Hacking: An Analysis of Current Methodology

1 Abstract

Hacking has become a significant threat to networks exposed to the Internet. In order to prevent systems from being hacked, the methods used by hackers must be well understood. Hackers begin by selecting and footprinting a target network. Once the target network is mapped, hackers proceed to map vulnerabilities and gain access by cracking passwords, using stack-smashing attacks, or spoofing the IP address of trusted machines. Hackers can then sniff internal network traffic or find other hosts that contain vital company secrets. Finally, a hacker can clean up system logs in order to conceal the fact that an attack occurred. In this paper we explain how each of these attack techniques is carried out.

2 Introduction

The Internet has become a widely used medium for companies, schools, and governments to share data. Because of the need to exchange electronic information, most computer networks are connected and exposed to traffic on the Internet. With this exposure comes security concerns. Hackers are a significant threat because often all that lies between a hacker and a company's internal secrets may be a poorly administered firewall or border router.

How significant is this threat? A sampling of traffic into and out of a network over a few days will often show hundreds, perhaps thousands of potentially malicious data packets. Hundreds of hacking web sites have been born over the past years providing information ranging from how to spoof email to how to gain root access on web servers. Hackers need not understand the technology behind their methods; they can simply download a script and go to work. In essence, the threat of a network being attacked is significant and should not be taken lightly because even the novice hacker is capable of launching a potentially damaging attack.

How can hackers be stopped? To answer this question, it is important to understand how a hacker attacks a system. From footprinting to log cleanup, a hacker's methodology must be well understood so that firewalls and Intrusion Detection Systems (IDS) can be built to prevent or detect future attacks.

3 Prelude to An Attack: Footprinting

Before a hacker attempts to gain access to a system, time must be spent gathering information about the target. This process is known as *footprinting*, and it is a critical step in subverting the security of a target system [14]. Footprinting is the hacking equivalent to "casing" a potential robbery location. Systematic footprinting allows the hacker to create a complete profile of the target system including information about the

domain, network blocks, IP addresses exposed on the Internet, and system architecture. Once the profile is known, a hacker will be able to focus on specific machines and ports to gain access to the system. Knowing how a hacker gleans this information should encourage the owners and administrators of potential target systems to limit the amount of information they make available about their systems.

3.1 Open Source Footprinting [14]

The first step a hacker takes is to visit the web site of a potential target. What might he find? A hacker is primarily looking for information regarding the location and type of computer systems and also contact information for system administrators. The contact information can be used to guess passwords at a later time, or it can be used for *social engineering*. Social engineering is the process of gaining potentially vital company or system information from company employees or affiliates by pretending to be or know someone in the company. This is a particularly powerful technique but we will not devote time to discuss this non-technical aspect of footprinting.

Most web sites do not provide vital information about their systems, but we were able to find a web site, http://www.mhasd.k12.wi.us/, that detailed how to access the system remotely. In addition, the help option offered information about the format of passwords and logins (i.e., logins were simply first and last name and passwords could not include '&' or '+' characters). The names of potential users were also available at a different location at the site. Clearly, this could lead to a brute force attack where passwords were guessed from a fixed dictionary.[1] While obscurity should not be relied upon for system security, it is advisable that web sites do not offer system specifics.

3.2 Network Enumeration [14]

Network enumeration is the next step in gathering information about a target system. A hacker will identify domain names and the network blocks associated with the target. Before 1999, Network Solutions had a monopoly as the main registrar for domain names. There are now multiple registrars for domain names (for a complete list of registrars consult http://www.internic.net/alpha.html), with each maintaining some `whois` servers. `Whois` is a simple directory service that can be accessed directly from machines with Internet access. The first step is to determine which registrar the target is registered with.

The following is a sample `whois` query from Solaris machine.[2,3] The use of the wildcard "." should be noted. This can be particularly useful when the complete name of the target is not known. [4]

---

[1] This will be discussed in greater detail in section 4.1 Password Cracking.
[2] The equivalent `whois` search can be done from http://www.internic.net/whois.html.
[3] The example searches are for .com, .net, .org, and .edu targets. http://www.allwhois.com/ is a good source for querying `whois` servers outside the United States. This site can also be used to query .gov, .us, and other domain extensions.

```
nova38(2)% whois –h whois.crsnic.net maury.

Whois Server Version 1.3

Domain names in the .com, .net, and .org domains can now be
Registered with many different competing registrars. Go to
http://www.internic.net for detailed information.

. . . .
MAURYROGOFFPR.COM
MAURYRICE.COM
MAURYREGIONAL.COM
MAURYREALTY.COM
MAURYREALTOR.COM
MAURYREALESTATE.COM
. . . .
```

If the domain the hacker is looking for is on the resulting list, the registrar can be found by entering the query:

```
nova38(3)% whois –h whois.crsnic.net mauryregional.com

Whois Server Version 1.3

Domain names in the .com, .net, and .org domains can now be
registered with many different competing registrars. Go to
http://www.internic.net for detailed information.

   Domain Name: MAURYREGIONAL.COM
   Registrar: NETWORK SOLUTIONS, INC.
   Whois Server: whois.networksolutions.com
   Referral URL: www.networksolutions.com
   Name Server: NS2.CL.BELLSOUTH.NET
   Name Server: MRH-NT4.MRHS.COM
   Updated Date: 03-jan-2001
```

Once the registrar is known, more information about the domain can be garnered by querying the registrar's `whois` servers. Most registrars maintain a web site with easy to use `whois` search capabilities. The following is an example of what information can be found by directly querying the registrar.

[4] Consulting `man whois` or http://www.networksolutions.com/en_US/help/whoishelp.html will offer assistance for more advanced queries.

```
nova38(8)% whois -h whois.networksolutions.com mauryregional.com
. . . .
Registrant:
Maury Regional Hospital (MAURYREGIONAL-DOM)
   1224 Trotwood Avenue
   Columbia, TN 38401
   US

   Domain Name: MAURYREGIONAL.COM

   Administrative Contact, Billing Contact:
      Phillips, Terry  (TP2469)  tphillips@MRHS.COM
      Maury Regional Health Center
      1224 Trotwood Avenue
      Columbia, TN 38401
      931-380-4121 (FAX) 931-540-4144
   Technical Contact:
      Hostmaster  (HOS260-ORG)  hostmaster@BELLSOUTH.NET
      BellSouth.net 1100 Ashwood Parkway
      Suite 200
      Atlanta, GA 30338
      US
      (770) 522-6300
      Fax- (770) 522-4002

   Record last updated on 03-Jan-2001.
   Record expires on 06-Jan-2003.
   Record created on 05-Jan-1998.
   Database last updated on 25-Mar-2001 02:38:00 EST.

   Domain servers in listed order:

   MRH-NT4.MRHS.COM             205.142.119.2
   NS2.CL.BELLSOUTH.NET         205.152.16.8
```

The query offers important information about administrative contacts and the DNS servers used by the domain.

The last piece of information needed from `whois` querying is to find the network block used by the domain. Internet Information Center (InterNIC) is responsible for assigning IP addresses for IP-based networks. Addresses are usually assigned as either Class C or Class B blocks. Class C address block specifies the first 3 bytes of a 4-byte IP address, leaving room for 254 individual addresses.[5] Class B blocks only specify the first 2 bytes, allowing for approximately 65,000 individual addresses. Large networks reserve Class B blocks, mid-sized networks often reserve multiple Class C blocks, and small networks reserve a single Class C block [8]. Knowing the blocks reserved for the target system provides for a rough estimate of the target network size. The American Registry for Internet Numbers (ARIN) is the source database for network blocks associated with

[5] Addresses with 0 or 255 as the last byte are reserved and are never assigned to hosts.

domains. The query can be performed from http://www.arin.net/whois/index.html or from a command line as in the following example:[6]

```
nova16(5)% whois -h whois.arin.net mauryregnet
Maury Regional Hopital (NETBLK-MAURYREGNET)
   Information Systems
   1224 Trotwood Avenue
   Columbia, TN  38401
   US

   Netname: MAURYREGNET
   Netblock: 205.142.116.0 - 205.142.119.255

   Coordinator:
      Phillips, Terry  (TP2469-ARIN)  mrh@radio.fm
      931-380-4121 (FAX) 931-540-4144

   Domain System inverse mapping provided by:

   MRH-NT4.MRHS.COM             205.142.119.2
   NS2.CL.BELLSOUTH.NET         205.152.16.8

   Record last updated on 30-Dec-1997.
   Database last updated on 4-Apr-2001 22:42:19 EDT.

The ARIN Registration Services Host contains ONLY Internet
Network Information: Networks, ASN's, and related POC's.
Please use the whois server at rs.internic.net for DOMAIN related
Information and whois.nic.mil for NIPRNET Information.
```

In this example, the target site has 4 class C networks reserved 205.142.116 – 205.142.119. Therefore, the network has about 1000 potential targets, although only a small fraction of these will prove to be active hosts. This query also reiterates the DNS servers used by the target. This helps cross-reference information gathered from previous queries.

In summary, `whois` queries can be a powerful tool for gathering network block and DNS information. We will show how this information can be used to potentially gain access to the target system.

[6] ARIN does not always store the names of the target by domain name. A more general search like "maury" may have to be performed as an intermediate step to get the correct ARIN listing, which in this case was "mauryregnet."

3.3 Scanning

Once the network block is known, the next step is to determine which IP addresses are accessible and what services are running on those machines. This is done via a process known as scanning.

Scanning is usually performed with tools that attempt to disguise network reconnaissance. The premier scanning tool Network Mapper or `nmap` is available from Fyodor at http://www.insecure.org/nmap/index.html [4]. This is a robust scanning application that allows hackers to scan networks using methods that can elude network firewalls and even elude some of the best Intrusion Detection Systems (IDS). The full `man` page for `nmap` is available at http://www.insecure.org/nmap/nmap_manpage.html, but some of the important features are listed below:

```
>nmap - h
nmap V. 2.30BETA17 Usage: nmap [Scan Type(s)] [Options] <host or net list>
Some Common Scan Types ('*' options require root privileges)
  -sT TCP connect() port scan (default)
* -sS TCP SYN stealth port scan (best all-around TCP scan)
* -sU UDP port scan
  -sP ping scan (Find any reachable machines)
* -sF,-sX,-sN Stealth FIN, Xmas, or Null scan (experts only)
  -sR/-I RPC/Identd scan (use with other scan types)
Some Common Options (none are required, most can be combined):
* -O Use TCP/IP fingerprinting to guess remote operating system
  -p <range> ports to scan.  Example range: '1-1024,1080,6666,31337'
  -F Only scans ports listed in nmap-services
  -v Verbose. Its use is recommended.  Use twice for greater effect.
  -P0 Don't ping hosts (needed to scan www.microsoft.com and others)
* -Ddecoy_host1,decoy2[,...] Hide scan using many decoys
  -T <Paranoid|Sneaky|Polite|Normal|Aggressive|Insane> General timing policy
  -n/-R Never do DNS resolution/Always resolve [default: sometimes resolve]
  -oN/-oM <logfile> Output normal/machine parsable scan logs to <logfile>
  -iL <inputfile> Get targets from file; Use '-' for stdin
* -S <your_IP>/-e <devicename> Specify source address or network interface
  --interactive Go into interactive mode (then press h for help)
Example: nmap -v -sS -O www.my.com 192.168.0.0/16 '192.88-90.*.*'
SEE THE MAN PAGE FOR MANY MORE OPTIONS, DESCRIPTIONS, AND EXAMPLES
```

By using the stealthy tool `nmap`, a hacker can easily determine specific machines that are susceptible to compromise. The first step is to determine which IP addresses in the network block are machines that are live hosts. This process can be done using the Internet Control Message Protocol (ICMP). ICMP was designed as a simple protocol to report network error conditions and supply basic network information. Unfortunately, ICMP can be used by hackers for network reconnaissance.

ICMP is a particularly good protocol for identifying active IP addresses. Unlike the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), ICMP does not connect to a particular service on a given host, but rather attempts to contact the host

operating system. Knowing the IP address of a host is enough to determine if the host is alive—simply send it an ICMP echo request, a *ping*, and if it responds, you know the machine is alive. Thus, using ICMP to determine live hosts on a network is often termed *ping sweeping* [14].

A hacker using `nmap` can use ICMP to probe a network. The first method has `nmap` send ICMP echo requests to every potential address in the block. This is done by issuing the `nmap -sP` command. This is a particularly noisy scan that can be detected by an IDS that looks for a threshold amount of ICMP echo requests originating from the same location over a given amount of time. Most IDS's are capable of detecting this type of scan, so it is not often used in practice [14].

If a hacker can limit the amount of echo requests issued, it may be possible to not trigger an IDS alarm because the threshold amount of ICMP requests will not be exceeded. By issuing ICMP echo requests to known broadcast addresses, the number of pings can be kept to a minimum. As previously stated, addresses with 0 and 255 for the last byte are reserved. These addresses are usually used as broadcast addresses to send a message to all addresses in the network. Therefore, a clever hacker will send ICMP echo requests to the 0 and 255 addresses for a given block and potentially get back 254 echo replies. This is much more efficient and stealthy than a scan that sends individual requests to all 254 potentially active machines [11]. Since fewer addresses need to be pinged, `nmap` can be used by entering the broadcast addresses individually using the –sP switch.

Sometimes networks are subnetted and there are more broadcast addresses than the standard 0 or 255 addresses. By sending an ICMP request type 17 (address mask request) to border routers,[7] the network mask can be obtained. The network mask is typically represented as a hexadecimal number that indicates how many bits of the IP address specify the network and how many specify the host. A standard class C network will have a mask of 0xFFFFFF00 with 24 bits dedicated to determining the network and 8 bits to determine the host. For such a network, the broadcast addresses are likely to be 0 and 255. A subnetted network might have a mask such as 0xFFFFFFC0 indicating the network is divided into 4 different entities. Thus, pinging 0, 63, 64, 127, 128, 191, 192, and 255 addresses would be prudent since these are the likely broadcast addresses [11].

ICMP appears to be a great tool for hackers to map active IP addresses in a network block. Security conscious network administrator have taken measures to limit the ICMP traffic in and out of their networks using firewalls and/or border routers. Many networks do not permit inbound ICMP echo requests at the cost of losing some network diagnostic capabilities. In addition, some routers are configured to not return "host unreachable" messages or reply to subnet mask queries. Finally, broadcast addresses are typically blocked for users outside the network, rendering the stealthy ICMP sweep techniques

[7] The addresses of border routers can be discovered using a tool called `traceroute`. This is available at ftp://ftp.ee.lbl.gov for UNIX and it comes with Windows NT but is called `tracert`. This utility will display the path a packet takes en route to the specified network.

useless. Thus, in practice ICMP has limited usefulness when targeting a security aware network [11].

If ICMP cannot be used effectively to map a target network, a hacker will turn to port scanning techniques to achieve the same results. Port scanning is the process of determining the available services on a host. TCP and UDP messages specify a 16 bit logical port[8] on the destination machine. Most ports have standard services running on well known ports (i.e., telnet typically runs on port 23), thus hackers can target specific services by querying specific port numbers.

Port scanning can be subdivided in three groups: horizontal, vertical, and block scans [18]. A horizontal scan is a scan that queries a specific port on numerous machines. This is used when an exploit is known for a particular service and the hacker wants to know what machines are running this vulnerable service. An example would be scanning for the notoriously vulnerable `wu-ftpd` on port 21. Alternatively, if a hacker wants to target a specific host, a vertical scan will be used in which all the ports on a given host are queried. For example, if a hacker wants to alter the content of the Maury Regional Hospital web site, all ports on the web server 205.142.119.2 would be scanned. Finally, a block scan is a combination of a vertical and horizontal scan. A block scan can determine the same information as an ICMP ping sweep (i.e., what machines are active in the network block), with the added benefit of determining the services running on the active hosts. An example would be scanning all the ports in the network blocks 205.142.116 – 205.142.119 to look for a vulnerability in the Maury Regional Hospital network.

No matter what scan type a hacker is using, there are methods to stealthily scan so that an IDS cannot detect the activity. `nmap` provides hackers will an arsenal of scanning options that will evade many IDS's that are currently in use.

`nmap -sT` establishes a full TCP connection with the host. TCP requires a three-way handshake between host and client to insure the reliability of the connection. The client sends a TCP message with the SYN flag set in the TCP header. The host will respond with the SYN and the ACK flags set. Finally, the client completes the handshake by sending an ACK to the host. The client and host have now established a reliable full-duplex connection in which messages can be sent from either the client or the host [8]. This type of scan is the default scan used by `nmap`, but it is not very popular in the hacking community. Once a full connection is established, the host will likely log the interaction. Clearly a hacker wanting to perform a stealthy scan will look toward other options.

The most popular scan type is the SYN scan. This connectionless scan can be performed using `nmap -sS`. This scan sends a message with a SYN in the TCP header but does not reply to the host's return message. Because the client never replies to the host's

[8] There are 65,536 logical ports on any given host.

return message, a full connection is never established and the "half connection" will not be logged. If the host port is open, a message with the SYN and ACK flags will be returned. If the host port is closed, a message with the RESET flag will be returned. Additionally, if a host is contacted that is not alive, a border router will likely respond with a "host unreachable" message. If the network is configured to not respond with ICMP host unreachable messages, there will be no reply when attempting to scan an inactive host. If the port is firewalled, there will also be no response. Thus, the hacker must attempt to differentiate between an inactive host and a firewalled port based on other data obtained in the scan [11, 14, 18].

As seen in the SYN scan, firewalls can present a problem for scans. The ACK scan is a scan that attempts to trick a firewall into letting messages through that would otherwise be filtered. Using an ACK scan in conjunction with a SYN scan can help determine firewall rule sets. In addition, an ACK scan can be used in place of an ICMP ping sweep. This scan can be performed using `nmap –sA`. Most firewalls will filter messages that are attempting to connect to a particular host or port, but will permit messages from "established" connections. The ACK is sent with all TCP messages after the instantiation of the full-duplex connection, so messages with the ACK flag set will be considered messages from established connections. Firewalls that save state might be able to stop such a scan by knowing that a 3-way handshake between the client and host did not occur. The FIN scan is similar to the ACK scan and can be performed using `nmap -sF`. The FIN flag is used to gracefully terminate a TCP connection, so firewalls will often allow these messages through [11, 14].

Although all of the above described scans use the TCP protocol, hackers may want to scan for UDP ports such as DNS port 53. This type of scan can be done using `nmap -sU`. UDP is an unreliable, connectionless protocol in which the host does not return a message to the client. If a host is unreachable, UDP messages expect an ICMP host unreachable message in return. The results of UDP scans are more difficult to interpret since no return message from the host could mean any of the following: the border router is configured to not return ICMP host unreachable messages, the unreliable UDP message never made it to the host, or the port was open on the host. For greater reliability, UDP scanning must be used in conjunction with TCP scans to map router configurations [11,14].

The scanning methods described can be made stealthier using some of the advanced features of `nmap`. `nmap` can fragment the TCP header using the –f switch so the message is sent in many TCP packets. Stateless firewalls and routers look at each packet received to determine if contact with the port or host destination is permitted. If the device does not save the previously encountered fragment, it will not be able to determine the port and destination of the packet and will likely allow the fragment to pass through [11]. Fragmentation is particularly powerful because even the best detection systems like Silicon Defense's *Snort*, are unable to correctly collect fragmented traffic, reassemble the stream, and flag it as potentially hostile [18].

In addition to fragmentation, `nmap` supports some other features to mask scanning activity. The `-D` switch sends scanning messages from forged IP addresses in addition to those from the true client. An IDS will likely detect all of the scans, but it will be difficult to discern which is the true scanning address [11, 14, 18]. Another IDS-busting flag is the –randomize_hosts flag. Some simple IDS's look for sequential scans of network addresses. This flag will scan the hosts in random order thus evading the watchful eye of the network IDS. Finally, many IDS algorithms look for a threshold amount of messages from the same source within a fixed time period. If the threshold time period is exceeded, the scan will not be flagged. This can be done with the `-T` flag in `nmap`. `nmap` supports both very slow scan speeds (referred to as "paranoid") and very fast scans (referred to as "insane"). The paranoid scan will likely be too slow for an IDS to detect, while insane should only be used if detection is not a concern [11, 14, 18].

Current commercially available IDS's are not able to detect all of the stealthy scans that `nmap` supports. Standiford et al. proposed a prototype IDS in a recent paper "Practical Automated Detection of Stealthy Portscans." The system proposed is called SPICE (Stealthy Probing and Intrusion Correlation Engine). This portscan detector would be able to detect slow, fragmented, and distributed scans that current IDS's like Snort [16] and EMERALD [17] are unable to detect [18]. Until such a product is in wide use, `nmap` will continue to provide hackers with the ability to map networks without being detected.

3.4 Stack Fingerprinting [14]

Once the hosts and port have been mapped by scanning the target network, the final footprinting step can be performed. This step is called *stack fingerprinting*. Stack fingerprinting is the process of determining the operating system and possibly the versions of services running on the target hosts. The two primary methods used to fingerprint are *banner grabbing* and *active stack fingerprinting*.

Banner grabbing is a simple technique that merely requires the ability to telnet to a target port on a host. After hitting `Enter` a few times, the *banner* is sometimes returned specifying the vendor and version of the service running on the port.[9] Telnet is a versatile tool because the destination port can be specified as the second command line parameter. Thus, if we want to query a web server, we would specify port 80 after the host address. In the following example, the web server is specified in the returned text:

[9] Entering a valid command without valid parameters is also sometimes used. For example, entering 'get' without any parameters when connected to a HTTP port will often prove useful.

```
nova55(12)% telnet www.mauryregional.com 80
Trying 205.142.119.2...
Connected to mrh-nt4.mrhs.com (205.142.119.2).
Escape character is '^]'.
get
HTTP/1.1 400 Bad Request
Server: Microsoft-IIS/4.0
Date: Sun, 22 Apr 2001 20:57:11 GMT
Content-Type: text/html
Content-Length: 87

<html><head><title>Error</title></head><body>The parameter is
incorrect. </body></html>Connection closed by foreign host.
```

Banner grabbing is a simple but it has some drawbacks. First, opening a telnet connection to many ports on many hosts can be time consuming. In addition, if many connections are established, an IDS may be able to draw a correlation between the connections and flag the traffic as potentially hazardous. Finally, a security-conscious network administrator will turn off banner presentation in the service software. If this is the case, determining the vendor and version of a service will be more difficult and may have to be inferred using other fingerprinting techniques.[10]

Another fingerprinting technique is called active stack fingerprinting. This technique involves sending packets to services on a host and analyzing the TCP/IP stack behavior in returned packets. Because each OS vendor interprets the RFC standards for TCP/IP stack implementation differently, the OS can be determined based on knowledge of how different OS stacks respond to requests. An example would be a FIN probe. When a FIN packet is received, the receiving stack should terminate the connection but not respond to the sender. However, Windows NT machines respond with a FIN/ACK packet.

`nmap` is also the tool of choice for stack fingerprinting. The `-O` option can be used with a host address and `nmap` will make an educated guess about the host OS. `nmap` uses an OS signature file called `nmap-os-fingerprints` to determine the OS.[11] This file contains hundreds of OS idiosyncrasies that help increase the accuracy of `nmap`'s guesses.

Like banner grabbing, active stack fingerprinting has some drawbacks. Clearly, `nmap` will not be 100% accurate so a hacker expecting to use a particular exploit for a specific service might have a rude awakening when the exploit fails. In addition, IDS's are becoming more aware of probes using bogus TCP flag combinations. `nmap` uses these when probing a host, so this activity may be more likely to be detected.

[10] Many OS's come with standard service packages. If the OS can be determined, the service can be guessed with a high probability of accuracy.
[11] This file can be added to by submitting a new signature via http://www.insecure.org:80/cgi-bin/nmap-submit.cgi.

Stack fingerprinting is a significant step in footprinting. The hacker will have been able to determine the hosts, the services running on these hosts, the vendor and version of the active services, and the OS of the host. Using this information, a hacker will be able to determine vulnerabilities that can be used to gain access to the target system.

4 Gaining Access

After a target system has been footprinted, a hacker will attempt to gain access to the network. Using knowledge of the hosts, the hacker will determine what vulnerabilities may be present and attempt to exploit them. Techniques used to access a system range from the straightforward task of password cracking to the more complex task of IP spoofing.

4.1 Password Cracking

Depending on the network, password attacks can range from being extremely efficient to completely ineffective. In all cases, password attacks are the easiest to understand conceptually, so most attackers begin their attacks by trying one or more of them.

4.1.1 Online Password Attacks

The most straightforward password "attack" is online guessing. The attacker finds himself sitting in front of a login prompt, knows one or more legal login names, and begins trying to guess the passwords for these login names. If the attacker is familiar with the owner of the account he is trying to crack, he may be able to narrow the search space of possible passwords. For example, if I were trying to crack John Tobler's password, I would first try words and phrases associated with Wake Forest University (his beloved alma mater).[12] Barring familiarity with the victim, the attacker looks specifically for accounts that have rarely (if ever) been used. These accounts may still be using default passwords that have never been changed by the account's owner. The word "password" is commonly used as a default password. For active accounts, a shockingly effective attack is to simply try a user's login name as the password.[13] Reversing the letters in the login name also works surprisingly often. For example, if the login name is `koconnor`, the attacker will try the password `ronnocok`.

Online guessing attacks have obvious pitfalls. A common problem is that many accounts are automatically disabled after an incorrect password is entered five times in a row. Although this also makes these accounts susceptible to a denial of service attack, it does successfully thwart the lion share of guessing attacks.

[12] In fact, John Tobler's password *is* a phrase related to Wake Forest University!
[13] Ian Alderman has had success with this style of attack.

Another problem with password guessing is that it is a slow process. Login prompts are wise to password guessing attacks and will pause for a moment after an incorrect password is entered. These pauses, however brief, can add enough overhead to the password guessing process that the attacker will abandon it in favor of another exploit. A notable exception to this, however, is when the login prompt is served up by a web server. According to Ian Alderman, most web servers impose no delay on wrongly entered passwords, thereby allowing attackers to efficiently bang away at the login prompt.

Lastly, guessing attacks (like most other attacks) are susceptible to logging by a system administrator. Unless the attacker is successful in compromising the system and then cleaning these logs, the system administrator will know the system has been attacked. The administrator will then take a fine-toothed comb to the machine and the rest of the network, making it highly likely that other exploits initiated by the attacker will be discovered and eradicated. Logging is not perfect however, as Ian Alderman informed us. As an example of this, consider the `su` command, which is used by ordinary users to assume the role of the root user. When `su` is executed, it has the following behavior:

- If the password is correct, immediately return control to the user.
- If the password is wrong, wait one second, log the event, and then return control to the user.

Attacking this can be accomplished by the following steps:

1. Guess the root password.
2. If `su` returns control to you in less than half a second, you are root.
3. Else, hit `Ctrl-C` to abort the `su` command. Go back to Step 1.

Nothing shows up in the log files since the failed password attempt is not logged until after the one-second delay. Moreover, the attacker does not need to wait for the entire one-second delay before guessing the next password.

4.1.2 Offline Password Attacks

Offline guessing of passwords is another approach to cracking passwords. Some Unix systems store the encrypted version of every user's password in a world readable file, `/etc/passwd`. The following is an excerpt from an example[14] `passwd` file:

[14] This `passwd` file comes from a password cracking tutorial at http://www.hackersclub.com. It was allegedly taken from a real-world system.

Figure 1. An example `passwd` file.

```
root:RqX6dqOZsf4BI:0:1:System PRIVILEGED Account:/:/bin/csh
preardon:jVaNsHh6f0vPw:596:15:Pat Reardon:/usr/email/users/preardon:/bin/csh
thutchcr:Hdy6Y.86nVZTQ:388:84:Todd Hutchcraft:/usr/stusers/users/thutchcr:/bin/csh
whenley:YvQ641.sul3Gg:1349:84:William Henley:/usr/stusers/users/whenley:/bin/csh
```

There are seven fields in a `passwd` file, the first two of which are a login name and an encrypted password. The encrypted password is typically created by first appending a 2-byte "salt" value to the user's clear text password and then passing this through a well-known one-way hash function. The salt value is randomly chosen among 4096 possibilities so that the same clear text password can be encrypted in 4096 different ways.[15] The hash functions are known to be secure in the following sense: Given an encrypted password, `X`, it is computationally infeasible to find any string that will hash to `X`. This is known as 2nd pre-image resistance. Therefore no attacker will spend his precious time trying to reverse engineer the hash function.[16,17]

Instead, attackers sidestep this difficulty by making the following observation: many users choose simple, easily guessed passwords. The only potential obstacle is to acquire the `passwd` file for the target system, and this is often as easy as anonymously `ftp`'ing to the target and `get`'ing it since it is world-readable. The attacker then feeds the `passwd` file to the latest version of his favorite password-cracking program, which will output the login name and associated password for each password it successfully cracks. Currently the most popular password cracking programs are *Crack* [10] and *John the Ripper* [2], both of which are freely available for download from the web.

Crack and *John the Ripper* both provide three different modes of operation, corresponding to how determined the attacker is. In the most basic mode, you feed the cracking program the `passwd` file along with a "dictionary" file of possible clear text passwords. The cracker then hashes each word in the dictionary file and compares it to each entry in the `passwd` file—if there is a match, an account has been compromised. Running *John the Ripper* on the above excerpted `passwd` file with a generic dictionary file produced the following results:

[15] For authentication purposes, the two byte salt value is stored in clear text in the `passwd` file and is pre-pended to the encrypted password. For example, in Figure 1, `Rq` is the salt value that was used to encrypt root's password.
[16] There are notable exceptions to this general rule. Windows NT, for instance, uses a proprietary hash function that can be broken relatively easily.
[17] For years the de facto hash function for Unix systems has been the standard system function `crypt()`. Recent distributions of Linux now support password hashing with the MD5 instead of Unix's standard `crypt()` function.

Figure 2. Cracked accounts:

Password	*Username*
hammer	preardon
julianne	thutchcr
marino	whenley

Of the 509 entries in the complete `passwd` file, John the Ripper successfully cracked five using the generic dictionary. After downloading additional dictionaries from http://hackersclub.com/km/files/password_cracker/wordlists/index.html, we were able to crack another handful of accounts. A dictionary of women's names, for one, turned up a user with a password of `Hollie`.[18] At the Hacker's Club website alone, there are well over 150 different dictionary files ranging from sports words to Yiddish words to Shakespearean characters [7].

The second mode of operation also uses a "dictionary" file, but it also uses a set of rules to morph each dictionary word into other closely related words. For example, there are rules which replace the letter 'I' with the number '1' and the letter 'O' with the number '0'.[19] Thus, if the dictionary file contains the word 'IDIOT', the cracker will check if any account uses the password '1D10T', 'ID10T', etc. This effectively increases the size of the dictionary and thereby increases the chances of cracking a password. Of course it also increases the amount of time it takes to crack the `passwd` file.

The third mode of cracker operation does not use a dictionary file, but instead launches a purely brute force attack on the `passwd` file. The cracker systematically searches the entire space of all possible clear text passwords in the search of account passwords. While the first two modes of operation are typically useful when trying to crack *any* account in the `passwd` file, this third mode is most commonly used when the attacker wants to crack a specific account (such as the root user's). Note that even though the attacker is focusing on only a single account, the computational effort required to search all possible passwords is likely to be daunting.

One might wonder whether a clever attacker could pre-compute a lookup table of all possible clear text passwords paired with their one-way hashes. This would save him the trouble of using a dictionary or brute force attack because he could efficiently lookup the clear text corresponding to any encrypted password.[20] Suppose the attacker has a

[18] The choice of women's names was recommended to us by Ian Alderman. In his own experience, he had found that men frequently choose women's names as passwords. Indeed, we see in Figure 2 that user `thutchcr` is another example of this.
[19] System administrators encouraged such substitutions for a while, but attackers soon picked up on it.
[20] The attacker could pre-compute a hash table of encrypted passwords and their clear text equivalents. This would allow him to determine a clear text password from an encrypted password in almost O(1) time.

dictionary of half a million entries[21] each of which are eight bytes in length. This will require 8MB to store the dictionary itself and just over 16GB to store the 231 encrypted passwords that are each 13 bytes long. It was not too long ago that a 16 GB hard drive seemed almost impossibly large, but today any attacker will have a hard drive at least twice this big (leaving plenty of room left over for Warez). This accentuates the need to choose passwords that are not easily derived from dictionaries.

We conclude this section by noting that current systems offer an alternative to the world-readable `passwd` files we have been discussing up to this point. This alternative moves the encrypted passwords from the world-readable `passwd` file into a file that is readable only be the root user. In a sense, part of the `passwd` file is now being shadowed elsewhere, so the scheme is referred to as a *shadow password* scheme. The `/etc/passwd` file still exists and is world readable, but the field for the encrypted password is now replaced by a meaningless, place-holding character.[22] For example, if the `passwd` file in Figure 1 were shadowed, it would look like the following:

Figure 3. A shadowed `passwd` file.

```
root:*:0:1:System PRIVILEGED Account:/:/bin/csh
preardon:*:596:15:Pat Reardon:/usr/email/users/preardon:/bin/csh
thutchcr:*:388:84:Todd Hutchcraft:/usr/stusers/users/thutchcr:/bin/csh
whenley:*:1349:84:William Henley:/usr/stusers/users/whenley:/bin/csh
```

The encrypted passwords are now replaced by asterisks, providing no password information to the attacker (although it does provide the attacker with a handy list of valid login names).

As best as we can discern, shadowed password files are a security measure that stifles offline password guessing rather effectively. We could not discover any exploits against systems using shadowed password files. When asked, even the members of Berbee's red team were unaware of any such exploits.

4.2 Stack Smashing [12]

One method for gaining unauthorized root access on a machine is to mount a stack smashing attack against it. There are two basic steps in this attack. First, the attacker locates a stack smash-able program that runs with root privilege on the target machine. Second, the attacker executes the stack smash by injecting malicious code into a buffer in the vulnerable program. When the vulnerable program begins to exit, the malicious code is triggered and transforms the vulnerable program into a root user shell—a hacker's

[21] We're being a little generous. The College Edition of Webster's English Dictionary has fewer than 200,000 entries. Maybe the hacker has also included Webster's Swedish Dictionary in honor of his homeland.
[22] The place-holding character is useful for backward compatibility reasons.

dream. In this section we discuss the technical challenges facing an attacker who wishes to launch a stack smashing attack.

The first challenge facing the attacker is to find a stack smash-able program running with root privilege on the target machine. The notion of a "stack smash-able" program is essential to understanding how stack smashing attacks work, so we will drive it home by way of an example program. The program, written in C and fittingly named `Smashable.c`, contains the two essential elements that make a program stack smash-able, namely:

1. A buffer, say `input_buffer`, that can be written to by a user of the program
2. A copy into `input_buffer` that is carried out by a function that does not do bounds checking

Smashable.c

```
#include stdio.h

main (int argc, char *argv[]) {
      char input_buffer[128];
      strcpy(input_buffer, argv[1]);
}
```

A quick inspection of the `Smashable.c`'s code shows that it meets both criteria for being stack smash-able. First, the buffer `input_buffer` can be written into by any user of `Smashable` by simply passing a command-line parameter. Second, the function `strcpy` fails to do bounds checking when copying this command-line parameter into `input_buffer`. In particular, this means that `strcpy` will not complain if the length of `argv[1]` is greater than the 128 bytes that have been allocated to `input_buffer`. Instead, `strcpy` will quietly overwrite the addresses that immediately follow `input_buffer` in memory.

Now that we know what a stack smash-able program looks like, we will explain how an attacker is able to exploit stack smash-able programs to his advantage. The key lies in the point just noted—buffer copying functions like `strcpy` that do not do bounds checking will overwrite the memory addresses immediately following the buffer they are copying into. Let us take a close look at exactly what would get overwritten in our `Smashable.c` program and we will see exactly why stack smash-able programs open us up to stack smashing attacks.

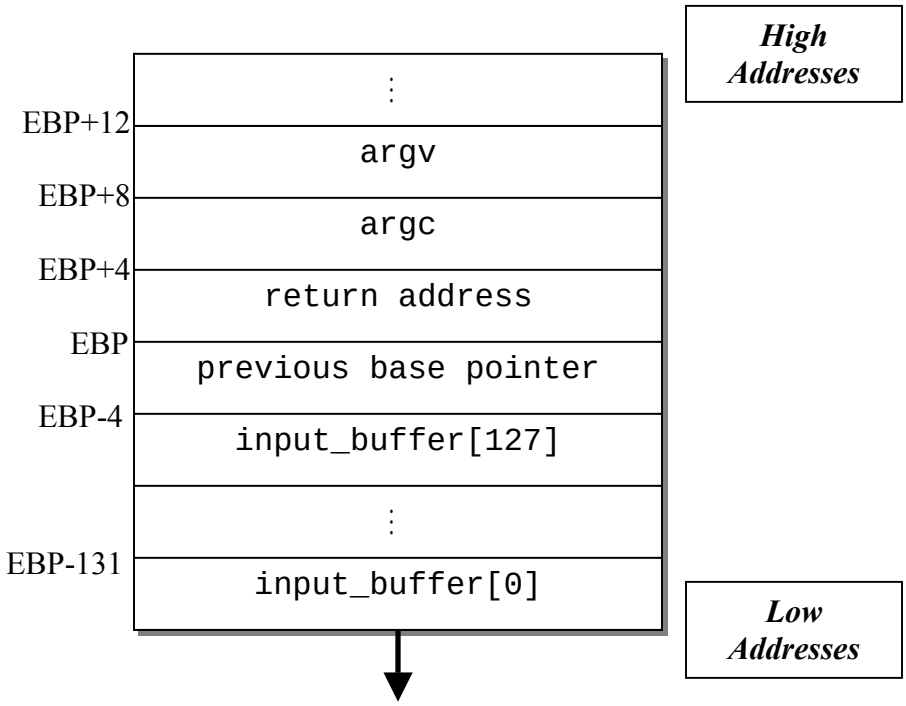

Figure 4. Depiction of the stack immediately after `Smashable` is executed. The arrow indicates that the stack is growing from high addresses to low addresses. EBP, the *base pointer*, is a convenient reference point in the stack and is explained more fully in the text. 4-byte memory addressing is assumed.

Figure 4 depicts what memory looks like in the neighborhood of `input_buffer`. All of these addresses belong to a region of memory called the *stack*, which is so named because it grows and shrinks dynamically in a Last-In-First-Out (LIFO) order. We will assume `Smashable` is running on an Intel[23] processor in which case the stack will grow from high memory addresses towards lower memory addresses.

How do we know that the stack looks like Figure 4? This is because the layout of the stack is determined solely by programming conventions that are designed to make a programmer's life more convenient. This convenience has two aspects, both of which are exploited by the stack-smashing attacker. First, the stack makes it easier for the programmer to access function parameters and local variables, both of which are stored on the stack. This is because they are stored relative to a dedicated register, referred to as the base pointer register, or EBP, that remains constant during the "lifetime" of a function

[23] Motorola and MIPS use the same convention.

call. For example, in `Smashable`, the programmer knows that the first parameter to `main`, namely `argc`, will be stored at the address (EBP + 8). Likewise, the programmer knows that a local variable like `input_buffer` will be stored just below where the base pointer points, namely at the address (EBP – 128). Second, the stack facilitates returning from function calls by storing the return address at a standard location.

Let us "watch" the invocation of Smashable in order to see exactly how the stack is built up on its way to resembling Figure 4. Just before the function call to `main` is made[24], the function parameters `argc` and `argv` are pushed on `Smashable`'s stack so that `Smashable` will be able to access them. Next, the call to `main` is made. This first pushes the current program counter onto the stack and then sets the program counter to the address of `main`'s first instruction. This value that is pushed on the stack is the return address. Third, as `Smashable` begins executing its own code, it saves the current base pointer register onto `Smashable`'s stack and then loads the base pointer register with the current value of the stack pointer. This way the base pointer that was being used by the function that called `main` can be restored when `main` returns. Finally, `Smashable` allocates enough space on its stack to hold its local variables, which in this case is just `input_buffer`. The final result is the stack depicted in Figure 4.

We can now, finally, answer the question of what gets overwritten when `input_buffer` is overflowed. The first four bytes that are overflowed hold the value of the saved base pointer. Although overwriting this value will likely crash the function that called `main`, it is of no use to our stack-smashing attacker. Rather, what our attacker is after is the value stored in the next four bytes—the return address. The attacker's goal is to overwrite this return address so that it points into `input_buffer`, where the attacker will have previously inserted malicious code. This way, when `main` returns it will begin executing this malicious code. What's worse, if `main` runs with root privelege, then the attacker's malicious code will also run with root privilege. Obviously it is worth the attacker's effort to find a stack smash-able program running with root privilege.

At this point, launching the stack smashing attack just requires working out a few details. For example, what should the malicious code do? More often than not, the malicious code will simply `exec` a shell. This gives a lot of bang for the buck since a very small snippet of code is able to create a launching pad for a huge number of other exploits.

There is also a subtle restriction on the bytes that make up the malicious code, namely that none of them can be zero. This relates back to what makes a program stack smash-able. If any of the bytes of the malicious code were zero, then the buffer copying function (e.g., `strcpy`) would interpret this byte as a null byte and prematurely stop copying bytes into the target buffer. The stack smash would then almost certainly fail

[24] The caller of `Smashable`'s `main` function can logically be considered the operating system. However, what's important to the current discussion is simply that a call was made—not who made the call.

since the targeted buffer would most likely not have been overflowed. Wily attackers can easily avoid this complication however by clever substitutions for code bytes that would otherwise be zero. For example, a simple substitution for an instruction that uses the constant zero is the following:

Before	**After**
`mov $0, eax`	`xor eax, eax`

The "after" instruction performs the same operation as the "before" instruction, but does not contain a zero byte in its binary representation.

There are two final details that must be considered by the attacker before he launches a stack smashing attack. First among these is the question of what value to overwrite the return address with. At first blush the answer is obvious: the attacker must overwrite the return address with the address that holds the first instruction of his malicious code. Since this code was placed directly into the overflowed buffer, this address is simply the address of the first byte in the overflowed buffer. However, this buffer is stored on the stack, which grows and shrinks dynamically. If the programmer is not intimately familiar with the source code of the program he is trying to smash, he may not have a clear mental picture of the stack he is trying to smash. How then can he determine the address at which the target buffer begins? How can he find the needle buried in the haystack?

One obvious tact that the attacker might take is to simply use trial and error. He knows that the stack grows from high addresses to low addresses, and he knows the address from which the stack started growing (this is a constant across all programs). The attacker also knows that the average program rarely pushes more than a few thousand bytes onto the stack. Thus the attacker knows the address of the target buffer within at most a few thousand bytes and could therefore guess the buffer's actual address within a few thousand attempts.

It turns out that there is a more clever approach. If the attacker front pads his malicious code with NOP instructions he can greatly increase his chances of locating the target buffer's address. The following example illustrates why this is true: Suppose that the buffer we want to overflow is 512 bytes and that the malicious code we want to have run is just 100 bytes. Instead of putting the malicious code at the beginning of the buffer, we first load the first 412 bytes of the buffer with NOPs.[25] Next we copy our malicious code into the end of the buffer. Finally, we overflow the remainder of the buffer with enough extra bytes so that the return address is overwritten with our guess of where the buffer will end up on the stack.

[25] NOP stands for No Operation. Not surprisingly, the NOP instruction does nothing to change the state of the system. Its only use is as a placeholder in pipelined architectures.

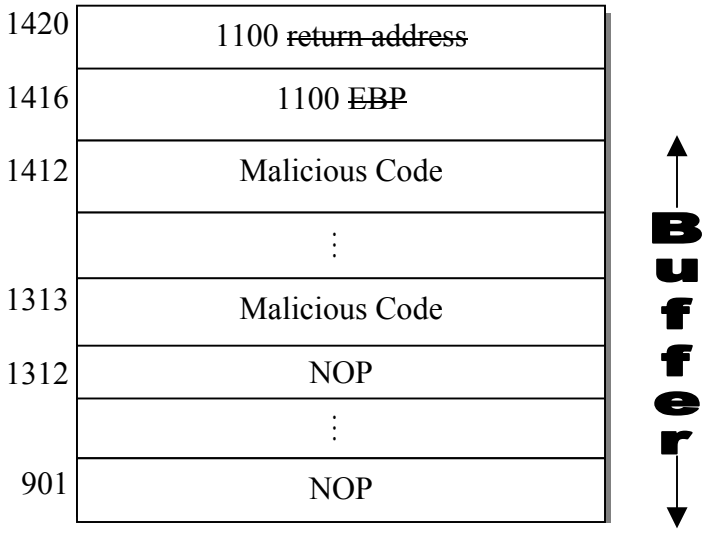

Figure 5. Depiction of the stack during a stack smashing attack that uses NOPs for padding. Addresses 901 to 1412 store the overflowed buffer. The first 412 bytes of the buffer contain NOP instructions. The remaining 100 bytes of the buffer contain the attacker's malicious code. The EBP and the return address have been overwritten with the attacker's guess of the buffer's beginning address—1100.

Figure 5 illustrates why the NOPs are so helpful to the attacker. Suppose that the attacker guesses that the buffer will be stored on the stack at address 1100 but that the buffer is in fact stored at address 901. Since the attacker has front loaded the buffer with 412 bytes of NOPs, addresses from 901 to 1312 will contain a trail of NOPs leading straight to the first byte of the attacker's malicious code. Thus, when the program jumps to the overwritten return address of 1100, it will simply execute 212 NOPs before executing the malicious code. A small price to pay for root access! The 412 NOPs have made the attacker's guessing 412 times more effective since any guess between address 901 and 1313 will result in his malicious code running successfully. In general, the more NOPs the attacker can fit in the target buffer, the easier his stack smashing attack will be.

This leads us to the final detail the attacker must take into consideration—what if the buffer he wants to overflow is too small to fit his malicious code, let alone any NOPs? The attacker is still not necessarily foiled. If the attacker is able to set environment variables on the machine he is attacking, he can stuff the NOPs and the malicious code into an environment variable.[26] Since environment variables are also stored on the stack,

[26] For example, the attacker could write a program that used `setenv` to set an environment variable, say `smasher`, to be his malicious code front-padded with NOPs. This environment variable would be stored at some address on the stack. The attacker would then overflow the buffer of his selected stack smashable program with the guessed address of the environment variable.

the attacker's goal is now to overwrite the return address with the address of this environment variable rather than the address of the target buffer. The beauty of this approach is that environment variables are not limited in size, allowing the attacker to front pad it with a large number of NOP instructions.

4.3 IP Address Spoofing [1]

4.3.1 Overview

Any system that uses IP addresses as a means of authentication can be compromised by an attack known as *IP address spoofing*. Spoofing allows an attacker to exploit a trust relationship between two machines, A and B, in order to gain unauthorized access to A.[27]

The idea behind the spoofing attack is the following. Suppose that some form of trust relationship exists between machines A and B and suppose that A and B authenticate themselves to each other solely by way of their IP addresses. That is, if A receives a packet whose source address is B's IP address, then A blindly believes that this packet must have originated from machine B. In an environment like this, an attacker is able to impersonate machine B in order to gain unauthorized access (usually root access) to machine A.

Here is a brief overview of the attack:

Outline of a IP Spoofing Attack

Step 1) The attacker initiates a denial of service attack on machine B.
Step 2) The attacker, masquerading as B, opens a connection to A.
Step 3) The attacker, still masquerading as B, takes advantage of the trust relationship between A and B by installing a back door on A.
Step 4) The attacker ends the denial of service attack on machine B.

This attack presents two challenges to the attacker. The first challenge is that the attacker must be able to create packets with forged source addresses. Assuming, as we will, that the attacker is root on the machine from which he is attacking, this poses no difficulty to the attacker. Whether by writing his own utility or by compiling a custom kernel, a respectable hacker should not have trouble creating forged packets.

The second challenge presents a more interesting problem for the attacker to solve. In Step 2 above, the attacker must open the connection to A *blindly*. That is, because the attacker is masquerading as B when it tries to open the connection to A, A's response

[27] Of course, the attacker could also use A to gain unpriveleged access to B. The steps would be identical with only the roles of A and B reversed.

packet will get sent to B rather than to the attacker's machine. Let us look carefully at the packets that are exchanged in Step 2 to see why this "blindness" presents a difficulty for the attacker.

Step 2 Expanded

M(B) → A: $SYN(ISN_{M(B)})$.
A → M(B): $ACK(ISN_{M(B)} + 1)$. $SYN(ISN_A)$
M(B) → A: $ACK(ISN_A + 1)$.

Here we have written M(B) to mean that the attacking machine, M, is impersonating machine B by sending forged IP packets. "SYN" and "ACK" stand for their respective TCP flags, and "ISN" stands for "Initial Sequence Number."

Opening a TCP connection between M(B) and A requires the two machines to synchronize sequence numbers by way of the usual three-way handshake. However, note that the second message, which contains A's initial sequence number, does not arrive at M but at B (where it is immediately dropped because of the denial of service attack that was initiated in step 1). How then can M(B) complete the three-way handshake with A if it does not know what value to ACK?

This is the main challenge facing the attacker, and he overcomes it by investigating how machine A's sequence numbers change over time. This is as easy as periodically opening a telnet connection from M to A and inspecting the sequence numbers that A sends to M during the three-way handshake.[28] In most cases a simple pattern will emerge that will allow the attacker to predict, with high probability, the sequence numbers that A will use in the future.

As a concrete example, suppose A was running the BSD operating system. BSD simply increments its sequence number by 128,000 each second and by 64,000 after each connection is made. It would not take too many probes before the attacker recognized this pattern and was able to predict future sequence numbers with high probability.

Armed with this information about A's sequence numbers, the attacker is able to complete the three-way handshake in Step 2 after no more than a modest number of attempts. He is, at last, in a position to exploit the trust relationship that exists between machines A and B. Machine A believes that all the commands it receives at this point are coming from the trusted machine B. The attacker completes the attack by using B's trusted status to install a backdoor on A for later use. Now, the next time the attacker wants to gain access to A he will not need to worry about sequence numbers at all.

[28] We assume that the attack is being launched from a compromised machine, M, so that there is no need to protect M's identity in this exchange.

4.3.2 Common Attack Scenario

In the Unix world, the "r" commands represent the most common trust relationship that can be exploited by IP address spoofing. These commands include `rlogin`, `rsh`, `rcp`, etc. By impersonating some trusted machine B, an attacker is able to remotely run all of these commands on machine A. Moreover, since the attacker is almost certainly the root user on the machine from which he is attacking, he will appear to A to be the root user on machine B—this means that all of the commands that the attacker runs remotely on A will execute with root privilege on A!

The most typical command that the attacker will run on A is the following:

```
cat + + >> ~/.rhosts
```

This creates a backdoor on machine A by instructing A to trust the root user on any machine whatsoever, not just on machine B. The primary advantages of this command are that it is *non-interactive* (so that the attacker can completely disregard A's response to the command) and *short* (so that the attacker can end the spoofing phase of the attack immediately after executing it).

This very attack was launched by Kevin Mitnick in 1994 on Christmas day [15]. Unfortunately for Mr. Mitnick, his chosen victim, Tsutomu Shimomura, was a security expert. Dr. Shimomura's machines used the packet-sniffing program, `tcpdump`, to record each step of Mitnick's spoofing attack. The attack became part of the CERT database less than a month after it first occurred.

5 Post Compromise

After a system has been compromised, the potential for damage is significant. A hacker will have access to the internal network and will attempt to compromise internal hosts. Internal information is exposed and the hacker simply needs to use some readily available hacking tools to uncover network data.

5.1 Rootkits

Usually the first step after gaining root access on a machine is to upload a rootkit.[29] A rootkit is a group of utilities that help setup backdoors on the rooted machine for future access and facilitate the gathering of internal network information. Rootkits typically include Trojans, sniffers, and system log cleaners [14].

[29] The term rootkit is derived from utilities used after gaining 'root' access on UNIX hosts. Rootkits are not exclusively used for UNIX hacking and can be found for any type of operating system.

5.1.1 Trojans [14]

Trojans are customized versions of system utilities that the hacker uses to gain access to the system at a later time, to steal passwords, or to hide the fact that malicious programs are running. An example would be a custom version of `telnetd`. `telnetd` is a UNIX-based daemon that controls telnet connections to the host. A hacker can alter `telnetd` so that when a terminal type is set to a special type (known to the hacker) access is granted without authentication [9]. This Trojan is particularly useful if an administrator changes passwords frequently. Another example of a Trojan would be to install a modified version of the `ps` command that would not report that any of the attacker software is running on the compromised system.

While Trojans are useful to the hacker, security conscious network administrators can detect them. There are administrative utilities like `md5sum` that can compute a checksum of executables on a host. These checksums can be stored securely and checked against the current executables at regular time intervals to detect host compromise.

5.1.2 Packet Sniffing

Packet sniffers can only be used on machines on which the hacker has root access. Packet sniffers are powerful tools that, in many cases, allow the attacker to capture login names, passwords, and sensitive data that is transferred by any machine that is on the same local network as the compromised machine.

Sniffing attacks are most commonly launched from a computer that is connected to a shared Ethernet network, although they can also be launched from switched Ethernet networks as we will see. There is no reason why other types of local area networks cannot be sniffed, but we will focus our discussion on Ethernet networks since they are by far the most prevalent.

5.1.2.1 Sniffing Shared Ethernet Networks [6]

To understand how Ethernet networks are sniffed, it is important to first understand their basic architecture. In a *shared* Ethernet network, every machine shares the same physical wire for sending and receiving data. Every machine is connected to this wire by an Ethernet adapter, and each Ethernet adapter is uniquely identified by a factory-assigned, 48-bit number referred to as its Media Access Control, or MAC, address.

When packets show up on the shared wire, it is the job of each machine's Ethernet adapter to check whether the packet is addressed to its own MAC address. If it is, the adapter copies the packet into memory. The kernel then passes the packet up the appropriate network protocol stack (typically the TCP/IP stack) on its way to its final destination (e.g., a telnet process).

If the packet's destination address does not match the Ethernet adapter's MAC address, the adapter usually just ignores the packet. However, if the adapter is in "promiscuous" mode, it copies *every* packet it sees on the shared wire into memory. It does not matter whether the packet's destination address matches the adapter's MAC address or not—the packet will be unconditionally passed to the kernel and on to any application that is set up to handle it (including the attacker's sniffer program). It does require root access to put an Ethernet adapter in promiscuous mode, but we're assuming that the attacker already has root access.

The attacker can now see all the traffic that passes on the shared Ethernet. Most of this traffic is of no interest to the attacker, but some of the packets will contain login names and passwords that the attacker may be very interested in. For example, if a user on the shared Ethernet opens a telnet session to any host whatsoever, the attacker will see this user's login name and password since they are both sent in clear text. The same is true for programs like FTP. Encryption of login names and passwords is not necessarily enough to thwart the attacker either. If the authentication scheme being used is susceptible to replay attacks, then the attacker can just as easily impersonate the user with the encrypted forms of the login name and password. If the attacker is after data then encryption provides more security assuming the cryptographic algorithm is secure.

One approach a system administrator might take to fight sniffing is to install Ethernet adapters that do not support promiscuous mode. Such Ethernet adapters do exist, but they are not easy to find anymore. Moreover, the PC99 standard established by Intel and Microsoft requires that all Ethernet adapters support promiscuous mode [13].

Instead, the usual tact taken by a system administrator concerned about sniffing is to move from a shared Ethernet network to a *switched* Ethernet network. In a switched network the packet's destination address is compared to each machine's MAC address only at the switch, rather than at every single machine's Ethernet adapter. When the switch determines that the incoming packet's address matches machine A's MAC address, the switch puts the packet on a line dedicated solely to A. This means that there is no single machine from which the attacker can sniff the entire network's traffic—if the sniffer is installed on machine A, then the attacker will only see the packets sent from and received by A. Of course, some machine's traffic will be a lot more interesting to the attacker than others. Any server, for instance, will still receive numerous login packets that the attacker could capture were he able to install a sniffer on the server. Presumably servers will be more carefully guarded and better administered than the average user's machine though.

5.1.2.2 Sniffing Switched Ethernet Networks [6]

Switched networks are not a silver bullet for sniffing attacks. Although switched networks do thwart most sniffing attacks, advanced attackers are often able to collect the same information from a switched network that they would have collected from a shared network. We will discuss a few different ways this is possible in the following paragraphs.

One attack against switched networks exploits the switches themselves. When a switch receives a large number of packets destined for MAC addresses that it does not recognize, the switch may become concerned that it has been improperly configured. The switch worries that these MAC addresses are in fact on the local network and that its current list of address mappings is incomplete. The switch responds by invalidating its entire list of address mappings and sending all packets to all machines on the local network (in the lingo, the switch changes from "bridging" mode to "repeating" mode). If these unrecognized MAC addresses were destined for a machine unknown to the switch, this machine will now receive these packets and be able to use the local network.

However, it is easy to see how this "fail open" behavior of a switch can be exploited by the sniffing attacker. The attacker, based from a machine on the local network, simply sends out a wave of packets destined for MAC addresses that are not present on the local network. The switch then panics and jumps from bridging mode to repeating mode, effectively transforming the switched network into a shared network. The attacker can now go about sniffing in the usual manner, with access to every packet that is sent or received on the local network.

Another novel switch exploit is to remotely configure a "monitor" port on the switch. Every packet that crosses the switch is copied to the monitor port, regardless of its ultimate destination. Switches support monitor ports so legitimate sniffing can be carried out by system administrators. Administrators may simply log these packets to disk, or they may filter these packets on the fly looking for suspicious activity. If an attacker is able to set up a monitor port for himself, he has once again transformed a switched network into an effectively shared network.

Setting up a monitor port on a switch requires that the attacker overcome two difficulties. First, setup will require that the attacker know the password on the switch being attacked. Quite often these passwords are just the factory default: "public" or "private." Second, the switch will often be configured to accept commands from only a specific IP address. As we have seen, though, authentication schemes based only on IP addresses can be compromised by a spoofing attack. The attacker needs only to sniff a packet that has been sent from the IP address that is trusted by the switch. The attacker will then know which IP address to spoof in order to take control of the switch. These packets can be easily identified since they will contain Simple Network Management Protocol (SNMP) messages.[30]

Some switches also have an "auto-learning" feature that can be used by the attacker to impersonate a particular machine on the local network. Suppose the attacker begins sending out packets whose source addresses are the target machine's MAC address. The switch will soon decide that the attacker is in fact the target and begin forwarding all packets destined for the target machine to the attacker's machine. What if the target is also sending packets to the switch with its own MAC address? In this case the switch

[30] SNMP is the "language" in which commands are sent to network devices like switches.

would auto-learn the attacker's forged MAC address for only brief interval before reverting back to the target's true MAC address. If the attacker is only interested in a sampling of the traffic that is being sent to the target, this might be acceptable.[31] However, if sampling the traffic is not adequate then the attacker must mount a denial of service attack against the target in order to minimize the chances of the switch reverting.

The attacker can also sniff a switched Ethernet network by taking advantage of the ARP protocol. ARP is a protocol used by all Ethernet networks to bind IP addresses to MAC addresses. For example, suppose Kevin and John are working on two machines on the same Ethernet network and that Kevin would like to send John a packet. Let us also suppose that Kevin knows John's IP address but that Kevin does not know John's MAC address. To find John's MAC address Kevin must use the ARP protocol.

Kevin broadcasts an ARP query to every machine on the local Ethernet network. This query effectively asks each machine the following question: If you have John's IP address, what is your MAC address? To reduce traffic on the network, the query also contains a source field that includes Kevin's IP address and MAC address. This way, when John replies to Kevin, he will not need to initiate another ARP query to determine Kevin's MAC address. In fact, every machine on the local network takes note of the source IP address and MAC address pair that is contained in the query. Even though they do not respond to Kevin's ARP query, they cache the binding between Kevin's IP address and MAC address for future use.

There are two aspects of the ARP protocol that make it attractive to a sniffing attacker. First, whether the network is switched Ethernet or not, ARP packets are broadcast to all machines connected to the network. Second, machines blindly cache the contents of ARP queries without first validating them. Using both these facts, the attacker is able to make global changes to the network topology and thereby sniff the packets he is interested in.

For example, suppose the attacker broadcasts an ARP query inquiring about a non-existent IP address. No machine will respond to this query. However, if this query's source field contains the local router's IP address and the attacker's MAC address, each machine on the network will blindly cache this information in its ARP table.[32] Now, whenever one these machines sends traffic to the router's IP address, the machine will unwittingly direct the traffic to the MAC address of the attacker's machine. It is the attacker's job to then forward this traffic on to the router so as not to bring the network to a conspicuous standstill.

[31] Assuming the traffic was being sent over a reliable connection, there would be no overt signs that the attacker was sampling the target's traffic. Every packet would still arrive at the target's machine, although there would be a slight lag in the delivery of the sampled packets.

[32] In fact, this same attack could be waged against any single machine on the local network by inserting their IP address in the source field of the ARP query. Here we use the router as an example since it often receives the most sniff-worthy traffic.

Two ICMP messages provide similar fodder for the attacker. For one, the attacker might send the target machine an ICMP redirect packet. This packet would inform the target machine that any packet that it would have ordinarily sent to machine `X` should instead be redirected to the attacker's machine. For another, the attacker might send the target machine an ICMP router advertisement claiming that the attacker's machine is the router. In both cases, the router will be now able to sniff the target machine's packets.

5.1.2.3 Countermeasures and Counter-Countermeasures [6]

We conclude our discussion of packet sniffing with a brief discussion of what attackers do to keep their sniffing attacks from being detected. Thinking as a network administrator, the easiest way to check whether someone is sniffing on our network is to check whether some machine's network adapter is in promiscuous mode. In Unix the programs `ifconfig` and `netstat` are installed by default on most systems and will report whether or not an adapter is in promiscuous mode. There are other less common programs like `cpm` (which stands for "check promiscuous mode") that provide the same information [5]. The wiliest of attackers knows about all of these programs, however, and he will replace them with trojans that never report an adapter as being in promiscuous mode. This requires changing the source code, recompiling the programs, and possibly even recompiling the kernel on the target machine. In cases where the attacker is not able to replace these programs with imitations, he will at least disable them in order to buy himself some time.

Network administers worth their lofty IT salaries will also use the following technique to ferret out packet sniffers on a shared Ethernet network. To check whether a machine with a particular IP address, say $IP_{suspect}$, is sniffing the network, a network administer creates a ping request packet.[33] At the IP layer this ping packet is addressed to $IP_{suspect}$, but at the MAC layer this packet is addressed to a non-existent MAC address. Therefore when this packet is put on the shared Ethernet wire, no machine *should* reply to it since the packet's destination MAC address will not match the MAC address of any machine's Ethernet adapter. However, if the suspected machine's Ethernet adapter is in promiscuous mode, it will indiscriminately snatch up the packet and pass it up the protocol stack to the IP layer. Since the IP address of the suspect machine will match the IP address in the packet, the IP layer will send a ping reply back to the network administrator. The sniffer is exposed. Not to be outdone, attacker's have become wise to this and have begun to institute address filters in software. These filters prevent the sniffing computer from replying to baited packets.

5.1.3 Log File Cleanup [14]

Hackers do not want to provide the attacked site with information regarding the nature of their activities. Thus, log file cleanup is an important last step in the hacking process.

[33] There is nothing special about ping, anything that generates a response will do. Indeed a similar ferreting technique uses ARP queries: create an ARP query asking about $IP_{suspect}$'s MAC address and send it to a non-existent MAC address. If you get a response, $IP_{suspect}$ is sniffing the network.

A number of log cleaners are available in standard rootkits, but often a text editor will be sufficient. The `/etc/syslog.conf` offers a wealth of information regarding the location of system logs. This is a simple text file that usually contains explicit comments regarding the nature and location of the log files. Any log file that is in text format can be altered via a text editor. For binary logs, rootkit tools such as `wzap` [19] are needed.

Unfortunately, it is difficult to protect log files against cleanup. Often there is evidence that a log file has been modified, but there is not enough information to link back to the hacker. Log cleaning can erase the record of the attack and the hacker can disappear back into cyberspace.

There do exist some crafty system administrators who use a clever technique to protect their log files [5]. The basic idea is to use Ethernet adapters whose outputs have been disabled. All log updates are sent to the machine that has the modified adapter and then saved to its local disk. Since an attacker cannot receive information about this machine over the network, it is very difficult if not impossible for him to modify the logs to his advantage.

6 Putting It All Together: The Trinoo Attack [3]

Up to this point we have seen many techniques that hackers use to target and gain access to systems. Any full-fledged attack will use a combination of these techniques. One such attack is the *distributed denial-of-service attack* or DDOS. DDOS attacks render a target host useless by flooding it with maliciously derived traffic from many attacking machines. These attacks have gained media attention in recent years with major Internet sites like Yahoo! and eBay having been targeted and temporarily shut down. We will look at the Trinoo attack as an example of a DDOS attack.

Trinoo is software that creates a network of *master* and *daemon* machines to launch a DDOS attack.[34] A typical attack scenario consists of the following stages:

1. The attacker uploads the following to a root compromised machine: scanning tools, the Trinoo master program, sniffing software, and rootkit software. This machine can be compromised via the methods described in section 4.1 and 4.2. This machine, which is referred to as the Trinoo master, coordinates the Trinoo attack.
2. A horizontal scan is performed over a wide range of network addresses to identify hosts that have an exploitable service such as `wu-ftpd` on port 21. The scan can be implemented using the tools and methods described in section 3.3.
3. A script is constructed and executed that carries out the exploit on all the vulnerable machines. These exploits commonly take advantage of stack smashing vulnerabilities as described in section 4.2.

[34] The source code for Trinoo is available for free download from http://packetstorm.securify.com/distributed/

4. A second script is constructed and executed that installs the Trinoo daemon software on these compromised machines. These machines are referred to as Trinoo zombies.
5. A rootkit is installed on each machine to cover that attacker's tracks so that a system administrator will not detect that the machine has been compromised, as described in section 5.1.1.
6. The Trinoo master initiates the DDOS attack by sending a TCP message to each of the Trinoo zombies. When the zombies receive this message, they each begin sending a barrage of UDP packets to random ports on the targeted victim's machine.
7. The target machine is rendered useless because it cannot handle the flood of incoming UDP packets.

DDOS attacks present a significant threat to machines connected to the Internet. These attacks are difficult to stop because of their distributed nature. Discarding packets from one IP address will not significantly stem the flow of incoming packets. The only solution is to limit the rate of all incoming packets at a firewall or border router. In addition, the attackers are difficult to track down. The attack is launched from compromised machines, so the attacker of these machines must be found. With effective rootkit log cleaners (as described in section 5.1.3), the hacker can often avoid detection.

7 Conclusion

In this paper we have presented some of the most common tools and techniques that hackers use to compromise computer systems. These include network enumeration and scanning, fingerprinting, password cracking, stack smashing, spoofing, and sniffing. Without understanding these basic attack building blocks, it is not possible to effectively defend against the many attacks (like the Trinoo attack) that use them. Our hope is that this paper provides this understanding.

References

1. daemon9. *Phrack*. Issue 48, Article 14. http://www.phrack.com.

2. Designer, Solar. *John The Ripper*. http://www.openwall.com/john.

3. Dittrich, David. *The Trinoo Attack.* http://staff.washington.edu/dittrich/misc/ddos.

4. Fyodor. http://www.insecure.org/nmap.

5. Gillis, G. "Packet Sniffing," *DEFCON*, 1997. http://www.angelfire.com/electronic/ackka/docs/20.htm.

6. Graham, Robert. *Sniffing (Network Wiretap, Sniffer) FAQ*. http://www.robertgraham.com/pubs/sniffing-faq.html.

7. Hackers Club. http://www.hackersclub.com.

8. Harold, Elliotte R. Java Network Programming. O'Reilly, Sebastopol, CA, 1997. pp. 1-233.

9. Klaus, Christopher. *Backdoors*, 1997. http://rootshell.com/docs/backdoors.txt.

10. Muffet, Alec. *Crack*. http://www.users.dircon.co.uk/~crypto.

11. Northcutt, Stephen. Network Intrusion Detection: An Analyst's Handbook. New Riders, Indianapolis, IN, 1999. pp. 1-105, 241-254.

12. One, Aleph. *Phrack*. Issue 49, Article 14. http://www.phrack.com.

13. PC 99 Design Guide. http://www.pcdesguide.com/default.htm.

14. Scambray, Joel, S. McClure, G. Kurtz. Hacking Exposed: Second Edition. McGraw-Hill, Berkeley, CA, 2001.

15. Shimomura, Tsutomu. *Mittnick Attack Account.* http://www.networkcomputing.com/unixworld/security/001.add.html.

16. Snort. http://www.snort.org.

17. SRI International. http://www.sri.com.

18. Standiford, Stuart, J. Hoagland, J.M. McAlerney, "Practical Automated Detection of Stealthy Portscans". *ACM CCS IDS Workshop,* November 1, 2000. http://www.silicondefense.com/research/pubs.htm.

19. Wzap. http://member.nbci.com/unixug/progs.