Global Information Assurance Certification Paper

Detecting Compromised Systems With An Automated NMAP Fingerprinting Tool

Yet another layer of defense in depth

Jason Ostermann
SANS Track I - GSEC
Version 1.4b, option 1

Abstract

Every system requires many levels of defense against attacks. These defenses need to both help prevent intrusions and also detect intrusions. Many software packages exist to fulfill these roles, but all have certain weaknesses. Many remote exploits take advantage of buffer overflows in faulty software to install a backdoor which allows the intruder to use the system at their leisure. Current tools do not provide sufficient detection of such exploits. An automated fingerprinting tool based on NMAP can directly detect and report such a system change.

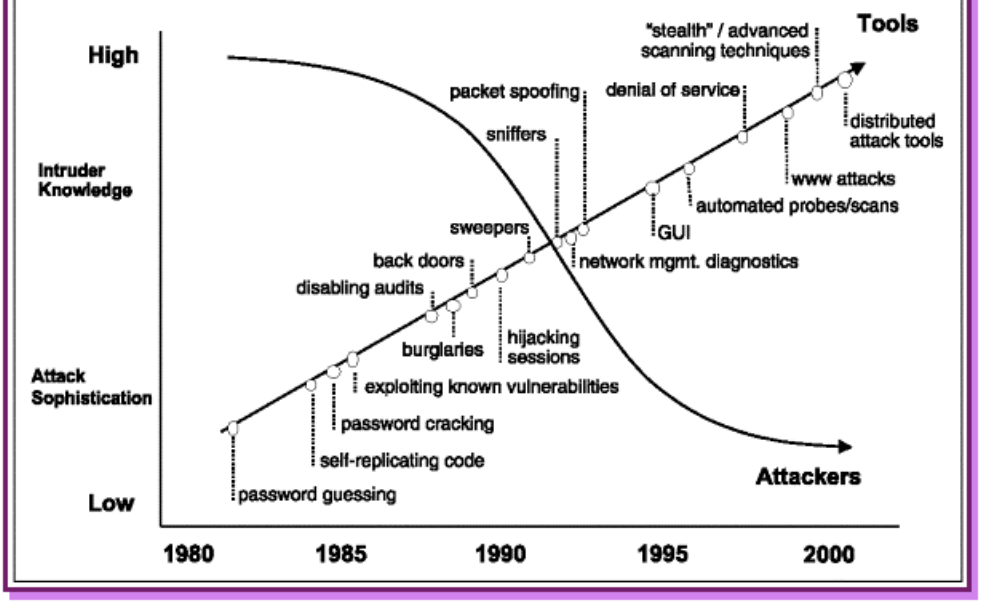
The Problem

In any well-configured network, a trade-off between security and usability is established to provide services and protect the machines from improper use. This paper focuses on the average publicly available machine on a DMZ providing services to the Internet at large. This may be a company's web server, an email server, DNS server, VPN server, or any other machine that is connected to the Internet. As with any installation, many levels of protection are recommended for such machines.

Network vulnerability probes have skyrocketed over the past few years. These probes may only be reconnaissance, or they may be an active attack against a machine. Neither Unix nor Windows systems are inherently secure from network attacks. In fact, both types of operating systems are regularly scanned and exploits are continuously released for both. (Current Scanning Activity) Attacks take many forms, from mostly harmless worms that only spread themselves to malicious code that modifies the host machine. Even harmless worms, such as the Robert T. Morris worm of 1988, need to be detected and prevented. While the RTM worm did not cause direct damage to the machines it infected, its uncontrolled behavior made many machines unusable (Spafford). While it appears this was an accidental side effect of the RTM worm, similar tactics are used to launch denial of service attacks today. These DOS attacks can easily cause millions of dollars of damage to corporations who find their Internet services effectively disabled.

Attack sophistication has increased significantly in the past few years, meaning that vulnerabilities are getting easier to exploit. Before the Internet boom of the mid 90's, a cracker needed a significant skill set to compromise a machine. Over the past few years, attacks have become more automated and more universal. (Allen, et all) The following graphic from "Improving the Security of Networked Systems" shows the trend towards more powerful tools allowing easier unauthorized access to machines.

Figure 1. *Attack Sophistication vs. Intruder Technical Knowledge*



The threat is significantly increased by the ease of use of current 'autorooters'. "A successful autorooter will give crackers what they want: complete control of a target machine with little effort, fast. Scanning networks for vulnerable machines, gaining unauthorized administrative access, installing backdoors, all the tricks of the trade, can all be achieved at the click of a button." (Tanase) With this type of tool, an entire class C subnet can be scanned and all vulnerable machines rooted in a matter of minutes. Adding worm capabilities to these programs is typically trivial. Distributed attacks of this nature will most likely become as commonplace as email viruses have become in the past five years.

This paper focuses on a specific, but common, category of exploit. These exploits use vulnerabilities within common server software, such as BIND, Sendmail, IIS, and Apache, to run arbitrary code. Almost all attacks will modify the machine in one way or another. Current exploits tend to install root kits or Denial Of Service daemons. Root kits allow the intruder to access the machine as root whenever they want, typically over a rogue telnet or ssh server. Denial of service daemons have either a timed attack or listen on the network for an attack command.

The root cause of the vulnerabilities is typically buffer overflows in network services. That simply boils down to bad programming. On the surface, buffer overflow exploits sound very complex. However, tutorials exist to assist in the creation of such exploits. (Aleph One) Once such an exploit is written, it is effective against all similar machines. In some instances, such as BIND, IIS and WU-FTPD, many overflows are discovered over a period of years. If all programmers followed strict security guidelines, vulnerabilities would sharply decrease. Since that's not likely to happen, users of software packages must

stay up to date. New releases typically patch discovered vulnerabilities, and in the case of Open Source projects, the patch can be written and distributed within hours of the discovery.
Even packages that have a good track record for security may contain a devastating vulnerability. The OpenSSH program had very few vulnerabilities in their past, with none allowing root access to its host. In 2002, a vulnerability was discovered in OpenSSH's authentication scheme that affected all recent versions which allowed an attacker to gain root access to the host. OpenSSH is primarily distributed as part of OpenBSD and developed to the same standards. The OpenBSD group prides themselves with proactive security auditing in all their software development. Their homepage, www.openbsd.org, even proudly proclaimed "No remote holes in the default install, in over 6 years!" After the OpenSSH vulnerability was discovered, a workaround was quickly published, and a new version with a completely different authentication scheme was released at the same time as the vulnerability report. (CERT® Advisory CA-2002-18) No exploits were reported, but the OpenBSD group still changed their motto to "Only one remote hole in the default install, in 6 years!" The upshot is that while keeping software current and watching CERT for advisories will help reduce your exposure, it is by no means a guarantee.

A wise administrator will use machine state tools such as Tripwire to monitor the vital files of their machines. If the attack modifies these files, such as adding itself to the startup scripts, a properly configured state monitor will detect the change and warn that administrator. Now that Tripwire has become fairly common, attacks can find ways to disable the service or bypass it completely. Exploits can be installed without changing any system files. Once the machine reboots or the rogue service crashes, the attacker simply needs to re-run their exploit to recreate their back door. In such an instance, there would be no changes to the system for a state monitor to detect. Such an exploit could install itself in a busy location, such as /tmp, and obfuscate itself to avoid manual detection. Since this exploit is careful not to alert a state monitor, it will avoid overwriting typical tools such as ps, netstat and ls. Without replacing these utilities with special versions designed to hide the existence of the exploit, the intrusion can be easily detected locally on the machine. Local detection using these utilities is seldom performed, and very seldom automated. Usually, a thorough search only occurs if a user or administrator notices strange behavior on a machine. This will usually be a very long time after the machine was exploited, and usually after much damage has been done.

After a machine is exploited, it presents a hazard to its owners and to all other machines connected to it. Just as open SMTP relays act as hopping points for spam distributors, exploited machines act as hopping points for system attackers. These compromised machines must be detected and quarantined as quickly as possible.

How to Detect

While preventing such attacks is difficult to impossible, detecting them is quite easy. All one has to do is determine whether or not a new network service has been started on a machine where no such changes should have been made. The NMAP (www.insecure.org) port scanner already provides a wonderful base to build upon. NMAP is a very easy to use network tool that scans specified hosts for all open network ports. (Fyodor) A rather simple set of scripts could be built around NMAP to provide automated scanning and reporting of specified machines.

When a system is in a known-good state, an NMAP report is generated as a baseline configuration, or fingerprint. This fingerprint lists the currently open listening ports. When the machine is known to be good, this list will only contain the services explicitly desired on that machine. If this list ever changes, there must have been a modification of the machine. NMAP only reports active listening ports, and does not detect connections to a machine. In other words, it doesn't matter how many clients are connected to services on that machine. All that matters are the available network services. Also note that it can only report services available to it. If a machine is extensively protected by a firewall, many attacks of this nature will result in new services that the attacker cannot access. For instance, if a root kit starts a telnet server on port 13373, but the firewall in front of the server only allows ports 80 and 443 to that machine, the exploiter will not be able to make use of the machine. Of course, this intrusion still needs to be detected so the faulty software can be corrected and to prevent any other attacks that may damage the system.

The following is an example NMAP run:

closet:~$ ./nmap 10.1.1.2 -sT -sU -oN 10.1.1.2.scan > /dev/null

-sT specifies a standard TCP connection scan. This is the simplest and most obvious scan. Since there is no need to act covertly, an obvious and reliable scan should be used.
-sU specifies to also scan UDP ports. While less common than TCP servers, a UDP server can easily be used to control a machine.
-oN specifies to output normal human-readable text to 10.1.1.2.scan.
The redirection to /dev/null is used because NMAP also echoes its report to stdout while running. Since this will most likely be run by cron, we do not want any output. Cron, by default, captures all output by scheduled programs and emails it to the user that scheduled the task.

closet:~$ cat 10.1.1.2.scan

```
# nmap 3.28 scan initiated Sun Aug  3 16:44:52 2003 as: ./nmap -sT -sU -oN
scan1.txt 10.1.1.2
Interesting ports on 10.1.1.2:
(The 3060 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
25/tcp     open        smtp
80/tcp     open        www
113/tcp    open        ident
# Nmap run completed at Sun Aug  3 16:48:17 2003 -- 1 IP address (1 host up)
scanned in 204.700 seconds
```

This run shows a fairly typical machine that only offers FTP, Telnet, SMTP, www and ident services. Keep in mind that a scan on a standard install for complex distributions such as Solaris typically result in over a hundred open ports. To modify this output into a fingerprint friendly format, the # comments and parentheses lines need to be removed. The comment lines will change with every scan, and the parentheses lines change whenever there is a service change. Since the number of closed ports is of very little usefulness for this system, that line can also be removed. These modifications can be placed into a single line as:

```
closet:~$ grep –v ^# 10.1.1.2.scan | grep –v ^\( > 10.1.1.2.fp
```

The resulting file, 10.1.1.2.fp, needs to be protected like any other critical system file. At the very least, its permissions should be read only for everyone. Better yet, it should be monitored by TripWire. Even better, it can be stored on a CD-R to prevent any modification whatsoever.

A periodic scan of the protected machines should be compared to the fingerprints generated earlier. If there are any differences, an alert should be emailed to an administrator, and/or an administrator could be paged. To compare results, the periodic scan should be processed just like the fingerprint, then compared to the fingerprint.

```
closet:~$ ./nmap 10.1.1.2 -sT -sU -oN 10.1.1.2.scan > /dev/null
closet:~$ grep –v ^# 10.1.1.2.scan | grep –v ^\( > 10.1.1.2.test
closet:~$ diff 10.1.1.2.fp 10.1.1.2.test
```

If diff reports anything, then the network state of the system under question changed, and should be investigated. The following is an example:

```
closet:~$ diff 10.1.1.2.fp 10.1.1.2.test
33a34,42
> 1014/udp   open        unknown
```

> 1015/udp open unknown
> 1016/udp open unknown
> 1017/udp open unknown
> 1018/udp open unknown
> 1019/udp open unknown
> 1020/udp open unknown
> 1021/udp open unknown
> 1022/udp open unknown

There are now nine new UDP ports open. If there have not been authorized configuration changes to the machine in question, this could be an indicator of an intrusion. An administrator should use this information to investigate the machine and determine the cause of the anomaly.

Making it usable

The system needs three main components: an installer, a periodic scanner, and a system modification tool. The installer should follow a process similar to the following:

1) Collect vital information, such as installation locations, methods to contact administrators, and scanning schedule.
2) Copy the various scripts to a secure location
3) Create a location to store the fingerprint data
4) Collect a list of machines, either via the terminal or from a file, to monitor
5) Add the scanning script to cron
6) Create the fingerprints

Since creating the fingerprints may require a very large amount of time, the script may wish to offer to schedule that task to run at a later time.

The scanning script needs to scan all the protected machines and compare the current scan's output to the fingerprint. Any differences should be stored in order to be relayed to an administrator and/or kept as evidence. After all the machines have been scanned, a report showing the failures or specifying all successes should be sent to an administrator. The following is an example script in pseudo-bash:

```
ERR=0
DATE=`date +%m%d%y`
for each machine; do
        nmap $machine –sT –sU –oN /tmp/$machine.scan.$DATE > \
                /dev/null
        grep –v ^# /tmp/$machine.scan.$DATE | grep –v ^\( > \
```

```
                /tmp/$machine.test.$DATE
        if diff $machine.fp /tmp/$machine.test.$DATE > /dev/null; then
                ERR=1
                echo “Differences detected on $machine:” >> \
                        /tmp/report.$DATE
                echo “(fingerprint to the left, current to the right)” >> \
                        /tmp/report.$DATE
                diff $machine.fp /tmp/$machine.test.$DATE >> \
                        /tmp/report.$DATE
        else
                rm –f /tmp/$machine.test.$DATE
        fi
done
if [ $ERR –eq 1 ]; then
        sendmail $ADMIN < /tmp/report.$DATE
fi
```

The primary pseudo code is the `for each machine` segment. If the scanning system simply stores each machine’s fingerprint in a known directory as <machine’s ip>.fp, then that line can be changed to:

```
for machine in `ls $fp_dir | sed s/\.fp//`
```

If the implimentor chooses to store the list of machines in a text file with one IP per line, that command would be similar to:

```
for machine in `cat $machine_list`
```

Modification tools should be provided to add or remove machines from the system and to update a machine’s fingerprint after an authorized change. While updating this monitoring system whenever making any other change to a network is bothersome, it simply needs to be added to the already long list of standard tasks to be completed. Just as tripwire must be updated when a major system file is changed, this monitoring system will need to be updated whenever a system change occurs. If the network is well planned and seldom changed, this should never be a problem. If the network is highly dynamic, the administrators may wish to tackle other problems before implementing a scanner.

Making it secure

The machine performing the network scans needs to be protected along with your other machines. Ideally, this machine will have unfiltered access to all the machines it needs to probe. That certainly does not mean that a firewall should be modified to allow all traffic from this machine into a private network without

any questions asked. An ideal implementation would have a scanner on each network with unfiltered direct access to all machines it needs to monitor. Since that is unlikely, a single machine in a private network may suffice. If at all possible, this machine should be protected by TripWire and should not have a console attached. The fingerprint library and TripWire database should be stored on a CD-R to prevent modification.

With some extra work, the entire scanning system can be chroot jailed and use only static binaries. For this to work, the fingerprint database must be mounted under the location of the scanning system. For instance, if the scanner is located in /usr/local/scanner, the fingerprint database must be somewhere like /usr/local/scanner/fpdb. The system would likely need static versions of the following programs installed within its scope:

- Bash
- Sendmail (client only, see)
- Sed
- Diff
- Grep
- Nmap

Each of these should provide minimal difficulty in compiling statically. If an attacker compromises this machine, it is unlikely that they would replace these static binaries with compromised versions. This safety measure would likely keep the scanner running even if the scanning machine were compromised.

Inherent Problems

There are a few inherent problems with such a system. Primarily, it provides a very promising list of machines and their services in one location. If the scanning machine is compromised and the attacker spends some time investigating, they may be able to find the fingerprint database. If the database is stored on a CD-R, then it will show up in the mount table, making detection easy. With such a database, an attacker would have a perfect list of machines to further investigate. This could assist their spread and success of attacks throughout the network. Keeping the fingerprint database on disk, so it does not show up in mount can mitigate the risk. The administrator can also obfuscate the location of the scanning system, such as somewhere under the vast X11 directory tree, or somewhere deep in /var.

If any of the monitored machines provide RPC services, the system may report many false positives. RPC services typically do not use predefined ports. Instead, they bind to a random port then register themselves with portmap. Since their port can change with just a reboot or process reload, the scanner would report a false positive. With some advancement, the scanner can be

modified to monitor RPC services explicitly. `rpcinfo` provides a list of RPC services available on a host. For hosts providing RPC services, an outline similar to the following would create a better fingerprint:

1) Scan the system as usual and process the NMAP output
2) Run `rpcinfo –p $MACHINE` to get a list of RPC services
3) Remove the services listed in the rpcinfo report from the fingerprint
4) Add a processed list of services to the end of the fingerprint

The scanner would use the same steps to generate a matching fingerprint, which could be directly compared to the original. This solution also requires the scanning machine to access the portmapper, typically on TCP port 111, on the monitored machine.

Conclusion

Attacks continually improve and advance. “Script kiddies” who use prepackaged exploits and do not possess the knowledge to manually exploit a machine are already widespread. Vulnerability probes occur continually. Simply put, system defenses must be extensive to protect one’s machines on the current Internet. For administrators who control only a few machines and continually access and monitor those machines, a scanning system such as this may not provide a valuable return on investment. For installations with many machines and very little individual monitoring, an automated scanning tool may detect many intrusions that went otherwise unnoticed. This tool is certainly not a silver bullet, but may prove to be exactly the right tool at the right time.

Aleph One. “Smashing the Stack for Fun and Profit.” Phrack Volume 7, Issue 49. November 8, 1996. URL: https://www.phrack.com/phrack/49/P49-14 (August 10, 2003).

Allen, Alberts, Behrens, Laswell and Wilson. “Improving the Security of Networked Systems.” October 2000. URL:http://www.stsc.hill.af.mil/crosstalk/2000/10/allen.html (August 2, 2003).

Carnegie Mellon Software Engineering Institute CERT Coordination Center. “CERT® Advisory CA-2002-18 OpenSSH Vulnerabilities in Challenge Response Handling.” December 6, 2002. URL:http://www.cert.org/advisories/CA-2002-18.html (August 20, 2003).

Carnegie Mellon Software Engineering Institute CERT Coordination Center. “Current Scanning Activity.” August 19, 2003. URL:http://www.cert.org/current/scanning.html (August 19, 2003).

Fyodor. “The Art of Scanning.” Phrack Volume 7, Issue 51. September 1, 1997. URL: https://www.phrack.com/phrack/51/P51-11 (June 30, 2003).

Spafford, Eugene. “The Internet Worm Incident.” September 19, 1991. URL: http://www.cerias.purdue.edu/homes/spaf/tech-reps/933.pdf (August 25, 2003).

Tanase, Matt. “Introduction to Autorooters: Crackers Working Smarter, not Harder.” August 21, 2002. URL:http://www.securityfocus.com/infocus/1619 (July 27, 2003).

The OpenSSH Group. “A Revised OpenSSH Security Advisory, preauth.” July 18, 2002. URL:http://www.openssh.com/txt/preauth.adv (August 15, 2003).