Design Of A Default Redhat Server 6.2 Honeypot by Stephen Holcroft on 16/07/03 ← back

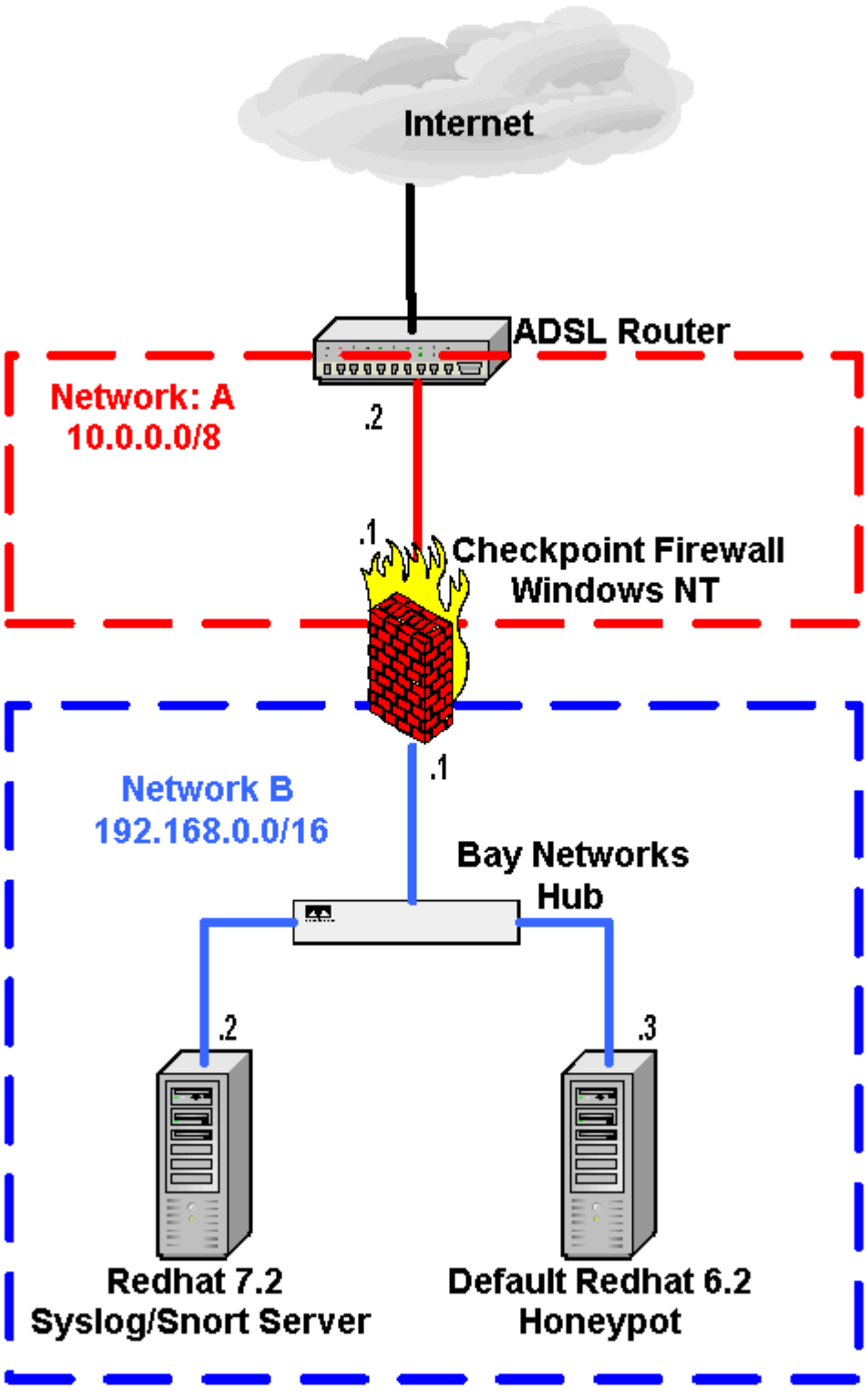
Introduction

The following paper is a description of how I have designed and implemented a honeypot system. The paper describes how the honeypot is used to capture data in layers using different techniques. The aim of the honeypot is to discover the techniques and tactics used by blackhats (hackers) to compromise computer systems. The methods used are similar to the methods used by the Honeynet Project.

The Honeypot

This particular honeypot used a default server installation of RedHat 6.2. I wanted to emulate the type of systems out there in the wild now and also give the blackhat a reasonable chance of success. There would be little point in setting up a RedHat 7.2 system with all the latest patches for my first attempt as I would be waiting a while for the compromise. I had recently asked a sysadmin I knew for a RedHat CD and they gave me version 6.2, so this seemed to be a reasonable representation of what was out there. The IP address of the honeypot was not advertised anywhere, there was also no DNS name associated with the IP so there was no reason for anyone to express any interest in the system (in theory!).

The honeypot I used sat on a network with 2 other machines. The following is a simple diagram of the network, the real IP addresses have been replaced with private addresses to preserve the identity of the honeypot:-



The firewall is a Windows NT machine running Checkpoint FW-1, this machine controlled who had access to which parts of the network. The set-up is very simple:-

Source	Destination	Port	Action	Track	Comment
Anywhere	Firewall	any	Drop	Long	Don't allow anyone to connect to the firewall and log
Firewall	Anywhere	any	Accept		Allow the firewall access to Anywhere
Anywhere	Syslog Server	any	Drop	Long	Don't allow anyone access to the Syslog server and log

Anywhere	Honeypot	any	Accept	Long	Allow anyone to connect to the honeypot and log
Honeypot	Anywhere	ftp, smtp, domain-udp, http	Accept	Mail	Allow honeypot limited access out and send an email/page notification
Honeypot	Anywhere	any	Drop	Mail	Drop any other access out from the honeypot and send an email/page notification
Anywhere	Anywhere	any	Drop	Long	Clean up rule, drop and log any other traffic

I was a bit concerened at first about allowing the honeypot access out onto the Internet after it was compromised, after all one of my rules was to not to put other systems in danger, allowing the hacker access out would do this. The last thing I wanted was my honeypot being used as a launching pad for a denial of service attack, scan, hack or any other bad behavior. I didn't want Scotland Yard knocking on my door to tell me my computer had been used to steal half a million credit card numbers. I needed a method of letting me know exactly when the system was compromised, that way I could log onto the firewall straight away and make sure when the hacker connected out from the honeypot he was not putting any other systems in danger. If he did then it would be easy to block the connection. Luckily Checkpoint FW-1 gave me the perfect way of doing this, the firewall program is able to send emails when logs are generated. I set it up so when there was any connection made out from the honeypot an email was generated, this was then forwarded to my mobile phone as a text message. I tested this before the honeypot was opened up by generating some traffic out from the honeypot and I received an alert on my phone 12 seconds after the traffic was generated.

I also decided to limit the type of traffic allowed out from the firewall. I decided on ftp as I knew the blackhat would probably want to download a rootkit from an ftp server after they compromised the system. I allowed smtp out as I know a lot of root kits send the system details to an anonymous email account after the compromise so the blackhat has a record of which systems he has access to. I allowed domain-udp as it would probably be needed by the blackhat for looking up domain names. Lastly I allowed http, as an alternative method used by blackhats to download rootkits is via a web site, this can be done via the command line using programs such as "wget". Any other services the blackhat needed could be added after the system was compromised, as long as they were not placing other systems in danger.

I also was also skeptical about having the syslog/snort server on the same network as the honeypot, the blackhat might try his luck and compromise it as well. They would not be able to gain access from the internet as the firewall would block this but if the honeypot was compromised they would have access to the syslog/snort server as it resides on the same network. I could have placed it on a different interface on the firewall to make sure all traffic between the two is controlled by the firewall but this would have meant I could not run snort on the syslog server. I decided to harden the syslog/snort server, I patched it completely up to date installing all the latest packages and kernel updates. I then turned off all services apart from remote syslog, this meant there would be no open ports for the blackhat to connect to. I then installed ipchains allowing only syslog from the honeypot, everything else was blocked. A scan of the system using nmap confirmed it was only listening on udp 514 (syslog).

Logging in Layers

I wanted to capture as much information as possible and I wanted it to be as reliable as possible. If I was to truly understand how the black hat had hacked me then I needed as much as I could get. I knew I had to capture the data in layers, so I decided to work with 6 different layers (data sources).

- Firewall logs
- Ethereal
- Syslog
- Snort
- Tripwire
- Bash keystroke kogging

The First Layer -- Firewall Logs
The Checkpoint firewall logs were the first layer of logging, this would show me details of any traffic entering and leaving the network. I would know the date and time, the IP addresses, the port they were trying to connect to and the time. I knew the firewall logs on the firewall would be accurate as I was the only person with access to it. Checkpoint logging is not the most detailed logging around but it serves its purpose as the first line of logging.

The Second Layer -- Ethereal

The second layer of logging was to use a packet sniffer on the firewall, the program of choice was Ethereal. This is a stunning little piece of software designed to capture every part of a packet as it passes through the firewall. I set this up so it would record packets on the firewall inside interface. I would recommend installing it if you haven't seen it yet. It is very easy to use and there is a neat feature that allows you to follow a connection from start to finish in clear text, for example the following output is how ethereal see's an ftp session. As you can see all the commands are presented clearly in plain text. This would be perfect for viewing commands issued from the hacker to the honeypot once it was hacked, the only potential problem is if the hacker uses ssh to connect as its packets are encrypted. Ethereal will record all packets going through the firewall in both directions and also all packets sent between the honeypot and the syslog/snort server.

The Third Layer -- Syslog
The third layer of logging was syslog on the honeypot itself. I wanted to read the syslog files but I knew if it was hacked there would be a very good chance the blackhat would erase the logs or modify them to remove his activities. So I decided to have syslog on the honeypot send the logs to a remote server. There was a problem though, if the hacker got in and looked at the syslog.conf file carefully (some root kits do this as part of their install process) and saw it was logging remotely to another machine they would very quickly realize something was up. I removed the syslog package from the honeypot, I then downloaded the source and re-installed it, this time changing syslog so it reads a different config file before re-compiling it. I left the original configuration file in the same place /etc/syslog.conf, but the actual file syslog was reading from contained the following line at the top:-

. @192.160.0.3

This meant to the backhat everything looked fine, in actual fact syslog was still logging locally but also sending every log generated to the syslog server which was set up so its own syslog accepted remote messages. I knew that the rootkit would most likely try and replace syslogd with a trojan version of its choice, if it did this then the logs would be unreliable I needed to find a way of stopping this. To prevent this I copied syslogd to another file name for example /usr/bin/exampled, I then killed syslogd and ran /usr/bin/exampled instead. This way as far as the rootkit/blackhat was concerned syslogd was not running, they could replace syslogd with a trojan but they would not be able to start it as syslog was already running under /usr/bin/exampled, this worked perfectly the last time the honeypot was hacked, the rootkit replaced syslogd but was unable to start the new trojaned version, the blackhat left everything as it was believing syslog was not running.
I knew if the hacker was good enough they would discover this anyway, if they installed a packet sniffer they would see the packets being sent to the syslog server. This was just to delay the process, and hopefully the average script kiddie would not realize.

The Fourth Layer -- Snort
The forth layer was Snort, this is a nice piece of IDS software, very easy to install from rpm and configured in a matter of minutes. Snort acts as a packet sniffer and IDS system depending on how you run it. I ran snort so it would record every packet entering the network and log them to /var/log/snort on the snort server. Like Ethereal, Snort would give me the IP address of the attacker, port numbers time etc.. Snort also acts as an IDS system so it compares the packets to well known attacks and generates an alert file if any attacks are recognized. Like Ethereal snort would capture all traffic entering and leaving the network and all traffic between the honeypot and the syslog server.

The Fifth Layer -- Tripwire
The fifth method of logging was to install Tripwire on the honeypot. Tripwire works by building a database of files on the system. When a tripwire check is run it compares the files on the system to the database and it alerts you if there have been changes to any of the files. This is excellent for a honeypot as it gives you an idea of what files the hacker has changed/trojaned after they hack the system. I saved the database and the config file to a floppy disk and removed it from the system. This would prevent a hacker modifying it corrupting the data.

The Sixth Layer -- Bash Keystroke Logging
The sixth and final method of logging was to patch the bash shell on the honeypot. The patch which is supplied courtesy of Antonomasia (ant@notatla.demon.co.uk) is applied to a clean version of bash version 2.03, you will need to download the source and apply the patch as directed in the code, make sure you link all the other shells to bash after you install it, you don't want the blackhat switching shells to avoid the logging. Once applied it then logs all keystroke commands issued on the honeypot to syslog which is then sent to the remote syslog server. This provides details of all the blackhats commands once he has

compromised the system. This is very useful as it allows you to track commands even when the blackhat uses ssh. Ethereal or Snort would not be able to record the commands if ssh was used as it encrypts the packets.

Conclusion

Six layers of logging might seem like overkill but it lets you build up a better picture of what's going on and introduces some redundancy. For example if ethereal crashed on the firewall you would still be able to see the contents of the packets through snort and vise versa. If the tripwire database on the floppy became corrupted then I would still be able to see the commands the blackhat typed on the honeypot using ethereal and snort or bash, this would give me an indication of what files were being changed. If the blackhat used ssh then the bash logger would still record commands issued.

It's all about recording as much reliable information as possible to assist you in working out exactly how the honeypot was compromised and the activities after. If you need further details on this honeypot or you have any questions/comments please feel free to email me.

Resources

Honeypots, Definitions and Value of Honeypots: http://www.enteract.com/~lspitz/honeypot.html
The Honeynet Project: http://project.honeynet.org/
Ethereal Home Page: http://www.ethereal.com/
Snort Home Page: http://www.snort.com/
Tripwire Home Page: http://www.tripwire.org/
Nmap Home Page: http://www.insecure.org/nmap
Bash Keystroke Logging Patch: http://project.honeynet.org/papers/honeynet/bash.patch
An Evening with Berferd: http://www.securityfocus.com/library/1793