

Appendix B

```
### Default Honeyd behaviour
create default
set default default tcp action reset
set default default udp action reset

### Windows computers behaviour
create windows
set windows personality "Windows 2000/XP/ME"
set windows default tcp action reset
set windows default udp action reset
\# Emulated IIS 5.0.
add windows tcp port 80 "/usr/bin/perl /usr/share/honeyd/scripts/iis-0.95/iisemul8.pl"
add windows tcp port 139 open
add windows tcp port 137 open
add windows udp port 137 open
add windows udp port 135 open
bind 10.0.0.10 windows
set 10.0.0.10 uptime 16120000
bind 10.0.0.14 windows
set 10.0.0.14 uptime 11690000


### Linux 2.4.x computer beahviour
create linux
set linux personality "Linux Kernel 2.4.0 - 2.5.20"
set linux default tcp action reset
set linux default udp action reset
# Emulate QPOP 2.53 daemon.
add linux tcp port 110 "/bin/sh /usr/share/honeyd/scripts/pop3.sh"
# Emulate Sendmail 8.12.2 daemon.
add linux tcp port 25 "/bin/sh /usr/share/honeyd/scripts/sendmail.sh"
# Emulate wu-ftp 2.6.0(5) daemon.
add linux tcp port 21 "/bin/sh /usr/share/honeyd/scripts/ftp.sh"
bind 10.0.0.13 linux
set 10.0.0.13 uptime 55596000
bind 10.0.0.15 linux
set 10.0.0.15 uptime 56223000
bind 10.0.0.11 linux
set 10.0.0.11 uptime 55989000
bind 10.0.0.16 linux
set 10.0.0.16 uptime 23082000

### Cisco router behaviour
create router
set router personality "Cisco IOS 11.3 - 12.0(11)"
```

```
set router default tcp action reset
set router default udp action reset
# Emulated simple telnet daemon, with Cisco banner.
add router tcp port 23 "/usr/bin/perl /usr/share/honeyd/scripts/router-telnet.pl"
set router uid 32767 gid 32767
set router uptime 327650000
bind 10.0.0.12 router
```