## Intrusion Detection Attacks Database

These are attacks that were used as a part of: The 1998 ID evaluation, The 1999 ID Evaluation training data, and The 1999 ID Evaluation test data.

Attacks considered to be New in 1999 are those that did not appear in 1998 or the 1999 training data and are denoted as such.

Many of these attack descriptions are taken directly from Kris Kendall's MIT Master's thesis ( available on the publications page ). Thus there are many references included in the text, which we have included at the bottom of this page. For more details on the notation and terminology used below, please refer to the thesis.

### **Contents**

1. [Denial of Service Attacks](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "dos)

* [Apache2](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "apache)
* [arppoison](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "arppoison) (New in 1999 test)
* [Back](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "back)
* [Crashiis](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "crashiis)
* [dosnuke](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "dosnuke) (New in 1999 test)
* [Land](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "land)
* [Mailbomb](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "mailbomb)
* [SYN Flood](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "syn) (Neptune)
* [Ping of Death](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "pod) (POD)
* [Process Table](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "pt)
* [selfping](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "selfping) (New in 1999 test)
* [Smurf](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "smurf)
* [sshprocesstable](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "sshprocesstable) (New in 1999 test)
* [Syslogd](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "syslogd)
* [tcpreset](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "tcpreset) (New in 1999 test)
* [Teardrop](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "teardrop)
* [Udpstorm](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "udpstorm)

2. [User to Root Attacks](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "u2r)

* [anypw](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "anypw) (New in 1999 test)
* [casesen](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "casesen) (New in 1999 test)
* [Eject](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "eject)
* [Ffbconfig](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "ffb)
* [Fdformat](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "fdf)
* [Loadmodule](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "loadmod)
* [ntfsdos](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "ntfsdos) (New in 1999 test)
* [Perl](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "perl)
* [Ps](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "ps)
* [sechole](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "sechole) (New in 1999 test)
* [Xterm](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "xterm)
* [yaga](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "yaga) (New in 1999 test)

3. [Remote to Local Attacks](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "r2l)

* [Dictionary](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "dict)
* [Ftpwrite](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "ftp)
* [Guest](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "guest)
* [Httptunnel](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "httptunnel)
* [Imap](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "imap)
* [Named](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "named)
* [ncftp](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "ncftp) (New in 1999 test)
* [netbus](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "netbus) (New in 1999 test)
* [netcat](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "netcat) (New in 1999 test)
* [Phf](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "phf)
* [ppmacro](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "ppmacro) (New in 1999 test)
* [Sendmail](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "sendmail)
* [sshtrojan](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "sshtrojan) (New in 1999 test)
* [Xlock](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "xlock)
* [Xsnoop](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "xsnoop)

4. [Probes](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "probes)

* [insidesniffer](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "insidesniffer) (New in 1999 test)
* [Ipsweep](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "ipsweep)
* [ls\_domain](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "ls_domain) (New in 1999 test)
* [Mscan](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "mscan)
* [NTinfoscan](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "ntinfoscan)
* [Nmap](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "nmap)
* [queso](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "queso) (New in 1999 test)
* [resetscan](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "resetscan) (New in 1999 test)
* [Saint](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "saint)
* [Satan](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "satan)

5. [Data](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "data)

* [Secret](https://archive.ll.mit.edu/ideval/docs/attackDB.html" \l "secret)

### **Denial of Service Attacks**

A denial of service attack is an attack in which the attacker makes some computing or memory resource too busy or too full to handle legitimate requests, or denies legitimate users access to a machine. There are many varieties of denial of service (or DoS) attacks. Some DoS attacks (like a mailbomb, neptune, or smurf attack) abuse a perfectly legitimate feature. Others (teardrop, Ping of Death) create malformed packets that confuse the TCP/IP stack of the machine that is trying to reconstruct the packet. Still others (apache2, back, syslogd) take advantage of bugs in a particular network daemon. Figure 6-1 provides an overview of the denial of service attacks used in the 1998 DARPA intrusion detection evaluation. Each row represents a single type of attack. The six columns show the attack name, a list of the services that the attack exploits, the platforms that are vulnerable to the attack, the type of mechanism that is exploited by the attack (implementation bug, abuse of feature, masquerading, or misconfiguration), a generalization of the amount of time the attack took to implement, and a summary of the effect of the attack. The following sections describe in detail each of the Denial of Service attacks that were included in the 1998 DARPA intrusion detection evaluation.

|  |  |
| --- | --- |
| **Apache2**  R-a-Deny (Temp./Admin.) | **Description** The Apache2 attack is a denial of service attack against an apache web server where a client sends a request with many http headers. If the server receives many of these requests it will slow down, and may eventually crash [4].  **Simulation Details** This exploit was adapted from C code originally posted to the bugtraq mailing list. A C- shell wrapper was also created which executes the apache2 C program in a loop until the server being attacked is no longer responsive. As soon as the attack was launched the load average (as reported by the 'top' program) of the victim server jumped to 5 or more. As more and more requests were submitted to the web server the memory usage and load average of the victim continued to climb until eventually the httpd daemon ran out of memory and crashed. At this point the server no longer responded to http requests and the httpd daemon needed to be restarted by the superuser for service to be restored.  **Attack Signature** Every http request submitted as part of this exploit contains many http headers. Although the exact number and value of these headers could be varied by an attacker, the particular version of the exploit which was used in the 1998 DARPA evaluation sent http GET requests with the header 'User-Agent: sioux\r\n" repeated 10000 times in each request. The actual content of the header is not important for the exploitùthe exploit is only dependent on the fact that http request contains many headers. A typical http request contains twenty or fewer headers, so the 10000 headers used by this exploit are quite anomalous. |
| **arppoison**  P-a-Deny (Temp.) | **Description** ARP Poison is a Denial of Service attack that was developed specifically for the 1999 MIT-LL Evaluation. In this attack the goal is to trick hosts on the same ethernet into "learning" the wrong "Mac" address for known IP addresses. The attacker must have access to the Local Area Network.  **Simulation Details:** The attack is carried out by running a binary program that takes the victim IP address as a commandline parameter. While running, at root-level, the attack listens on the network for "arp who-has" requests, to which it responds, as quickly as possible, with a bogus machine-level address. Thus, packets destined for the victim are "re-routed" at the mac-level and connections/packets to the victim are disrupted. As more than one machine might respond to a given arp request, the attack is not 100% effective, during any attack it is possible that one or more hosts on the ethernet have cached the correct mac address for a given ip and are not tricked into storing the bogus one.  During the 1999 Lincoln Evaluation, the attacks were manually verified to ensure that atleast a few connections to the victim were disrupted as a result.  **Attack Signature:** The attack could be detected by analyzing the ARP protocol, and observing that for a given arp who-has request the machine performing the attack consistantly responds with the wrong (often completely bogus) machine-level address. |
| **Back**  R-a-Deny (Temporary) | **Description** In this denial of service attack against the Apache web server, an attacker submits requests with URL's containing many front slashes. As the server tries to process these requests it will slow down and becomes unable to process other requests [55].  **Simulation Details** The Back attack was implemented as a C shell script that used the Netcat [31] tool to generate network traffic. This shell script was adapted from a script originally posted to the Bugtraq mailing list. Although the number of front slashes in the URL sent by the shell script could be varied, the number of front slashes that was determined to be optimal for denial of service against Apache running on Linux 4.2 was between six and seven thousand. The Back attack causes instances of the httpd process on the victim to consume excessive CPU time. This consumption of the CPU slows down all the system's activities, including responses to network requests. The system recovers automatically when the attack stops.  **Attack Signature** An intrusion detection system looking for the Back attack needs to know that requests for documents with more than some number of front slashes in the URL should be considered an attack. Certainly, a request with 100 front slashes in the URL would be highly irregular on most systems. This threshold could be varied to find the desired balance between detection rate and false alarm rate. |
| **Crashiis**  R--Deny | **Description** CrashIIS is a Denial of Service attack against the NT IIS web-server. The attacker sends a malformed GET request via telnet to port 80 on the NT victim. The command "GET ../.." crashes the web server and sometimes crashes the ftp and gopher daemons as well, because they are part of IIS.  View the page from NTSecurity.net for more information.  **Simulation details** A fully automated Expect script on a Unix attacker telnets to port 80 on the NT victim and sends the command "GET ../..". Running "crashiis.exp 172.16.112.100" will crash Hume's webserver (and possibly ftp and gopher as well).  **Verification** After the attack has completed, the IIS Web server on the victim should be down: - Type the command "telnet hume.eyrie.af.mil 80" - it should no longer connect. - Use a browser to access a page on the server (e.g. http://hume.eyrie.af.mil) - it should not load the page.  **Cleanup** Somebody must manually restart the victim's webserver. Sometimes the ftp and gopher services need to be restarted as well.  **Attack signature** Sniffing the network traffic will reveal the malformed GET command. The victim's security audit log will show that Dr. Watson ran when the service(s) crashed. However, Dr. Watson will also run for other reasons. Therefore, using this audit signature for detection will most likely result in false alarms. |
| **dosnuke**  R-b-Deny (Temp./Admin.) | **Description** DoSNuke is a Denial of Service attack that sends Out Of Band data (MSG\_OOB) to port 139 (NetBIOS), crashing the NT victim (bluescreens the machine).  **Simulation details** A Perl script, dosnuke.pl, runs on an NT attacker. Open dosnuke.pl for editing and set the time of day to run the attack. Then run dosnuke.pl or place it in the startup group. The script takes no arguments (always attacks Hume 172.16.112.100).  **Verification** The victim machine will display a bluescreen. To remotely verify the success of the attack, try pinging 172.16.112.100. There should be no response from the victim.  **Attack signature** The attack creates a NetBIOS connection. The packets are flagged "urg" because of the MSG\_OOB flag. The attack can be detected by searching the sniffed data for a NetBIOS handshake followed by NetBIOS packets with the "urg" flag.  The victim's audit logs will indicate a reboot after the system is restarted. Most likely, the attacker had to hard reboot the machine (physically press the reset button) because he did not have a password to login or unlock the machine. A soft reboot audit signature is a "SeShutPrivilege" Privilege Use Event followed by an event stating, "Windows NT is starting up." A hard reboot audit signature does not include the "SeShutdownPrivilege" event.  A hard reboot can be used to detect but not identify the DoSNuke attack, because other attacks may also result in hard reboots (NTFSDOS, AnyPW, etc.). In addition, a hard reboot may occur without an attack (power outages, system halts, etc).  Note: Originally, the attack sent the string "Hey, I can't help getting these nasty VXD errors!" It now sends a blank string instead. Other versions of the attack may still send the string. |
| **Land**  R-b-Deny (Administrative) | **Description** The Land attack is a denial of service attack that is effective against some older TCP/IP implementations. The only vulnerable platform used in the 1998 DARPA evaluation was SunOS4.1. The Land attack occurs when an attacker sends a spoofed SYN packet in which the source address is the same as the destination address [17].  **Simulation Details** The land exploit program used in the DARPA evaluation was adapted from a C implementation found at http://www.rootshell.com. The exploit is quite simple and the code could easily be rewritten in any language with access to the TCP sockets interface. The code sends a single SYN packet with the source address spoofed to be the same as the destination address. Within the simulation, this exploit was run against a Sun SPARC WorkStation running SunOS version 4.1. When a TCP SYN packet with an identical source and destination address was received by this host, the system completely locked up. In order to restore service, the machine had to be physically turned off and on again.  **Attack Signature** The Land attack is recognizable because IP packets with identical source and destination addresses should never exist on a properly working network. |
| **Mailbomb**  R-a-Deny (Administrative) | **Description** A Mailbomb is an attack in which the attacker sends many messages to a server, overflowing that server's mail queue and possible causing system failure.  **Simulation Details** This exploit was implemented as a perl program that constructed mail messages and connected to the SMTP port of the victim machine directly. The Mailbomb perl program accepted as parameters the e-mail addresses of victims as well as the number of e-mail messages to send. Although the Mailbomb exploit was used several times throughout the simulation with different parameters, a typical attack would send 10,000 one kilobyte messages (10 megabytes of total data) to a single user. This volume of messages was not enough to adversely effect the performance of the server or cause system failure. As implemented, this attack was more of a nuisance for a particular user than a real threat to the overall security of a server.  **Attack Signature** An intrusion detection system that is looking for a mailbomb attack can look for thousands of mail messages coming from or sent to a particular user within a short period of time. This identification is a somewhat subjective process. Each site might have a different definition of how many e-mail messages can be sent by one user or to one user before the messages are considered to be part of a mailbomb. |
| **SYN Flood (Neptune)**  R-a-Deny (Temporary) | **Description** A SYN Flood is a denial of service attack to which every TCP/IP implementation is vulnerable (to some degree). Each half-open TCP connection made to a machine causes the 'tcpd' server to add a record to the data structure that stores information describing all pending connections. This data structure is of finite size, and it can be made to overflow by intentionally creating too many partially-open connections. The half-open connections data structure on the victim server system will eventually fill and the system will be unable to accept any new incoming connections until the table is emptied out. Normally there is a timeout associated with a pending connection, so the half-open connections will eventually expire and the victim server system will recover. However, the attacking system can simply continue sending IP-spoofed packets requesting new connections faster than the victim system can expire the pending connections. In some cases, the system may exhaust memory, crash, or be rendered otherwise inoperative [13].  **Simulation Details** The neptune exploit code used in the simulation was compiled from C code originally posted to the bugtraq archive. The neptune program allows the user to specify a victim host, the source address to use in the spoofed packets, the number of packets to send, and the ports to hit on the victim machine (including an 'infinity' option that would attack all ports). The neptune exploit was effective against all three of the victim machines used in the simulation. Every TCP/IP implementation is vulnerable to this attack to a varying degree depending on the size of the data structure used to store incoming connections and the timeout value associated with half-open connections. As a point of reference, sending twenty SYN packets to a port on a Solaris 2.6 system will cause that port to drop incoming requests for approximately ten minutes. During the simulation, a neptune attack which sent 20 SYN packets to every port from 1 to 1024 of the Solaris server once every ten minutes was able block incoming connections to any of these ports for more than an hour.  **Attack Signature** A neptune attack can be distinguished from normal network traffic by looking for a number of simultaneous SYN packets destined for a particular machine that are coming from an unreachable host. A host-based intrusion detection system can monitor the size of the tcpd connection data structure and alert a user if this data structure nears its size limit. |
| **Ping Of Death**  R-b-Deny (Temporary) | **Description** The Ping of Death is a denial of service attack that affects many older operating systems. Although the adverse effects of a Ping of Death could not be duplicated on any victim systems used in the 1998 DARPA evaluation, it has been widely reported that some systems will react in an unpredictable fashion when receiving oversized IP packets. Possible reactions include crashing, freezing, and rebooting [14].  **Simulation Details** Several implementations of the Ping of Death exploit can be found at http://www.rootshell.com as well as many other sources on the web. This exploit is popular because early versions of the ping program distributed with Microsoft Windows95 would allow the user to create oversize ping packets simply by specifying a parameter at the command line (i.e. ping ûl 65510). Thus, many users could potentially exploit this bug without even making the effort to download and compile a program. The Ping of Death attack affected none of the victim systems used in the evaluation. The attack was included as an example of an attempted known attack that fails to have an effect.  **Attack Signature** An attempted Ping of Death can be identified by noting the size of all ICMP packets and flagging those that are longer than 64000 bytes. |
| **Process Table**  R-a-Deny (Temporary) | **Description** The Process Table attack is a novel denial-of-service attack that was specifically created for this evaluation. The Process Table attack can be waged against numerous network services on a variety of different UNIX systems. The attack is launched against network services which fork() or otherwise allocate a new process for each incoming TCP/IP connection. Although the standard UNIX operating system places limits on the number of processes that any one user may launch, there are no limits on the number of processes that the superuser can create, other than the hard limits imposed by the operating system. Since incoming TCP/IP connections are usually handled by servers that run as root, it is possible to completely fill a target machine's process table with multiple instantiations of network servers. Properly executed, this attack prevents any other command from being executed on the target machine. An example of a service that is vulnerable to this attack is the finger service. On most computers, finger is launched by inetd. The authors of inetd placed several checks into the program's source code that must be bypassed in order to initiate a successful process attack. In a typical implementation (specifics will vary depending on the actual UNIX version used), if inetd receives more than 40 connections to a particular service within 1 minute, that service is disabled for 10 minutes. The purpose of these checks was not to protect the server against a process table attack, but to protect the server against buggy code that might create many connections in rapid-fire sequence. To launch a successful process table attack against a computer running inetd and finger, the following sequence may be followed: 1. Open a connection to the target's finger port. 2. Wait for 4 seconds. 3. Repeat steps 1-2. This attack has been attempted against a variety of network services on a variety of operating systems. It is believed that the imap and sendmail servers are vulnerable. Most imap server software contains no checks for rapid-fire connections. Thus, it is possible to shut down a computer by opening multiple connections to the imap server in rapid succession. With sendmail the situation is reversed. Normally, sendmail will not accept connections after the system load has jumped above a predefined level. Thus, to initiate a successful sendmail attack it is necessary to open the connections very slowly, so that the process table keeps growing in size while the system load remains more or less constant [6].  **Simulation Details** The version of this exploit used in the simulation was implemented as a perl script that would open connections to a port every n seconds, where the port and the number of seconds n are specified as run-time parameters. The connections were maintained until a user terminated the script. The number of connections that must be opened before denial of service is accomplished depends on the size of the process table in the operating system of the victim machine. By using the Process Table attack on the fingerd port as described above, the process table of a Solaris server was exhausted after opening approximately 200 connections (at a rate of one connection every four seconds it took about 14 minutes before the process table was full). After the process table was full, no new process could be launched on the victim machine until the attack was terminated by the attacking program or an administrator manually killed the connections initiated by the attacking script (which is quite difficult to do without launching a new process).  **Attack Signature** Because this attack consists of abuse of a perfectly legal action, an intrusion detection system that is trying to detect a process table attack will need to use somewhat subjective criteria for identifying the attack. The only clue that such an attack is occurring is an 'unusually' large number of connections active on a particular port. Unfortunately 'unusual' is different for every host, but for most machines, hundreds of connections to the finger port would certainly constitute unusual behavior. |
| **selfping**  R-b-Deny (Temp.) | **Description** The selfping attack is a denial of service attack in which a normal user can remotely reboot a machine with a single ping command. This attack can be performed on Solaris 2.5 and 2.5.1.  **Simulation Details** The ping command broadcasts echo\_request packets using the localhost as the multicast interface. Within a couple seconds the system panics and reboots. There are two version of this attack in the 1999 DARPA evaluation. One version creates an atjob on the victim machine and then logouts. The other, more malicious version, creates a cronjob which reboots the machine every 5 minutes. The administrator must remove the cronjob in order to keep the machine from rebooting. This attack was origally posted on http://www.rootshell.com  **Attack Signature** The attack reboots the machine within 10 seconds of executing the ping command. The only signature seen in sniffing data is a ping to the broadcast interface just before the machine dies. A ping to the broadcast interface is a perfectly normal action. In the less harmful version, the atjob can be seen until it executes, whereas in the malicious version, the cronjob remains until it is deleted. |
| **Smurf**  R-a-Deny (Temporary) | **Description** In the "smurf" attack, attackers use ICMP echo request packets directed to IP broadcast addresses from remote locations to create a denial-of-service attack. There are three parties in these attacks: the attacker, the intermediary, and the victim (note that the intermediary can also be a victim) [18]. The attacker sends ICMP 'echo request' packets to the broadcast address (xxx.xxx.xxx.255) of many sub-nets with the source address spoofed to be that of the intended victim. Any machines that are listening on these subnets will respond by sending ICMP 'echo reply' packets to the victim. The smurf attack is effective because the attacker is able to use broadcast addresses to amplify what would otherwise be a rather innocuous ping flood. In the best case (from an attacker's point of view), the attacker can flood a victim with a volume of packets 255 times as great in magnitude as the attacker would be able to achieve without such amplification. This amplification effect is illustrated by Figure 6-2. The attacking machine (located on the left of the figure) sends a single spoofed packet to the broadcast address of some network, and every machine that is located on that network responds by sending a packet to the victim machine. Because there can be as many as 255 machines on an ethernet segment, the attacker can use this amplification to generate a flood of ping packets 255 times as great in size (in the best case) as would otherwise be possible. This figure is a simplification of the smurf attack. In an actual attack, the attacker sends a stream of icmp 'ECHO' requests to the broadcast address of many subnets, resulting in a large, continuous stream of 'ECHO' replies that flood the victim.  **Simulation Details** Because the simulation network for the 1998 DARPA evaluation has a flat network topology with only two physical subnets, the smurf attack as described above could not be implemented on the simulation network. For this reason, the 'smurfsim' program was developed to recreate the observable effects of a smurf attack. Smurfsim uses the raw socket API to construct ICMP packets with forged source addresses. Smurfsim takes as parameters the IP address of the victim, the number of packets to send, the average percentage of hosts on a subnet that are alive, and a comma-separated list of subnets. The program then randomly constructs a list of hosts that are alive on each of the subnets in the comma-separated list and starts sending 'echo reply' packets to the victim, that have been spoofed to look like they originating from the hosts in the list. This behavior is exactly what would occur if an attacker had performed an actual Smurf attack in which 'echo request' packets (with the source address spoofed to be that of the victim machine) were sent to the broadcast address of each subnet given in the parameter list. Several different simulated Smurf attacks were included in the evaluation data. In the most extreme case, the smurfsim program was used to simulate a smurf attack that generated traffic from 100 subnets for a period of one hour. During this period of time the entire simulation network was unresponsive and other network sessions (such as normal users trying to send e-mail, etc) would time out before they could be completed. In all, this particular attack instance generated over two gigabytes of network packets.  **Attack Signature** The Smurf attack can be identified by an intrusion detection system that notices that there are a large number of 'echo replies' being sent to a particular victim machine from many different places, but no 'echo requests' originating from the victim machine. |
| **sshprocesstable**  R-a-Deny (Temp.) | **Description** SSH Process-table is similar to the process-table attack in that the goal of the attacker is to cause sshd daemon on the victim to fork so many children that the victim can spawn no more processes. This is due to a kernel limit on the number of processes that the OS will allow.  **Simulation Details** The attack is in binary form, and takes the victim ip as a command-line argument. The attack works by making a number (serveral hundred, commonly) of connections to the victim via ssh, but does not complete the login process. If the sshd login timeout is long enough and the connections happen rapidly enough, the hosts process table will be exausted and no new process will be able to be spawned until others timeout or exit.  **Attack Signature** This attack will be evident due to the large number of rapid ssh connections to the host, the inability of processes to spawn on the host, and the fact that request for new network logins (requiring child processes) will be denied, for the duration of the attack. There may be other obvious signs as well. |
| **Syslogd**  R-b-Deny (Administrative) | **Description** The Syslogd exploit is a denial of service attack that allows an attacker to remotely kill the syslogd service on a Solaris server. When Solaris syslogd receives an external message it attempts to do a DNS lookup on the source IP address. If this IP address doesn't match a valid DNS record, then syslogd will crash with a Segmentation Fault [54].  **Simulation Details** The Syslogd exploit used in the 1998 DARPA evaluation was a C program that was originally posted to the Bugtraq mailing list. This code was compiled to create an exploit program that was used to remotely cause the syslogd program to crash on a Solaris 2.5 server. Once syslogd has crashed it must be manually restarted by an administrator for the logging service to be restored.  **Attack Signature** The one way to reliably recognize this attack with a network-monitoring intrusion detection system is to notice a packet destined for the syslog port that contains an unreachable source address. Of course, it may not be realistic for an intrusion detection system to check every packet destined for the syslog port to see whether or not the source address is resolvable. If no remote system logging is expected to occur on a particular network, any external syslog messages appearing on this network is likely to be an attack. Finally, a host-based intrusion detection system could be configured to notice the syslog process die because of a segmentation fault. |
| **tcpreset**  R-a-Deny (Temp.) | **Description** TCP Reset is a denial of service attack that disrupts TCP connections made to the victim machine. That is, the attacker listens (on a local or wide-area network) for tcp connections to the victim, and sends a spoofed tcp RESET packet to the victim, thus causing the victim to inadvertently terminate the TCP connection.  **Simulation Details** The attack exists in the form of a binary written especially for the 1999 Lincoln Evaluation. The binary, "coneos", takes the victim ip as a commandline parameter and must be run with root-level permissions on the attacking machine (in order to listen for connections to the victim promiscously and do the spoofing).  **Attack Signature** One way to detect the attack would be to look at the TCP session setup/takedown process, and note cases in which RESET packets appear to come from the machine that had initially attempted to begin the connection. (This might not be foolproof however, as there might be cases when this is a common/normal occurance.)  Unlike the arppoison attack, tcpreset was 100% effective in resetting (causing termination of) tcp connections as they were being brought up. (In the 1999 Lincoln Evaluation) |
| **Teardrop**  R-a-Deny (Temporary) | **Description** The teardrop exploit is a denial of service attack that exploits a flaw in the implementation of older TCP/IP stacks. Some implementations of the IP fragmentation re-assembly code on these platforms does not properly handle overlapping IP fragments [17].  **Simulation Details** The teardrop name is derived from a widely available C program that exploits this vulnerability. This exploit code can be found at http://www.rootshell.com and in the Bugtraq archives. Although many systems are rumored to be vulnerable to the teardrop attack, of the systems used in the DARPA evaluation, only the Redhat Linux 4.2 systems were vulnerable. The teardrop attack would cause these machines to reboot.  **Attack Signature** An intrusion detection system can find this attack by looking for two specially fragmented IP datagrams. The first datagram is a 0 offset fragment with a payload of size N, with the MF bit on (the data content of the packet is irrelevant). The second datagram is the last fragment (MF == 0), with a positive offset greater than N and with a payload of size less than N [5]. |
| **Udpstorm**  R-a-Deny (Administrative) | **Description** A Udpstorm attack is a denial of service attack that causes network congestion and slowdown. When a connection is established between two UDP services, each of which produces output, these two services can produce a very high number of packets that can lead to a denial of service on the machine(s) where the services are offered. Anyone with network connectivity can launch an attack; no account access is needed. For example, by connecting a host's chargen service to the echo service on the same or another machine, all affected machines may be effectively taken out of service because of the excessively high number of packets produced. An illustration of such an attack is presented in Figure 6-2. The figure demonstrates how an attacker is able to create a never-ending stream of packets between the echo ports of two victims by sending a single spoofed packet. First, the attacker forges a single packet that has been spoofed to look like it is coming from the echo port on the first victim machine and sends it to the second victim. The echo service blindly responds to any request it receives by simply echoing the data of the request back to the machine and port that sent the echo request, so when the victim receives this spoofed packet it sends a response to the echo port of the second victim. This second victim responds in like kind, and the loop of traffic continues until it is stopped by intervention from an external source [10].  **Simulation Details** Code that exploits this vulnerability was posted to the bugtraq mailing list. This program sends a single spoofed UDP packet to a host. This single spoofed packet is able to create a never-ending stream of data being sent from the echo port of one machine to the echo port of another. This loop created network congestion and slowdown that would continue until the inetd daemon was restarted on one of two victim machines.  **Attack Signature** This attack can be identified in two ways. First, the single packet that initiates the attack can be recognized because it is a packet originating from outside the network that has been spoofed to appear as if it is coming from a machine inside the network. Second, once the loop of network traffic has been initiated, an intrusion detection system that can see network traffic on the inside of the network can note that traffic is being sent from the chargen or echo port of one machine to the chargen or echo port of another. |

### **User to Root Attacks**

User to Root exploits are a class of exploit in which the attacker starts out with access to a normal user account on the system (perhaps gained by sniffing passwords, a dictionary attack, or social engineering) and is able to exploit some vulnerability to gain root access to the system. There are several different types of User to Root attacks. The most common is the buffer overflow attack. Buffer overflows occur when a program copies too much data into a static buffer without checking to make sure that the data will fit. For example, if a program expects the user to input the user's first name, the programmer must decide how many characters that first name buffer will require. Assume the program allocates 20 characters for the first name buffer. Now, suppose the user's first name has 35 characters. The last 15 characters will overflow the name buffer. When this overflow occurs, the last 15 characters are placed on the stack, overwriting the next set of instructions that was to be executed. By carefully manipulating the data that overflows onto the stack, an attacker can cause arbitrary commands to be executed by the operating system. Despite the fact that programmers can eliminate this problem through careful programming techniques, some common utilities are susceptible to buffer overflow attacks [2]. Another class of User to Root attack exploits programs that make assumptions about the environment in which they are running. A good example of such an attack is the load-module attack, which is discussed below. Other User to Root attacks take advantage of programs that are not careful about the way they manage temporary files. Finally, some User to Root vulnerabilities exist because of an exploitable race condition in the actions of a single program, or two or more programs running simultaneously [27]. Although careful programming could eliminate all of these vulnerabilities, bugs like these are present in every major version of UNIX and Microsoft Windows available today. Figure 7-1 summarizes the User to Root attacks used in the 1998 DARPA evaluation. Note from this table that all of the User to Root attacks can be run from any interactive user session (such as by sitting at the console, or interacting through telnet or rlogin), and that all of the attacks spawn a new shell with root privileges. The following sections describe each of the User to Root attacks that was used in the 1998 DARPA intrusion detection evaluation in greater detail.

|  |  |
| --- | --- |
| **anypw**  U--S | **Description** NukePW is a Console User to Root attack that allows the attacker to logon to the system without a password. A boot disk is used to modify the NT authentication package so that a valid username can login with any password string. Logins via telnet also work with any password.  **Simulation details** Insert the bootdisk containing the attack installed in the boot sector. Reboot the machine. A hexadecimal number will appear in the upper left of the screen. The number will slowly increment as the attack searches for the signature of the authentication package. Wait until an asterisk appears beside the number. Remove the diskette and reboot the machine. Later telnet to the machine as an administrator and enter any password to logon.  **Verification** Any password will be accepted with a valid username.  **Cleanup** The file c:\winnt\system32\msv1\_0.dll file must be replace with an uncorrupted copy.  **Attack signature** The sniffed data will reveal remote logons with incorrect password strings. If the attacker physically logs on to the machine and then locks the machine, only the password used to logon can unlock the machine.  The victim's security audit log will indicate a reboot after the system is restarted. Most likely, the attacker had to hard reboot the machine (physically press the reset button) because he did not have a password to login or unlock the machine. A soft reboot audit signature is a "SeShutPrivilege" Privilege Use Event followed by an event stating, "Windows NT is starting up." A hard reboot audit signature does not include the "SeShutdownPrivilege" event. A hard reboot can be used to detect but not identify the AnyPW attack, because other attacks may also result in hard reboots (DoSNuke, NTFSDOS, etc.). In addition, a hard reboot may occur without an attack (power outages, system halts, etc). |
| **casesen**  U2R | **Description** CaseSen is a User to Root attack that exploits the case sensitivity of the NT object directory. The attacker ftps three attack files to the victim: soundedt.exe, editwavs.exe, psxss.exe (the names of the files were chosen to make the attack more stealthy). The attacker then telnets to the victim and runs soundedt.exe. A new object is created in the NT object directory called \??\c: which links to the directory containing the attack files. A posix application is started activating the trojan attack file, psxss.exe, which results in the logged in user being added to the Administrators user group.  Read the page from NTSecurity.net for a more complete explanation.  **Simulation details** Two fully automated Expect scripts are run on a Unix attacker: case\_s.exp and case\_b.exp. Case\_s.exp (the setup script) ftps the attack files, telnets to the victim, and runs the attack. It also deletes the three attack files after they have been used. Case\_b.exp (the breakin script) telnets to the victim with the new Administrator privileges, runs some commands, and cleans up by removing the user from the Administrators group and deleting other files generated by the attack.  First put the three attack files in /home/, where is the username of the attacker.  Run case\_s.exp from a Unix attacker.  "case\_s.exp <host> <user> <password>" e.g. "case\_s.exp 172.16.112.100 lucyj lucy7"  Later run case\_b.exp.  "case\_b.exp <host> <user> <password>" e.g. "case\_b.exp 172.16.112.100 lucyj lucy7"  **Verification** After case\_s.exp runs, the username specified in the command line of the attack should appear in the Administrators group on the victim machine (check the User Manager). After case\_b.runs, the username should no longer be in the Administrators group.  The tcpdump data can also be used to verify the attack. The dump file for the breakin script should contain the line "command completed successfully." This indicates that the command to remove the user from the Administrators group was successful, which means the entire attack was successful.  **Cleanup** The setup script deletes the three attack files. The breakin script removes the user from the Administrators group and deletes c:\inetpub\ftproot\winnt\, which is created during the attack setup.  **Attack signature** Sniffing the network traffic will reveal the transfer of the three files, psxss.exe, editwavs.exe, and soundedt.exe, and the execution of soundedt.exe. However, editwavs.exe and soundedt.exe were a names chosen specifically for the simulation. Other versions of the attack may use different filenames, e.g. dummyapp.exe.  A strong signature is left in the victim's security log as well. The log shows the execution of the files posix.exe and psxss.exe, whose filenames will not differ in other versions of the attack. In addition, a log entry states that the user is added to the Administrators group by NT AUTHORITY/SYSTEM because the user is added by an application (very uncommon). Normally, the Administrator would use usrmgr to add the user to a group, and the log entry would indicate that the user was added by ADMINISTRATOR, not NT AUTHORITY/SYSTEM.  **Problems and solutions** Usually the victim machine must be rebooted before the attack can be launched a second time. The attack can be launched no more than two times without rebooting the victim. |
| **Eject**  U-b-S | **Description** The Eject attack exploits a buffer overflow is the 'eject' binary distributed with Solaris 2.5. In Solaris 2.5, removable media devices that do not have an eject button or removable media devices that are managed by Volume Management use the eject program. Due to insufficient bounds checking on arguments in the volume management library, libvolmgt.so.1, it is possible to overwrite the internal stack space of the eject program. If exploited, this vulnerability can be used to gain root access on attacked systems[60].  **Simulation Details** A truncated version of the eject exploit that was used in the 1998 evaluation is shown in Figure 7-2. This exploit was originally posted to the bugtraq mailing list. The exploit script, once compiled, can be run in a command line session on a Solaris server to spawn a shell that ran with root privileges. There are several ways that an intrusion detection system might identify this attack. Assuming an attacker already has access to an account on the victim machine and is running the exploit as part of a remote session, a network-based system can look at the contents of the telnet or rlogin session the attacker is using and notice one of several features. First, assuming that an attacker transmits the C code to the victim machine unencrypted, the intrusion detection system could look for specific features of the source code. For example, an intrusion detection system could look for the string 'Jumping to address' on line 45 of the source code or the line 'execl (æ/bin/eject' ; æeject', & buf(char \*) 0);' from line 47. Even if the attacker encrypts the source code, the attack leaves a distinct signature. A segment of the transcript from an actual instantiation of this attack that was used in the simulation is shown below: pascal> /tmp/162562 Jumping to address 0xeffff6a0 B[364] E[400] SO[704] # An intrusion detection system that saw only these three lines has several clues that an attack has taken place. First, the user's prompt has changed from 'pascal>' to '#' without running the su command. Second, the string 'Jumping to address' is again printed. Of course, a careful attacker would remove this line from the source code, but simply looking for the string 'Jumping to address' would catch the less careful attacker. Finally, a host-based intrusion detection system could catch an eject attack either by noticing the invocation of the eject program with a large argument, or by performing bottleneck verification [43] on the transition from normal user to root user and noticing that the user did not make a legal user to root transition. |
| **Ffbconfig**  U-b-S | **Description** The Ffbconfig attack exploits a buffer overflow is the 'ffbconfig' program distributed with Solaris 2.5. The ffbconfig program configures the Creator Fast Frame Buffer (FFB) Graphics Accelerator, which is part of the FFB Configuration Software Package, SUNWffbcf. This software is used when the FFB Graphics accelerator card is installed. Due to insufficient bounds checking on arguments, it is possible to overwrite the internal stack space of the ffbconfig program [61].  **Simulation Details** This attack is very similar to the eject attack described above. Once again, C code to exploit this vulnerability was posted on the Bugtraq mailing list.  **Attack Signature** The means by which an intrusion detection system can identify the ffbconfig attack are similar to those described above for the Eject exploit. An attacker who is exploiting this vulnerability must first transfer the code for the exploit (either C code to be compiled, or pre-compiled code) onto the victim machine, and then run the exploit. As with the eject exploit, there are strings within the source code of the ffbconfig exploit script which identify the attack to a network or host based intrusion detection system. A host-based intrusion detection system can perform bottleneck verification or look for the invocation of the command '/usr/sbin/ffbconfig/' with an oversized argument for the '-dev' parameter. |
| **Fdformat**  U-b-S | **Description** The Fdformat attack exploits a buffer overflow is the 'fdformat' program distributed with Solaris 2.5. The fdformat program formats diskettes and PCMCIA memory cards. The program also uses the same volume management library, libvolmgt.so.1, and is exposed to the same vulnerability as the eject program [60].  **Simulation Details** Exploit code for this vulnerability was posted to the Rootshell Website [53] in March, 1997. The exploit code was used unmodified for the DARPA evaluation.  **Attack Signature** Methods for identifying this attack are nearly identical to those described for the eject and ffbconfig attacks. |
| **Loadmodule**  U-b-S | **Description** The Loadmodule attack is a User to Root attack against SunOS 4.1 systems that use the xnews window system. The loadmodule program within SunOS 4.1.x is used by the xnews window system server to load two dynamically loadable kernel drivers into the currently running system and to create special devices in the /dev directory to use those modules. Because of a bug in the way the loadmodule program sanitizes its environment, unauthorized users can gain root access on the local machine [8].  **Simulation Details** The code for the loadmodule exploit script is widely available on the internet. This code is usually in the form of a shell scriptùit does not need to be compiled before it is run. The steps of the attack are quite simple: 1. Change the value of the internal field separator (or IFS) variable to a slash. 2. Add '.' To the front of the PATH variable 3. Copy '/bin/sh' to './bin' 4. Execute '/usr/openwin/bin/loadmodule a'. When the loadmodule shell script (which is setuid root by default) executes, it attempts to run the command 'exec(æ/bin/a');'. Since the IFS variable has been changed to '/' the string '/bin/a' is parsed into two tokens, and the loadmodule script attempts to run the firstù'bin'. Since the attacker has conveniently put a copy of '/bin/sh' in the current directory and named it 'bin', the loadmodule script (which is running as root) will exec './bin'ùgiving the attacker a shell with root privileges.  **Attack Signature** This attack can be identified either by performing bottleneck verification with a host- based intrusion detection system, or by keyword spotting with a network based intrusion detection system. A simple rule could say that any session which contained the strings 'set $IFS='\/'' and 'loadmodule' in close proximity was probably a loadmodule attack. Of course, an attacker could quite easily hide from such a simple rule. Detailed discussion of such methods for hiding is presented in Chapter 11. |
| **ntfsdos**  U--S | **Description** This console-based attack reboots the system from a floppy disk containing NTFSDOS.EXE. This executable is used to mount the hard drives, giving the attacker the ability to read and copy files that would otherwise be protected by Windows NTFS security. The attack may be consider a User to Root attack because the attacker can access files that only the Administrator has permission to use.  NTFSDOS.EXE is a network file system redirector for DOS/Windows that is able to recognize and mount NTFS drives for transparent access. It makes NTFS drives appear indistinguishable from standard FAT drives, providing the ability to navigate, view and execute programs on them from DOS or from Windows.  **Simulation details** The attack is completely manual. Insert the diskette (a Windows98 boot diskette containing the ntfsdos program) into Hume's A drive. Push the reset button on the CPU. After the system reboots, type 'ntfsdos' at the DOS prompt. Change directories to c:\secret. Copy the secret files to the diskette. Remove the diskette and reboot the machine.  **Verification** The secret files will be stored on the diskette.  **Attack signature** The attack cannot be sniffed because it does not create network traffic and it cannot be logged because windows NT is not running (and therefore not auditing) during the attack. The victim's security audit log will indicate a reboot after the system is restarted. Most likely, the attacker had to hard reboot the machine (physically press the reset button) because he did not have a password to login or unlock the machine. A soft reboot audit signature is a "SeShutPrivilege" Privilege Use Event followed by an event stating, "Windows NT is starting up." A hard reboot audit signature does not include the "SeShutdownPrivilege" event. A hard reboot can be used to detect but not identify the NTFSDOS attack, because other attacks may also result in hard reboots (DoSNuke, AnyPW, etc.). In addition, a hard reboot may occur without an attack (power outages, system halts, etc). Another way to detect the attack is to monitor the last accessed time and date for the secret files. |
| **Perl**  U-b-S | **Description** The Perl attack is a User to Root attack that exploits a bug in some Perl implementations. Suidperl is a version of Perl that supports saved set-user-ID and set-group-ID scripts. In early versions of suidperl the interpreter does not properly relinquish its root privileges when changing its effective user and group IDs. On a system that has the suidperl, or sperl, program installed and supports saved set-user-ID and saved set-group-ID, anyone with access to an account on the system can gain root access [12].  **Simulation Details** A perl script that uses this vulnerability to gain root access was made publicly available in August 1996 [51]. The code is only two lines long, and can easily be executed from the command-line. Once this perl script has run, the user will be presented with a new shell that is running with root privileges.  **Attack Signature** The methods by which an intrusion detection system could identify a perl exploit attempt are identical to those described above for the loadmodule attack. A host-based intrusion detection system could notice that a root shell was spawned without a legal user to root transition, or a network-based intrusion detection system could look the strings '$>=0; $<=0;' or 'exec (æ/bin/sh');', which have little valid use except in an exploit attempt. |
| **Ps**  U-b-S | **Description** The Ps attack takes advantage of a race condition in the version of 'ps' distributed with Solaris 2.5 and allows an attacker to execute arbitrary code with root privilege. This race condition can only be exploited to gain root access if the user has access to the temporary files. Access to temporary files may be obtained if the permissions on the /tmp and /var/tmp directories are set incorrectly. Any users logged in to the system can gain unauthorized root privileges by exploiting this race condition [9].  **Simulation Details** This exploit is possible because of a combination of the ps program not carefully managing temporary files and a buffer overflow. A shell script that builds a carefully constructed temporary file, creates a C-file, compiles the code and executes the exploit was found at Rootshell.com [52]. Once an attacker has transfers this shell script onto a Solaris victim machine and runs it, a root shell will be spawned for the attacker.  **Attack Signature** Methods for finding this attack are essentially the same as the methods for finding the eject, ffbconfig, or fdformat attacks. |
| **sechole**  U--S | **Description** The attacker (a regular user) ftps to the victim and uploads test.exe and testfile.dll (filenames were chosen to be stealthy). The attacker then telnets to the victim and runs test.exe. The result is the attacker is added to the Administrators group.  Test.exe locates the memory address of a particular API function (OpenProcess) and modifies the instructions at that address in a running image of the exploit program on the local system. Test.exe requests debug rights that gives it elevated privileges. The request is successful because the access check for this right is expected to be done in the API that was successfully modified by the exploit program. Test.exe adds the user who invoked test.exe to the local Administrators group.  View the file from Micrsoft Knowledge Base for more information.  **Simulation details**Two fully automated Expect scripts are used. The setup script, sec\_s.exp, ftps the files from a Unix attacker, telnets to run the attack, and cleans up by deleting the attack files. The breakin script, sec\_b.exp, telnets to the NT victim with the new Administrator privileges, runs some commands, and removes the attacker from the Administrators group before logging off.  First put test.exe and testfile.dll in /home/, where is the username of the attacker. Run sec\_s.exp from a Unix attacker.  "sec\_s.exp <host> <user> <password>"  e.g. "sec\_s.exp 172.16.112.100 lucyj lucy7"  Later, run sec\_b.exp from a Unix attacker.  "sec\_b.exp <host> <user> <password>" e.g. "sec\_b.exp 172.16.112.100 lucyj lucy7"  **Verification**After running sec\_s.exp, the username specified in the command line should be in the Administrators group on the victim machine (verify by running usrmgr). After running sec\_b.exp, the user should no longer be in the Administrators group.  The tcpdump data can also be used to verify the attack. The dump file for the breakin script should contain the line "command completed successfully." This indicates that the command to remove the user from the Administrators group was successful, which means the entire attack was successful.  **Cleanup** The setup script deletes the attack files. The breakin script removes the user from the Administrators group.  **Attack signature** Sniffing the network traffic will reveal the uploading of test.exe and testfile.dll. The file transfers result in character strings that may be used in keyword detection. Another indicator of the attack is the security log entry for a user being added to the administrators group. The log entry states that the user is added to the administrators group by NT AUTHORITY/SYSTEM because the user is added by an application (very uncommon). Normally, Administrator would use usrmgr to add the user to a group, and the log entry would indicate that the user was added by ADMINISTRATOR, not NT AUTHORITY/SYSTEM.  **Problems and solutions**It is unlikely, but the victim system may lock up after the attack. If this happens, the victim will just reboot the machine. The attack still succeeds. |
| **Xterm**  U-b-S | **Description** The Xterm attack exploits a buffer overflow in the Xaw library distributed with Redhat Linux 5.0 (as well as other operating systems not used in the simulation) and allows an attacker to execute arbitrary instructions with root privilege. Problems exist in both the xterm program and the Xaw library that allow user supplied data to cause buffer overflows in both the xterm program and any program that uses the Xaw library. These buffer overflows are associated with the processing of data related to the inputMethod and preeditType resources (for both xterm and Xaw) and the \*Keymap resources (for xterm). Exploiting these buffer overflows with xterm when it is installed setuid-root or with any setuid-root program that uses the Xaw library can allow an unprivileged user to gain root access to the system [21].  **Simulation Details** C source code that exploits this vulnerability on Redhat Linux 5.0 systems was found at the Rootshell website [56]. Once again, an attacker can compile this C code, and when the resulting program is run the attacker is given a shell running with root privileges.  **Attack Signature** Methods for finding this attack are essentially the same as the methods for finding the eject, ffbconfig, or fdformat attacks. |
| **yaga**  U--S | **Description** Yaga is a User-to-Root attack. It adds the attacker to the Domain Admins group by hacking the registry. The attacker edits the victim's registry so that the next time a system service crashes on the victim, the attacker is added to the Domain Admins group. To setup the attack, the attacker must put onto the victim machine a file with the registry edit information. The attacker must also edit the registry. All this can be done via a telnet session. Once the setup is complete, the attacker can remotely crash a service on the victim machine (using CrashIIS for example) to add the user to the Domain Admins group.  **Simulation details** The setup expect script, yaga\_s.exp, telnets to the NT victim computer and creates the file. It then runs the CrashIIS attack to crash the IIS web server thereby adding the user to the Domain Admins group. The web server remains halted.  The breakin expect script, yaga\_b.exp, telnets to the victim as the user with the new Domain Admin permissions, runs some commands, and cleans up by removing the user from the Domain Admins group and restoring the original registry key.  Run yaga\_s.exp from a Unix attacker.  "yaga\_s.exp <host> <user> <password>" e.g. "yaga\_s.exp 172.16.112.100 lucyj lucy7"  Later, run yaga\_b.exp.  "yaga\_b.exp <host> <user> <password>" e.g. "yaga\_b.exp 172.16.112.100 lucyj lucy7"  **Verification** After yaga\_s runs, the user will be in the Domain Admins group. Run usrmgr on the victim machine to verify. After yaga\_b runs, the user should no longer be in the Domain Admins group.  The tcpdump data can also be used to verify the attack. The dump file for the breakin script should contain the line "command completed successfully." This indicates that the command to remove the user from the Domain Admins group was successful, which means the entire attack was successful.  **Cleanup** The breakin script removes the user from the Domain Admins group and restores the original AeDebug registry key. Manually, restart the IIS server(s). Sometimes just the web server crashes, but often times the ftp and gopher servers need to be restarted too.  **Attack signature** The creation of the file with registry information, "entry," is done with the cat command. As a result, there is a lot of indicative text being sent over the data line, "Aedebug" for example. Also, the sniffed data and the NT security audit logs for the victim machine will show that regedit.exe was run by the attacker.  Another indicator of the attack is the log entry for a user being added to the Domain Admins group. The log entry states that the user is added to the Domain Admins group by NT AUTHORITY/SYSTEM because the user is added by an application (very uncommon). Normally, the Administrator would use usrmgr to add the user to a group, and the log entry would indicate that the user was added by ADMINISTRATOR, not NT AUTHORITY/SYSTEM.  Finally, because yaga uses the CrashIIS attack, any attack signature left by CrashIIS will also be left by yaga. |

### **Remote to User Attacks**

A Remote to User attack occurs when an attacker who has the ability to send packets to a machine over a network but who does not have an account on that machine exploits some vulnerability to gain local access as a user of that machine. There are many possible ways an attacker can gain unauthorized access to a local account on a machine. Some of the attacks discussed within this section exploit buffer overflows in network server software (imap, named, sendmail). The Dictionary, Ftp-Write, Guest and Xsnoop attacks all attempt to exploit weak or misconfigured system security policies. The Xlock attack involves social engineering in order for the attack to be successful the attacker must successfully spoof a human operator into supplying their password to a screensaver that is actually a trojan horse. Figure 8-1 summarizes the characteristics of the Remote to User attacks that were included in the 1998 DARPA intrusion detection evaluation. The following sections provide details of each of these attacks.

|  |  |
| --- | --- |
| **Dictionary**  R-a-U | **Description** The Dictionary attack is a Remote to Local User attack in which an attacker tries to gain access to some machine by making repeated guesses at possible user names and passwords. Users typically do not choose good passwords, so an attacker who knows the username of a particular user (or the names of all users) will attempt to gain access to this user's account by making guesses at possible passwords. Dictionary guessing can be done with many services; telnet, ftp, pop, rlogin, and imap are the most prominent services that support authentication using user names and passwords. Figure 8-2 is a plot of the connections made to the pop3 port of a victim machine during a dictionary attack that is using the pop service to check for valid login/password combinations. The horizontal axis of this plot represents time in minutes, and each line segment in the plot is a single connection to the pop3 service. Lines representing successive sessions are displaced vertically slightly and wrap around (in this figure at roughly 1.5 minutes). The length of the lines represents the length of the pop sessions. Lines begin with a greater- than sign '>' and end with a less-than sign '<', and thus form an 'x' for short sessions. In all, this example dictionary attack consists of 40 attempts to log in, with a 4 second delay between each attempt.  **Simulation Details** A perl script that performed automated password guessing on a variety of services was developed specifically for use in our evaluation. The 'Netguess' perl script could take in a file of possible username/password combinations, or create guesses for the password based on simple permutations of the username. Within the simulation, this script was used to perform between 10 and 100 login attempts on the telnet, ftp, and pop services. When the script was successful in gaining access to the system, it would immediately quit and report success.  **Attack Signature** An intrusion detection system that finds attempted dictionary attacks needs to know the session protocol of every service that provides username/password authentication. For a given service, the intrusion detection system must be able to recognize and record failed login attempts. Once this functionality is available, detecting dictionary attacks is a matter of setting a detection threshold based on the number of failed login attempts within a given period of time. |
| **FrameSpoofer** | **Description** This attacks tricks the victim into believing he is viewing a trusted web site, but in actuality the page's main body is spoofed with a frame created by the attacker.  The attacker sends a fake email to the victim directing the victim to a web page which displays security procedures for Air Force Base computer networks. The page contains a link to a page with security procedures specific to Eyrie Air Force Base. When the user clicks on the link, it actually runs a javascript function which brings up the trusted web site but then inserts a malicious web page (with misleading information) into its main frame. The URL displayed in the browser remains unchanged.  View the page from NTSecurity.net for more information.  **Simulation details** Use sendmail.pl to send the email to Hume. The victim must manually receive the mail and click on the links. On a Unix attacker: "/.sim/bin/sendmail.pl mail.txt " As the victim, receive the mail and follow the links.  Important: Clear the browser cache on the victim machine before executing the attack. Otherwise, the browser will load cached pages from a previous execution of the attack, and no web traffic will be generated on the network.  **Verification** The security procedures page for Eyrie Air Force base will display one page and then switch to the spoofed page.  **Attack signature** Audit logs reveal nothing. The sniffed data will reveal the source code for the javascript webpage.  The attack can be detected if one carefully examines the javascript code contained in the sniffed data. |
| **Ftp-write**  R-c-U | **Description** The Ftp-write attack is a Remote to Local User attack that takes advantage of a common anonymous ftp misconfiguration. The anonymous ftp root directory and its subdirectories should not be owned by the ftp account or be in the same group as the ftp account. If any of these directories are owned by ftp or are in the same group as the ftp account and are not write protected, an intruder will be able to add files (such as an rhosts file) and eventually gain local access to the system [7].  **Simulation Details** This attack was implemented as an expect script which was created explicitly for use in the simulation. This expect script anonymously logged in to the ftp service on the victim machine, created a '.rhosts' file with the string '+ +' in it within the ftp home directory, disconnected from the ftp server, used rlogin to connect back to the server as user 'ftp', and finally performed some actions on the victim machine. Creating a '.rhosts' file in the ftp home directory with the entry '+ +' in it allows any user from any machine to rlogin to the victim as user 'ftp'.  **Attack Signature** An intrusion detection system can monitor for this attack by watching all anonymous ftp sessions and assuring that no files are created in the ftp root directory. |
| **Guest**  R-c-U | **Description** The Guest attack is a variant of the Dictionary attack described in Section 8.1. On badly configured systems, guest accounts are often left with no password or with an easy to guess password. Because most operating systems ship with the guest account activated by default, this is one of the first and simplest vulnerabilities an attacker will attempt to exploit [27].  **Simulation Details** The Guest attack is a simplified version of the Dictionary attack discussed earlier in this chapter. The same 'Netguess' perl script that was used to simulate a Dictionary attack was used to simulate the Guest attackùthe only difference between the implementation of the two attacks was the command-line options that were passed to the Netguess program. Whereas the Dictionary attack would try up to a hundred user names and thousands of username/password combinations, the Guest attack would make only a couple of login attempts, using combinations such as 'guest/', 'guest/guest', 'anonymous/' and 'anonymous/anonymous'.  **Attack Signature** Because the Guest attack is essentially a subset of the Dictionary attack, the methods for finding the two attacks are basically the same. An intrusion detection system that is looking for a Dictionary attack already should need only minor tuning in order to find attempts to log in to the guest account. |
| **HttpTunnel** | **Description** In an Http Tunnel attack, the attacker gains local access to the machine to be attacked and then sets up and configures an http client to periodically query a web server that the attacker has setup at some remote host. When the client connects, the server is able to send cookies that could request information be sent by the client, such as the password file on the victim machine. In effect, the attacker is able to "tunnel" requests for information through the http protocol.  **Simulation Details** This attack is carried out in two phases, first a setup, then use of the tunnel. The setup occurs with a login, by the attacker as a local user, to the victim, and a transfer of the web-client (script, source code, or binary) to the victim. During the setup login, the attacker might also add to their crontab file so that the webclient is periodically started. The Tunnel is used when the webclient connects to the attackers server, cookies requesting information are sent to the victim and the requests are fulfilled. This web server could run on any port that the attacker chooses, however most often, a non-well known port above 1024 will be used.  **Attack Signature** The Http Tunnel attack can be recognized by watching for the setup login session, transfer of the client to the victim, and perhaps setting up a job to be run periodically, or starting a background process to run the client. The using of the tunnel could be noticed by periodic connections to from the victim to the attacker on non-well-known ports, or ports greater than 1024. (However - port 80 could be used as well.) |
| **Imap**  R-b-S | **Description** The Imap attack exploits a buffer overflow in the Imap server of Redhat Linux 4.2 that allows remote attackers to execute arbitrary instructions with root privileges. The Imap server must be run with root privileges so it can access mail folders and undertake some file manipulation on behalf of the user logging in. After login, these privileges are discarded. However, a buffer overflow bug exists in the authentication code of the login transaction, and this bug can be exploited to gain root access on the server. By sending carefully crafted text to a system running a vulnerable version of the Imap server, remote users can cause a buffer overflow and execute arbitrary instructions with root privileges [16].  **Simulation Details** The Imap attack used in the 1998 DARPA intrusion detection evaluation was part of the Impack 1.03 attack toolkit [34]. This toolkit contained precompiled binary programs for the Linux platform that would scan for vulnerable machines, as well as send the necessary message to exploit the buffer overflow and gain access to a root shell. The Impack contained detailed instructions on how to use these precompiled programs and took very little skill to use. The release of the Impack made this vulnerability especially dangerous, as any user with a Linux machine and the ability to follow instructions could use this attack to remotely gain root access to any vulnerable hosts.  **Attack Signature** The Imap attack can be identified by an intrusion detection system that has been programmed to monitor network traffic for oversized Imap authentication strings. |
| **Named**  R-b-S | **Description** The Named attack exploits a buffer overflow in BIND version 4.9 releases prior to BIND 4.9.7 and BIND 8 releases prior to 8.1.2. An improperly or maliciously formatted inverse query on a TCP stream destined for the named service can crash the named server or allow an attacker to gain root privileges [19].  **Simulation Details** The version of the Named exploit used in the simulation was adapted from a C program originally posted to the Bugtraq mailing list. This program, once compiled, would connect to the named port on a victim machine and overflow a buffer of the named server with instructions that would send an xterm running with root privilege back to the attacker's X console. Because this attack involved interaction with X, all of the Named attacks included in the simulation were run by human actors.  **Attack Signature** The Named attack can be identified by watching DNS inverse query requests for messages that are longer than the 4096 byte buffer allocated for these requests within the 'named' server. |
| **ncftp**  R-b-User | **Description** Ncftp is an ascii UI ftp program for linux. This attack exploits one of the popular features of the program: the ability to get subdirectories recursively. New (sub)directories are created on the local machine using the system() command (e.g. if any directories on the remote host contain an expression in backticks, that expression will be evaluated on the local machine when the directory is created.  **Simulation Details** A directory was created on an outside attacking machine with a expression in backticks. This malicious directory was hidden underneath some other directories. When a user on a victim machine used ncftp to download the top level directory recursively, the malicous command was executed on the victim's host. In the 1999 DARPA evaluation, the malicious command mailed the victim's /etc/passwd file to the attacker. This attack was originally posted on http://www.rootshell.com.  **Attack Signature** In sniffing data, you can see the names of all the files that are trasfered in an ftp session. However, it is difficult to see what is the malicious command because most of the harmful expression in backticks is encoded in octal character codes. In addition, the mail sent back over to the attacker was encrypted by adding in spurious characters using sed so the file doesn't resemble /etc/passwd. |
| **netbus**  R--User | **DESRIPTION:** NetBus is a Remote to Local attack. The attacker uses a trojan program to install and run the Netbus server on the victim machine. Once Netbus is running, it acts as a backdoor. The attacker can then remotely access the machine using the Netbus client.  The attacker sends an email with a executable attachment (a game called whackamole). When the victim runs whackamole, it launches the Netbus server (explore.exe), and then launches the whackamole game. It also edits the registry so the Netbus server runs at every login.  The attacker can use the Netbus client program to manipulate files on the victim machine, download screen dumps, move the mouse pointer, etc. The attacker's access privileges are identical to the user currently logged on to the victim machine. So if an administrator is using the victim, the attacker can run a net command to setup a new admin user, resulting in a remote-to-root attack. The Netbus client can also be used as a probe attack to scan IP addresses for NetBus servers.  View the information page from www.netbus.com.  **Simulation details** Use Sendmail.pl to send the email to Hume. On a Unix attacker: "/.sim/bin/sendmail.pl netbus.txt " or send the email with attachment from an NT attacker. Later, on a NT attacker, run the NetBus client with the victim IP address (172.16.112.100).  **Verification** After the attack has completed, the victim machine should be remotely accessible via the Netbus client running on a Windows NT machine.  **Cleanup** Click the Server Admin button on the NetBus client and choose Remove Server. The registry will remain changed and explore.exe will remain in c:\winnt but the server will no longer be running and will not run until the attack is setup again.  **Attack signature** When the attacker uses the netbus client to access the victim, it creates network traffic that is easy to identify. The word Netbus will show up in the sniffed data and all of the commands are in plaintext.  Explore.exe is the most commonly used filename for the Netbus attack. The NT security log will show that explore.exe ran when the attachment was executed. When the Netbus server is running, it shows up in the victim process table as explore.exe. |
| **netcat**  R--User | **Description** NetCat is a Remote to Local attack. The attacker uses a trojan to install and run the netcat program on the victim machine on a specific port (53). Once netcat is running, it acts as a backdoor. The attacker can remotely access the machine through the netcat port without a username or password.  The attacker sends an email with a self-extracting executable attachment called y2ktest.exe. The email states that the file will install a program to test the victim machine for y2k compliance. When the victim opens the file, it creates a new folder c:\y2ktest and puts the y2ktest files into it. It also places into the folder the attack files, winlog.bat, winlog.exe, and winlog.txt (filenames were chosen to be stealthy).  The batch file, winlog.bat, automatically runs and controls the attack. It edits the Run key in the registry with winlog.txt so that the command  winlog -L -d -p 53 -t -e cmd.exe  runs every time a user logs on to the machine. Finally, all unnecessary attack files are deleted. The y2ktest folder and its contents, and c:\winnt\system32\winlog.exe are what remain.  The attacker later uses the command "nc -v 53" on a remote machine (Unix or NT with the nc program) to telnet to the victim without a username or password.  View the page from l0pht.com for more information.  **Simulation details** A Unix attacker sets up the backdoor on an NT victim machine. Usually, an NT attacker uses the backdoor, but a Unix attacker can use it as well.  Sendmail.pl is used to send the preassembled mail message y2katack.txt. The victim must manually open the email and runs the trojan. Then the attacker manually runs the client program.  On a Unix attacker:  /.sim/bin/sendmail.pl y2katack.txt  Later, on a Unix or NT attacker with netcat:  nc -v 172.16.112.100 53  The files included in the self-extracting zip file were originally called nc.bat, nc.exe, and nc.txt. They were changed to winlog.bat, winlog.exe, and winlog.txt respectively. This way, when the backdoor runs, winlog will appear in the victim’s process list instead of the more conspicuous name, nc. Sniffing the email transfer will not detect the attack because the attachment is MIME encoded.  When the self-extracting zip file is run, it tells the user that it puts a total of seven files into c:\y2ktest. Because the attack files are moved or deleted, the batch file copies one of the y2ktest files three times and renames them, (check1, check2, check3) so there are still seven files in the y2ktest directory.  The attack modifies the registry but does not run netcat (winlog) right away. The backdoor does not take affect until the victim user logs out and logs in again. This increases stealthiness because the setup of the attack is split into two steps.  The attacker should wait a few hours or days before exploiting the backdoor so that it is more difficult for the victim to connect the breakin with the setup. NetCat can use any port, but if it uses port 23, all telnet sessions to the victim will be unauthenticated.  **Verification** After the attack has completed, the victim machine should be remotely accessible without authentication: On a remote machine, type the command  nc -v 172.16.112.100 53  **Cleanup** Delete the winlog command from the Run registry key and remove winlog from the process table, if it is there.  **Attack signature** When the attacker connects to the backdoor, sniffing will reveal what looks like a telnet session, but it does not begin with a login name and password and it occurs on port 53.  The security audit log will show the execution of REGEDIT (trojan runs), later followed by the execution of winlog.exe (backdoor activated). |
| **Phf**  R-b-U | **Description** The Phf attack abuses a badly written CGI script to execute commands with the privilege level of the http server. Any CGI program which relies on the CGI function escape\_shell\_cmd() to prevent exploitation of shell-based library calls may be vulnerable to attack. In particular, this vulnerability is manifested by the "phf" program that is distributed with the example code for the Apache web server [11].  **Simulation Details** The Phf attack is quite simple to implement because it requires only the ability to connect to a network socket and issue an http request. Within the simulation, the Netcat [31] program was used to generate this http request. Although this vulnerability allows an attacker to run any command on the server, the command used throughout the simulation was '/bin/cat /etc/passwd'. Using the Phf attack with this command reveals the contents of the victim system's password file to users with no account on the victim machine.  **Attack Signature** To find the Phf attack, an intrusion detection system can monitor http requests watching for invocations of the phf command with arguments that specify commands to be run. Examples of commands that an attacker might attempt to execute by exploiting the phf exploit are: cat /etc/passwd, id, whoami, or xterm. |
| **ppmacro**  R--User | **Description** This Remote to Local attack uses a trojan PowerPoint macro to read secret files. This attack is based on a particular scenario. The victim user usually receives PowerPoint templates from an outside source via email attachment. He runs a built-in macro which inserts a graph displaying web statistics, saves the presentation as a ppt file, and posts it on the web.  The attacker writes a fake email and sends the template with additional code appended to the macro. The attack code reads secret files from the victim machine (in d:\home\secret\) and inserts them as white text in the background on the master slide of the presentation. When the presentation is posted on the web, the attacker can examine the ppt file to reveal the text of the secret file. The macro also stores a counter in the victim's registry so each time the user runs the macro, a different file from the secret directory is inserted into the presentation.  **Simulation details** One expect script, ppmacro.exp, creates and sends the email to an NT victim. The victim must manually open the email, create the PowerPoint presentation, and post it on the web. On a Unix attacker: "/.sim/bin/sendmail.pl ppatack.txt " The victim must then run Wusage (creates web statistics), rename one of the graphs in c:\winnt\reports to graph.gif, run PowerPoint, run the macro, and save the ppt file somewhere in c:\inetpub\wwwroot. The attacker later uses Netscape to download the ppt file.  **Verification** After the attack has completed, the attacker should be able to view the secret file by downloading the ppt file from the web. Run strings on the file to see the secret file text.  **Cleanup** To fully cleanup, delete the webstats key from the victim's registry. This is not recommended because editing the registry is detectable.  **Attack signature** The sniffed data will reveal the secret file text being transferred when the attacker downloads the ppt file from the web. The attacker could modify the macro to encrypt the secret file thereby making the attack more stealthy. Auditing reveals nothing about the attack. |
| **Sendmail**  R-b-S | **Description** The Sendmail attack exploits a buffer overflow in version 8.8.3 of sendmail and allows a remote attacker to execute commands with superuser privileges. By sending a carefully crafted email message to a system running a vulnerable version of sendmail, intruders can force sendmail to execute arbitrary commands with root privilege [15].  **Simulation Details** Although this vulnerability was widely known about at the time of the evaluation, no code that exploited this vulnerability had been posted to public forums such as Bugtraq, or Rootshell.com. A significant period of time (one person working for more than two weeks) was spent developing the first known implementation of this exploit explicitly for use in the evaluation. The implementation consists of a carefully constructed mail message which, when sent to the victim machine with a vulnerable version of sendmail, adds a new entry with root privilege to the end of the password file on the victim system. Once this new entry has been added, the attacker can log into the machine as this new user and execute commands as a root user. Figure 8-3 provides an illustration of an instantiation of the Sendmail attack as it was implemented in the simulation. In step 1 of this illustration, the attacker sends a carefully crafted e-mail message to the victim machine. In step 2, the sendmail daemon starts to process this message, overflows one of its buffers, and executes the attacker's inserted commands that create a new entry in the password file. In step 3, the attacker comes back to the victim machine and uses the new password file entry to gain root access to the victim machine and perform some malicious actions.  **Attack Signature** The Sendmail attack overflows a buffer in the MIME decoding routine of the sendmail program. In order for an intrusion detection system to identify a Sendmail attack it must monitor all incoming mail traffic and check for messages that contain a MIME header line that is inappropriately large. |
| **sshtrojan**  R-a-Deny (Temp./Admin.) | **Description** In SSH Trojan attack, the attacker tricks the system administrator into installing (as a "Y2K Upgrade") a trojan version of the SSH program. This trojan version allows the attacker (or anyone!) to login to the victim, via ssh, with the login "monkey" and no password. Upon login, a root privilege shell is spawned for the attacker.  **Simulation Details** During the setup phase of the attack an email is sent to the system administrator advising that he/she should install a particular y2k compliant version of the ssh server software that they use. The admin. then installs the software, thereby replacing the good version of sshd with the trojan version.  During the breakin phase, the attacker uses ssh to login to the victim via ssh as user "monkey". The login is sucessful, does not require a password, and yields a rootshell for the attacker.  **Attack Signature** The setup phase of the attack could be detected perhaps by identifying the initial email message as a hoax, or by detecting the download of sshd program software from a suspicious site. As the breakin part of the attack is carried out in an encrypted session there is little evidence on the network. However if host-based auditing is availble for the victim host, one might be able to detect the root shell spawned by ssh. (in the 1999 Lincoln Eval however, the attack was run against Linux, where no host auditing was available.) |
| **Xlock**  R-cs-Intecept (Keystrokes) | **Description** In the Xlock attack, a remote attacker gains local access by fooling a legitimate user who has left their X console unprotected, into revealing their password. An attacker can display a modified version of the xlock program on the display of a user who has left their X display open (as would happen after typing 'xhost +'), hoping to convince the user sitting at that console to type in their password. If the user sitting at the machine being attacked actually types their password into the trojan version of xlock the password will be sent back to the attacker.  **Simulation Details** This attack was created specifically for use in the evaluation. A special version of the xlock program was created by making small modifications to the publicly available source code for the xlock program. The standard version of xlock will not display on a remote display, and does not save or output the password that the user enters. A modified version of xlock was created that allows the attacker to display the screen saver on a remote display and returns the password entered by the victim. Because of the required interaction with the X server, the Xlock attack was always run by a human actor.  **Attack Signature** Two factors make this attack quite difficult for an intrusion detection system to identify. First, this attack is a spoofing attack that does not abuse a bug in the system, but relies on assumptions by the person who is currently using the system. Second, this attack is embedded in the X protocol, and an intrusion detection system that is trying to identify this attack must understand and parse these X communications in order to identify the Xlock attack. One non-optimal way of dealing with these difficulties, is to program an intrusion detection system to identify as suspicious any X traffic that is originating from an unknown machine that is destined for a machine being monitored. |
| **Xsnoop**  R-c-Intercept (Keystrokes) | **Description** In the Xsnoop attack, an attacker watches the keystrokes processed by an unprotected X server to try to gain information that can be used gain local access the victim system. An attacker can monitor keystrokes on the X server of a user who has left their X display open. A log of keystrokes is useful to an attacker because it might contain confidential information, or information that can be used to gain access to the system such as the username and password of the user being monitored.  **Simulation Details** The Xsnoop program used in the simulation was adapted from C code originally posted to the Bugtraq mailing list. The Xsnoop program runs locally on the attacker's machine. The program connects to the victim's X Server and requests to be notified of all X KeyPress Events. If the attacker has permission to make this request (as would happen if the victim had typed 'xhost +', or if the victim's X Server is configured to allow connections from all hosts by default) the Xsnoop program will be sent all KeyPress events that occur on the victim X Server. The Xsnoop program then uses the KeyPress events to provide the attacker with a view of all the keystrokes the victim. Because this attack required interaction with the X Server, it was always run by a human actor.  **Attack Signature** An network based intrusion detection system can identify the Xsnoop attack by parsing the X protocol information in packets destined for remote X clients and noting that X KeyPress events are being transmitted to a remote machine. Because the Xsnoop attack results from bad security policy (leaving an X Server unsecured) and not simply a bug, the presence of these packets alone does not signify that an attack is taking place. The intrusion detection system must know whether the security policy of the machine being monitored allows unauthenticated X connections from anywhere. |

### **Probes**

In recent years, a growing number of programs have been distributed that can automatically scan a network of computers to gather information or find known vulnerabilities [27]. These network probes are quite useful to an attacker who is staging a future attack. An attacker with a map of which machines and services are available on a network can use this information to look for weak points. Some of these scanning tools (satan, saint, mscan) enable even a very unskilled attacker to very quickly check hundreds or thousands of machines on a network for known vulnerabilities. Figure 9-1 provides a summary of the probes that are discussed in the remainder of this chapter. The following sections describe in detail each of the probes that was used in the 1998 DARPA intrusion detection evaluation.

|  |  |
| --- | --- |
| **insidesniffer**  P-a-Probe (Machines, Users) | **Description** Here the attacker merely attachs a new machine to an inside ethernet hub, configured with an ip, and begins sniffing traffic.  **Simulation Details** The attack can be carried out in two ways: a fairly "obvious" way to sniff is to allow/request that the sniffing software attempt to resolve (using DNS) the IP addresses to names, while a stealthier way is to disable this DNS lookup.  **Attack Signature** In the "clearer" case above, the Intrusion Detector could look for traffic to/from a completely new ip address, and that the traffic would consist entirely of dns queries for addresses from which it had just seen packets! In the stealthier case, we have not identified a foolproof way to detect the sniffer, given the "off-line" style of these evaluations. Some abnormalities might exist - in particular - if the sniffer was assigned an ip address, a subsequent ipsweep might reveal the presense of the new machine - however this might or might not indicate that the interface was in promiscious-mode. |
| **Ipsweep**  R-a-Probe (Machines) | **Description** An Ipsweep attack is a surveillance sweep to determine which hosts are listening on a network. This information is useful to an attacker in staging attacks and searching for vulnerable machines.  **Simulation Details** There are many methods an attacker can use to perform an Ipsweep attack. The most common methodùand the method used within the simulationùis to send ICMP Ping packets to every possible address within a subnet and wait to see which machines respond. The Ipsweep probes in the simulation were not stealthy - the sweeps were performed linearly, quickly and from a single source.  **Attack Signature** An intrusion detection system looking for the simple Ipsweep used in the simulation can look for many Ping packets, destined for every possible machine on a network, all coming from the same source. |
| **ls\_domain**  R-a-Probe (Network, Machines) | **Description** Here the attacker uses the "nslookup" command in interactive mode to "list" all machines in a given DNS domain from a mis-configured primary or secondary DNS server. Thus the attacker can learn what machines (IP addresses) belong to (and perhaps exist in) the domain.  **Simulation Details** The attacker (scripted with expect) performs the above operation and stores the knowledge in a file. Later, the file is opened, and each ip address is probed in some fashion. In the 1999 Lincoln Evaluation, the subsequent probe was an nmap probe of port 80 on each listed host.  **Attack Signature** The Detector might look for, first the domain listing process, and then will see the probe of all ip's listed in the DNS record. In particular, the attacker will probe machines that don't exist, but are listed. |
| **Mscan**  R-a-Probe  (Known Vulnerabilities) | **Description** Mscan is a probing tool that uses both DNS zone transfers and/or brute force scanning of IP addresses to locate machines, and test them for vulnerabilities [20].  **Simulation Details** The Mscan program used in the simulation was compiled from source code found at [57]. Mscan was easy to run and has several command line options for specifying the number of machines to scan and which vulnerabilities to look for. Within the simulation, mscan was used to scan the entire eyrie.af.mil domain for the following vulnerabilities: statd, imap, pop, IRIX machines that have accounts with no passwords, bind, various cgi-bin vulnerabilities, NFS, and open X servers.  **Attack Signature** The signature of this attack will vary depending on which vulnerabilities are being scanned for and how many machines are being scanned. In general, an intrusion detection system can find an Mscan attack by looking for connections from a single outside machine to the ports listed above on one or more machines within a short period of time. |
| **NTinfoscan**  R-a-Probe  (Known Vulnerabilities) | **Description** NTInfoScan is a NetBIOS based security scanner. It scans the NT victim to obtain share information, the names of all the users, services running, and other information. The results are saved in an html file named .html where victim is the victim's hostname.  View the page from mnemonix for more information.  **Simulation details** A Perl script runs on an NT attacker. Edit the first line of the ntis.pl with the time of day the attack should run and then run ntis.pl or put it in the Startup group and restart the machine. Ntis.pl automatically scans hume.eyrie.af.mil. The attack may take up to 20 min. to complete.  **Verification** There will be a file named hume.eyrie.af.mil.html in c:\.sim\attacks\logs (make sure the last modified date agrees with the date the attack was launched). Open the file to verify that data was collected by the scan.  **Attack signature** Sniffing reveals that the attack FTPs to the victim as user anonymous with password guestaccnt@compuserve.com and makes numerous HTML GET requests to files in such directories as /cgi-bin and /scripts. Originally, the ntis ftp'd to the victim with the password, ntinfoscan.  The security audit log can also be used to detect the attack. A login by IUSR via Advapi, followed by the execution of newdsn.exe by SYSTEM indicates a web scan. A login via KsecDD followed by multiple SAM\_USER accesses by SYSTEM indicates a netbios scan.  **Problems and solutions** Often, ntis will temporarily hang during the web services portion of the attack if it attempts GET request for inaccessible files. There is a timeout of 15min, after which the attack will complete. |
| **Nmap**  R-a-Probe (Services) | **Description** Nmap is a general-purpose tool for performing network scans. Nmap supports many different types of portscansùoptions include SYN, FIN and ACK scanning with both TCP and UDP, as well as ICMP (Ping) scanning [45]. The Nmap program also allows a user to specify which ports to scan, how much time to wait between each port, and whether the ports should be scanned sequentially or in a random order.  **Simulation Details** At the time of the evaluation, Nmap was the most complete publicly available scanning tool. During the simulation, Nmap was used to perform portscans on between one and ten computers using SYN scanning, FIN scanning, and UDP scanning of victim machines. Both sequential and random scans were performed in the simulation, and the timeout between packets was varied to be anywhere from one second to six minutes. The number of ports scanned on each machine was varied between three and one thousand.  **Attack Signature** The signature of a portscan using the Nmap tool varies widely depending on the mode of operation selected. Despite this variance of modes, all portscans share some common features. A portscan can be recognized by noting that network packets (whether via TCP or UDP, or via only FIN packets or only SYN packets) have been sent to several (or more) ports on a victim or group of victims within some window of time. Two factors complicate the identification of portscans. First, a portscan can happen very slowly. An attacker who is patient could probe one port per day. Current intrusion detection systems do not keep enough state to recognize a portscan happening over such a long period of time. With the amount of network traffic sent over a typical network, keeping enough active state within the intrusion detection system to recognize one connection per day for one hundred days as a one hundred day long portscan is simply not practical. Second, the connections don't necessarily all have to come from the same host. A group can perform a coordinated scan with each member scanning only a subset of machines or ports. By combining these methods, a group could perform a 'low/slow' portscan that would be very hard to recognize. |
| **queso**  R-a-Probe | **Description** QueSO is a utility used to determine a what type of machine/operating system exists at a certain IP adress. QueSO sends a series of 7 tcp packets to any one port of a machine and uses the return packets it receives to lookup the machine in a database of responses.  **Simulation Details** To make the attack more stealthy, we increased the delay between sending the packets. In the 1999 DARPA evaluation machines are sent the 7 QueSO packets with delays of 3, 5, and 10 minutes.  **Attack Signature** Since the time window in between packets can be fairly large, QueSO can be difficult to detect. The first 4 packets are normal requests to open and close a connection. The remaining 3 packets are abnormal requests looking for anomolous behavoir to help classify the machine and OS. The abnormal packets will flag systems looking for odd combinations of TCP flags or attempts to use TCP reserve bits. |
| **resetscan**  --Probe | **Description** ResetScan sends reset packets to a list of IP addresses in a subnet to determine which machines are active. If there is no response to the reset packet, the machine is alive. If a router or gateway responds with "host unreachable," the machine does not exist.  **Simulation details** A Perl script runs on a Linux attacker. ResetScan.pl reads a file containing a list of IP addresses and runs rscan with each address. After one minute, resetscan.pl runs rscan with each IP address again. The first run and the pause are necessary to give the gateway time to determine active hosts. The second run produces the actual results of the attack.  **Verification** The dump file will show reset packets sent to any of the IP addresses scanned that actually exist. In addition the tcpdump file will contain "arp who-has" requests for the range of ip's scanned that don't exist (from the first round of resets).  **Attack signature** Sniffing reveals reset packets sent to IP addresses with no previous connections, as well as a numerous "arp who-has" packets sent by the gateway, requesting mac addresses for ranges of non-existant ip addresses. |
| **Saint**  R-a-Probe  (Known Vulnerabilities) | **Description** SAINT is the Security Administrator's Integrated Network Tool. In its simplest mode, it gathers as much information about remote hosts and networks as possible by examining such network services as finger, NFS, NIS, ftp and tftp, rexd, statd, and other services. The information gathered includes the presence of various network information services as well as potential security flaws. These flaws include incorrectly setup or configured network services, well-known bugs in system or network utilities, and poor policy decisions. Although SAINT is not intended for use as an attack tool, it does provide security information that is quite useful to an attacker [58]. SAINT is distributed as a collection of perl and C programs and is known to run on Solaris, Linux, and Irix systems. Within the simulation, the Saint program was run from a Linux traffic generator and was used to probe several victim machines for vulnerabilities. SAINT's behavior is controlled by a configuration file which allows the user to specify several parameters. The most important parameters are the list of machines to scan, and how heavily to scan these machine (light, normal, or heavy). In light mode, SAINT will probe the victim for dns and rpc vulnerabilities and will look for unsecured NFS mount points. In normal mode SAINT will also check for vulnerabilities in fingerd, rusersd, and bootd, and will perform a portscan on several tcp (70, 80,ftp, telnet, smtp, nntp, uucp) and udp (dns, 177) ports. Heavy mode is the same as normal mode except that many more ports are scanned. A Saint scan of a network leaves a distinct signature that will vary depending on the level of scanning being performed. The Saint program performs each scan in a nearly deterministic fashion. To identify a Saint scan, an intrusion detection system needs to be able to recognize the distinct set of network traffic the scan creates. Figure 9-2 is a plot that provides a graphical view of this signature. The horizontal axis of this plot represents time in minutes, and the different services probed are presented along the vertical axis. The names of the services are shown on the left side of this plot, and connections for each service are plotted within each named region. The numbers after the service names are the number of separate tcp connections or of udp or icmp packets. Names ending in '/i' indicate that packets use the icmp protocol and names ending in '/u' indicate that packets use the udp protocol. All other services use the tcp protocol. Each line segment represents a connection to a service. This plot shows the unique signature of a medium level Saint scan. Because this signature does not change significantly across multiple instantiations of the Saint attack, an intrusion detection system that has been trained to recognize the pattern of connections shown in Figure  9-2 will probably detect other Saint attacks. |
| **Satan**  R-a-Probe  (Known Vulnerabilities) | **Description** SATAN is an early predecessor of the SAINT scanning program described in the last section. While SAINT and SATAN are quite similar in purpose and design, the particular vulnerabilities that each tools checks for are slightly different [24].  **Simulation Details** Like SAINT, SATAN is distributed as a collection of perl and C programs that can be run either from within a web browser or from the UNIX command prompt. SATAN supports three levels of scanning: light, normal, and heavy. The vulnerabilities that SATAN checks for in heavy mode are: ? NFS export to unprivileged programs ? NFS export via portmapper ? NIS password file access ? REXD access ? tftp file access ? remote shell access ? unrestricted NFS export ? unrestricted X Server access ? write-able ftp home directory ? several Sendmail vulnerabilities ? several ftp vulnerabilities Scans in light and normal mode simply check for smaller subsets of these vulnerabilities.  **Attack Signature** A SATAN scan of a network can be recognized by the consistent pattern of network traffic the program creates. The checks for the vulnerabilities listed above are always performed in the same order. Figure 9-3 shows a plot of the services probed during an example medium level Satan scan. The horizontal axis of this plot represents time in seconds and the various services that are probed are presented on the vertical axis. Each line segment in the plot represents a single connection to a service. Every medium level Satan scan will have a signature very similar to that shown in Figure 9-3. |

### **Data**

Data Attacks involve someone (user or administrator) performing some action that they may be able to do on a given computer system, but that they are not allowed to do according to site policy. Often, these attacks will involve transferring "secret" data files to or from sources where they don't belong.

|  |  |
| --- | --- |
| **Secret** | **Description** A "secret" attack is an attack where the attacker maliciously or mistakenly transfers data which they have access to to a place where it doesn't belong. For example, transferring data from a classified computer/network to a non-classified computer/network would constitute a "secret" attack.  **Simulation Details** We simulate these types of attacks by publishing a set of rules indicating that all files in a particular directory are "secret" and that they can not be moved out of that directory (wheather by 'cp', 'cat', 'ftp', or whatever.) Then the attacker logs in and performs such an action.  **Attack Signature** To recognize these attacks, the detection system must know which files are considered "secret", what the policies are regarding use of these files, and then simply look for actions carried out involving them. Naturally, attacks such as these can be hard to detect - a legitimate user could "cut-and-paste" information from one desktop window to another. |