Concepts in Computer and Network Insecurity*

Roger Dingledine, Kevin Fu

January 10, 2001

1 From Outside

1.1 Packet level protocol issues

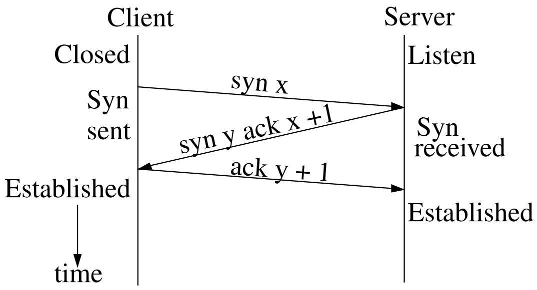

Figure 1: A typical setup of a TCP connection

The TCP protocol is a connection-oriented, reliable protocol built on top of the IP protocol. Connections are created by the three-way handshake illustrated in Figure 1. A connection is requested in step 1, when host A sends to host B a packet with the SYN flag turned on (a "SYN packet"). Host B responds with a packet with both SYN and ACK on (a "SYN/ACK packet"), and then host A completes the handshake using an ACK packet. Layered on top of this exchange is the use of what are called "sequence numbers" by both sides. Sequence numbers are used to describe order of packets, so both sides can take steps to make sure that all the data is getting through in the right order. A diagram of the TCP header follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: TCP Header

Specifically, the sequence number in a packet corresponds to the first byte of data in that packet. The acknowledgment number, which is a separate 32-bit element in the header, holds the value of the next *expected* sequence number[1], implicitly acknowledging all data up until that number.

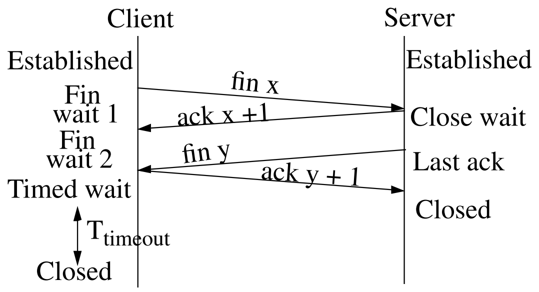

Figure 3: A typical tear down of a TCP connection

In addition to SYN and ACK, there are other flags: FIN ("I would like to close this connection now"), PSH ("please push all the data currently queued"), URG ("This message is urgent, please interrupt the current connection" – note that the response to this is highly implementation-dependent), and RST ("something has gone wrong, please reset this connection").

There are a number of possible ways to exploit the TCP protocol, either to deny service from a legitimate user or to obtain unauthorized privileges.

1.1.1 IP spoofing

In general, an attacker can claim to be from a different IP than he actually is, by setting the packet source address to the IP address of a different (possibly non-existent) host. This makes it easier to convince

[1] http://packetstorm.securify.com/mag/phrack/phrack48/P48-14

the target of a different identity in certain trust-based attacks (see below under TCP hijacking and rlogin attacks), and also makes it more difficult to trace the source of an attack. Note that if the attacker does spoof his IP, he needs to either be good at guessing responses, or have some way of intercepting or overhearing the response packets.

1.1.2 SYN flooding

A computer keeps track of pending connections (connections in the "SYN received" state in Figure 1) in a buffer. This buffer generally remembers between 5 and 15 simultaneous connections (it's system-dependent). Connections are removed from the queue as they are processed, or after a timeout period (generally 75 seconds). If more SYN packets arrive while the buffer is full, these packets are dropped (since the TCP protocol is only best-effort, it just assumes that the connection attempt will be retried).

Note that the attacker needs to spoof the source IP for the SYN packets to be some unreachable host – the target machine will respond with a SYN/ACK, and if that SYN/ACK arrives at a real host then that host would respond with a RST (it wasn't expecting to create a connection). This RST will remove the connection from the queue[2].

Note also that this requires relatively little bandwidth to the target, unlike more convention flooding attacks (e.g., ICMP floods), which rely on being able to overpower the bandwidth or other resources of the target through sheer force of incoming packets[3].

An effective solution to this problem is to use 'SYN cookies', which allow the machine to not have to remember an incoming connection until the handshake is completed. This is done by using some cryptographic hash (including the IP numbers and ports involved and some server-known secret) as the sequence number. If the ACK in the third phase of the handshake has the correct hash value, then the host can assume that the first two phases of the handshake were performed correctly.

1.1.3 Full connection flooding

Inetd is typically susceptible to a denial of service (DoS) attack involving many consecutive connects to the same service; this is sometimes called a "flooding" attack. When too many connections are made within a specified short period of time, inetd will terminate that service for a short period of time (typically five minutes), and print an error message on the system error log of the form:

```
inetd[345]: telnet/tcp server failing (looping), service terminated
```

There are attack programs, such as octopus[4], which are capable of opening and maintaining hundreds or even thousands of connections at once. Octopus will reliably terminate services that inetd controls, and for stand-alone daemons it can be used to overflow the target's process table or file descriptors. Indeed, it's particularly effective against sshd and sendmail, because they fail to exit cleanly when the descriptor table is full or nearly full.

Tying up the resources of the target machine is a very effective means of attack. Since it's working within the intended specifications of the protocol, there's nothing to 'fix' – and it's difficult to defend against, because most solutions tend to involve potentially reducing convenience or accessibility for legitimate users.

1.1.4 Sniping

If an attacker has the ability to read the packets of a TCP connection (that is, if he owns a machine local to either party or on the route between them), then he can keep track of the sequence numbers that both sides are using. From there, it's a simple matter to send a RST packet with the correct sequence number (from the right IP, of course). This will close down one side of the connection; when data is sent from the other side, a RST packet will be generated (data is being sent to an already-closed connection – this is considered an error).

[2] `http://packetstorm.securify.com/mag/phrack/phrack48/P48-13`
[3] `http://advice.networkice.com/Advice/Exploits/TCP/SYN_flood/default.htm`
[4] `http://www.hoobie.net/security/exploits/hacking/octopus.c`

1.1.5 Hijacking

More complex than simply sniping the connection is hijacking the connection. By knowing the current sequence numbers, an attacker can snipe one end of the connection and take over talking to the other side as if nothing had changed.

In a more sophisticated attack, the cracker spoofs the hardware address ("ARP spoofing"[5]) of one side, to convince the target that that side has changed hardware addresses. This causes the target machine to send its replies to this (new, possibly non-existent) address. This means that you don't have to snipe one of the connections, and you can conceivably reconnect the original client later by correcting the ARP tables and sending acknowledgments with updated sequence numbers [6].

1.2 Lower-level Protocols

1.2.1 Packet Fragmenting

While packet fragmentation attacks[7] are actually an IP-layer vulnerability (much lower-level than the TCP-layer vulnerabilities), they're still worth mentioning. Each packet has a TCP header as shown in Figure 2, but it also has an IP header which specifies the IP of the source and destination, how many hops the packet should travel before it gets dropped, etc. One of the options that can be set in the IP header is whether this packet is fragmented – often a router will need to fragment a packet to allow it through that section of the network, for instance because the gateway connects an ethernet network (where datagram size is typically limited to 1500 bytes) to an ATM network (where datagram size is limited to 56 bytes).

The 'offset' field in the header is used to put all the fragments of a datagram back together. One simple but effective attack that exploited a bug in a surprisingly large number of TCP implementations a few years ago was to provide faulty (for instance, negative) offset numbers in the packets. When reassembling the packet, the target machine would accidentally overwrite some other part of its kernel memory, causing arbitrary reboots or freezes.

1.2.2 ICMP exploits

The Internet Control Message Protocol (ICMP) is another protocol on top of IP (just like TCP is a protocol on top of IP). ICMP is used to communicate information about network topology changes, routing suggestions, etc. Common ICMP packet types are 'destination unreachable', 'source quench', 'redirect message', and 'routing advertisement'. ICMP packets are notorious for their lack of authentication, allowing a wide variety of exploits and attacks. However, most routers these days talk to each other using Border Gateway Protocol (BGP) or similar routing protocols, rather than using ICMP.

1.3 More Packet-level Issues

1.3.1 Probing

In addition to exploiting the TCP protocol to deny service or take over a connection, you can also use it to gain more information about a target machine. Following are a number of different types of probes[8].

- Simple probing. By making a connection to a port on the target's machine (that is, completing the TCP handshake described in Figure 1), you can determine if this port is listening. On the other hand, the target will be able to easily identify (and log) the fact that you made that connection.
- A slightly more subtle approach is to send just the initial SYN packet. If you receive a SYN/ACK, then chances are good that the target is listening on that port. On the other hand, if you receive a RST packet, the port is probably closed. By not responding to the SYN/ACK, you make it less likely that the target machine will log your probe.

[5] `http://staff.washington.edu/dittrich/papers/arp_fun.txt`
[6] `http://www.insecure.org/stf/iphijack.txt`
[7] `http://all.net/journal/netsec/9509.html`
[8] `http://packetstorm.securify.com/mag/phrack/phrack51/P51-11`

- FIN scanning: send a FIN packet to the port in question. If the port is closed, generally the target will be polite enough to respond with a RST. If the port is open, however, generally the target will silently drop the FIN. This approach to probing is even less likely to be logged; however, note that if your FIN packet is dropped en route to the target (either because of firewall filtering or because of general network congestion), you will falsely conclude that the port is available.
- ICMP echo reply scanning: because many firewalls block incoming ICMP echo requests (pings), scanning machines behind a firewall by ping doesn't work very well. But since the firewall configuration generally wants users behind the firewall to be able to ping *out, then ICMP echo replies (answers to pings) are not filtered. By sending echo replies to various hosts behind a firewall, it's possible to determine which of these hosts exist (are reachable) and which do not exist – for each host which doesn't exist, some router along the way is kind enough to send back an ICMP unreachable packet.*

1.3.2 Logging

Of course, there are programs (known broadly as Intrusion Detection Systems) designed to listen for these sorts of probes (as well as other attacks) and notify the host's administrators. If the target machine is very lightly loaded, then it will be difficult to do probing without getting noticed. However, if the target has thousands of connections a minute, it should be pretty straightforward to get a few packets in without tripping any alarms.

There are some techniques for making it more difficult to track you. For instance, spreading the probes out over the space of several hours, and doing the probes from multiple machines (if you have the resources) is generally safer. Taking this to an extreme, you might consider doing a thousand identical port scans, each spoofed from a different IP. Most of the responses are sent off to either a nonexistent or unsuspecting host, but you still get the results of the port scan (either because you can monitor the network to which one set of responses was sent, or because one of the original thousand machines was actually one of yours). In either case, it's difficult to track down which (if any) of a thousand machines is actually doing the probing.

Another technique for making it more difficult to detect your attacks is to make use of packet reassembly (see the section above on fragmentation attacks). It's very tricky for an intrusion detection system to monitor and analyze all fragments coming through a high-traffic gateway.

1.3.3 Sniffing

So far, we've been discussing sniffing from the point of view of gathering information from the packet headers. In addition to headers, datagrams generally have something more interesting as well: data. The bodies of the first few packets in a telnet or ftp session can often yield a username/password pair, allowing for simpler methods of gaining unauthorized access to the target.

1.4 Up a layer of abstraction

Of course, the above are just some of the attacks you can make against the underlying TCP protocol. There are a number of other interesting ways besides sniffing and low-level probing to gain information about a computer system.

- Finger: you can gain a surprising amount of information from simply fingering a machine to see the current logins, and sometimes personal information about users.
- TCP fingerprinting: programs like queso (and now nmap) can identify the operating system of the remote machine, narrowing down the types of attacks that are likely to succeed, and also giving you a good idea of the size of the incoming connections buffer (which makes SYN flooding more effective).
- Banners: you can learn a lot from a machine by simply connecting to its ftp port, telnet port, sendmail port, httpd port, etc and collating the information. Often sendmail will have a much better idea of the target machine's real name than the DNS tables do, especially if the machine is configured poorly. The telnet banner will often tell you the operating system flavor (including distribution) and architecture of the target.

- Port scanning can give you a good idea of the purpose of the machine. By noting which services are offered, you might notice patterns and make a good guess at the machine's OS and configuration. In addition, commands like `showmount -e` will list NFS exports; keep an eye out for mounts that are exported read/write.

- Though this doesn't count as a probe, echo port attacks exploit a relatively high-level protocol to deny service. The echo service (port 7) simply echoes whatever text is sent to it. If you convince two machines that they are both talking to each other on port 7, they will keep the conversation going until enough packets are dropped. More directly, if you send a packet to a target IP spoofed from that same IP, the machine may well spend all of its resources in a tight loop echoing to itself. This bug is fixed in most modern systems.

- DNS Spoofing: name servers tend not to authenticate connections as strongly as they should (routers have this same problem – it stems from the fact that they need to support as wide a variety of systems as possible, and some of those systems don't have a good way of authenticating in the first place), which means that attacks are often possible that add or change a line in the DNS tables. Specifically, you might convince a local name server that `www.microsoft.com` points to a different place. At the least, you have just denied service from normal Microsoft customers. A more subtle attack would be to mirror Microsoft's page onto your fake one, modifying the service packs or other patches to have bugs or backdoors.

1.5 Direct attacks

Rather than exploiting some protocol to gain information or deny a service, there are a number of exploits which attempt to actually obtain control of a machine.

- rlogin: by guessing the correct sequence number (in some TCP implementations this is relatively easy to do), an attacker can take advantage of trust relationships between computers. If one computer allows rlogin without a password from a certain domain, the attacker can spoof his IP to be that of a machine without that domain; even if he can't read the response packets, he can login and blindly open up another hole in the system or perform other actions.

- cgi: early Apache web servers (and others) included cgi programs such as `phf`, `handler`, etc. These programs were intended to make certain operations and interactions with the server more convenient, but they also included a number of security holes that would allow an attacker to request a copy of the password file, or sometimes even execute arbitrary commands (under the permissions of the web server, which were typically `nobody` but sometimes `root`) on the remote machine.

- Other remote services that are frequently exploited include sendmail (this program is enormous, and contains a very wide variety of vulnerabilities), amd and mountd (a couple of remote buffer overflow bugs have been found recently in both of these), ftp (a remote buffer overflow bug was found recently that involved creating long path names in the `incoming` directory), imapd and popd, etc. Exploits on RPC services on Suns have been common in the past year.

2 Viruses/Worms

Viruses are malicious programs which are spread through carelessness and faulty software. The virus actually embodies a theorem from complexity theory: the recursion theorem[9]. It basically states that a Turing machine can obtain a copy of its own program. In this way, a virus obtains a copy of itself to spread to other machines. Related to the virus is the worm. One of the most famous security incidents on the Internet was the worm of 1988. Those who attend this security seminar will get to watch a short video about the Internet Worm.

[9] M. Sipser. Introduction to the Theory of Computation, p. 200

3 Firewalls

One of the more heated arguments is whether to use firewalls. In fact, there are a number of books and courses just on firewalls[10]. As such, we will only take a cursory glace at the fundamental concepts of firewalls.

Many people misunderstand the purpose of a firewall. A firewall obeys the set of rules you lay down. If you do not set the rules properly or the rules do not correspond with the policy you wish to maintain, the firewall is not useful. When the ILOVEYOU virus invaded Dow Jones in Hong Kong, Daphne Ghesquiere responded, "I have no idea how it got through the firewall. It's supposed to be protected."[11] Her comment demonstrates the general misunderstanding about firewalls. People mistakenly believe that firewalls can automatically ward off all evils.

A firewall enforces an access control policy between two networks[12]. For instance, a firewall could be implemented at a gateway machine to the Internet from your local network. Or you could implement firewall technology inside your local operating system (e.g., the UNIX tcpwrappers).

Memory is a primary factor in firewalls. Because firewalls typically keep track of a lot of state (all active connections), a firewall needs a lot of memory to diffuse denial of service attacks.

There are three partitions created by a firewall: the external network (e.g., Internet), the Demilitarized Zone (DMZ) where the firewall exists, and the internal network.

In a corporate environment, firewalls may be an appropriate method of reducing the number of ways to breakin. In an academic environment, firewalls will likely impede the development of new Internet technology.

Now let's discuss a few classical ways of breaking through firewalls. Sophisticated firewall software should be resistant to these attacks (we hope!).

3.1 Fragmentation

A devious attack exploits fragmentation to get past firewalls. Many firewall programs block incoming packets based on their destination port. For instance, they might let in packets to port 80 (canonically the httpd port) but disallow packets to port 23 (telnet). Since TCP specifies that packets should only be reassembled at their final destination (otherwise they would be getting fragmented and reassembled many times during transmit), these firewall implementations tend to decide whether to let a set of fragments through based on the first fragment (the one that includes the destination port). However, if a later fragment specifies an offset that overwrites that header, once the packet is reassembled past the firewall it will suddenly claim to be destined for a port that should have been blocked.

3.2 FTP PORT hole

FTP uses two TCP sockets. One for the control (e.g., commands like `put` and `get`) and one for the data transferred. FTP clients setup the data socket in one of two ways: PORT mode or PASV mode. In PORT mode, the FTP client listens to a socket and tells the FTP server to connect to this socket. That is, the FTP client is now acting like a server. In PASV mode, the FTP client tells the server to listen on another port for data. Then the FTP client connects to this port[13].

A problem arises when a firewall wants to block incoming packets to random ports. FTP clients inside firewalls will not work when connecting to external FTP servers if the firewall blocks non-standard ports. Some firewalls used to look for PORT mode ftp clients.

From RFC959:

```
DATA PORT (PORT)

   The argument is a HOST-PORT specification for the data port
   to be used in data connection.  There are defaults for both
   the user and server data ports, and under normal
```

[10] Cheswick, William R., and Bellovin, Steven M. *Firewalls and Internet Security: Repelling the Wily Hacker.* New York: Addison-Wesley Publishing Company, 1994.

[11] `http://cms.syr.edu/downtimes/2000-05-04_iloveyou.html`

[12] Firewall FAQ by Marcus Ranum and Matt Curtin `http://www.interhack.net/pubs/fwfaq/`

[13] `http://web.mit.edu/rfc/rfc959.txt`

```
      circumstances this command and its reply are not needed.  If
      this command is used, the argument is the concatenation of a
      32-bit internet host address and a 16-bit TCP port address.
      This address information is broken into 8-bit fields and the
      value of each field is transmitted as a decimal number (in
      character string representation).  The fields are separated
      by commas.  A port command would be:

         PORT h1,h2,h3,h4,p1,p2

      where h1 is the high order 8 bits of the internet host
      address.

   PASSIVE (PASV)

      This command requests the server-DTP to "listen" on a data
      port (which is not its default data port) and to wait for a
      connection rather than initiate one upon receipt of a
      transfer command.  The response to this command includes the
      host and port address this server is listening on.
```

Researchers found a more interesting attack by launching FTP clients via Java applets[14]. If a user behind a firewall visited a web page with such a malicious Java applet, the Java applet could open up any port on the firewall.

In a completely unscientific test, we found that only a few FTP clients operate in passive mode by default: Athena FTP, OpenBSD ftp, and Netscape Communicator 4.61. Most FTP clients use PORT mode by default: Redhat 6.1 FTP client), SunOS 4.1.3 ftp, NetBSD ftp, Windows95 ftp, ncftp, wget, lynx, and xftp. This is why FTP can often not work by default through firewalls.

3.3 Tunneling

There exist TCP/IP tunnels for almost every transport imaginable (email, HTTP, floppy disks, modems, carrier pigeon, etc). It's virtually impossible to shut off such covert channels. Given any transport method, you can pretty much implement TCP/IP.

4 Web security

The Web is used not only for exchanging information, but also for conducting business. Despite the dot-bomb of this past year, e-commerce still thrives. But many sites have not implemented secure methods for e-commerce. This section discusses a few of these problems.

4.1 Client authentication

Users typically log into personalized Web sites by entering a username and password. This process is called authentication. A wide variety of sites use this style of authentication: banks/brokerages, subscription-based newspapers, chat sites, shopping, web-based email, etc. Unfortunately, the online community has not embraced a single standard for client authentication on the Web. In fact, most sites invent their own cryptographic protocols! Noticing this disparity, several graduate students from the MIT LCS Applied Security Reading Group[15] set out to analyze and categorize client authentication on the Web. This section discusses the pitfalls of such unverified homebrew schemes.

[14] Martin, David M., Rajagopalan, Sivaramakrishnan, and Rubin, Aviel D., "Blocking Java Applets at the Firewall," The Proceedings of the 1997 Symposium on Network and Distributed Systems Security. `http://www.cs.bu.edu/techreports/1996-026-java-firewalls.ps.Z`

[15] Applied Security Reading Group, `http://pdos.lcs.mit.edu/asrg/`

Client authentication on the Web is implemented by any of three basic mechanisms: SSL personal certificates, HTTP authentication, and proprietary schemes using cookies or hidden form variables.

SSL includes client certificates (or Personal Certificates) to authenticate clients to servers. For instance, MIT students use personal certificates to check grades, pre-register, etc. A client certificate contains a public key certified by a trusted third party (such as the MIT certificate authority). A user also has the private key associated with the client certificate. A user can authenticate by engaging in a challenge-response protocol with a server.

HTTP authentication is a very simple way to send a username and password to a Web server. Both the username and password are sent unencrypted to the server. This is just as insecure as plaintext telnet. However, it may be possible to use HTTP authentication securely over SSL.

The third category, proprietary schemes, compose the bulk of client authentication on the Web. In the seminar, we give examples of sites that were broken because:

1. SSL and plain HTTP do not compose: BankOnline.com

2. Letting clients name the price: Instant Shop

3. Security through obscurity: New England Bride

4. Predictable sequence numbers: fatbrain.com

5. Misuse of cryptography: WSJ.com

4.2 Secure Sockets Layer (SSL)

The Secure Sockets Layer (SSL)[16] can provide channel security. This includes confidentiality, authentication, and integrity protection. SSL does not address traffic analysis or trust management. You can read all about SSL in the IETF RFC[17] or Eric Rescorla's excellent book[18]. You can carry out some simple demonstrations on your own[19] including an SSL dump program[20].

SSL is not without its pitfalls. While most people believe that the SSL protocol itself is secure, many problems have appeared in implementations of SSL. Even if SSL achieves its security goals, problems can arise from faulty software or unintended uses. We will discuss the following case studies:

1. Neither Netscape or Microsoft have published their rule set for deciding which CA roots to include in browsers.

2. Certificate revocation. [Sun CA]

3. Random number generation. [Netscape]

4. SSL authenticates servers, not content. [Akamai]

5. Leaked HTTP traffic. [SprintPCS]

6. Compromised server private key → decrypt future and past traffic.

7. Caching sessions on IP address rather than hostname [Netscape/IE]. Demo exploit in seminar.

[16]The IETF calls the standardized version the Transport Layer Security (TLS).
[17]`http://web.mit.edu/rfc/rfc2246.txt`
[18]SSL and TLS, ISBN 0-201-61598-3
[19]`http://www.pdos.lcs.mit.edu/asrg/2000-11-13.html`
[20]`http://www.rtfm.com/ssldump/`

5 Distributed Denial of Service

Denial of Service (DoS) is a type of attack where an adversary blocks communication or depletes valuable computing resources. For instance, email bombing is a type of denial of service because it overloads mail servers. Distributed Denial of Service (DDoS) has the same effect, but its source is more widespread and hard to stop. It involves an adversary who gains access to many drones across the Internet. Often these machines are compromised boxes on university networks. The adversary then instructs the drones to attack a service. The drones can easily cloak their identity when certain ingress and egress filters are not in place[21].

6 User-level Local Attacks

In most cases, it's easier to obtain access to a target machine at the user level of privilege than the root level. (Often, this can be done by sniffing passwords and then simply logging in.) Once in, however, the attacker has a number of avenues available to 'break root' from a local shell. A few examples follow.

6.1 telnet ld_preload

In general, telnet reads an environment variable which can specify which dynamic libraries should be used on the remote side during the login process. In some older systems, these libraries are loaded by `telnetd` as root before control is transferred to the user. If the attacker puts a modified `libc` library on the target machine and then logs in specifying he wants that library as his `libc`, he can force the machine to execute instructions as root[22].

6.2 Dictionary Attacks

If the local password file is readable, the attacker can simply grab a copy and start an offline dictionary attack on it, hoping that root's password is something easily guessed.

One solution to this vulnerability is to use 'shadow' password files, where the actual process of comparing to see if a password is valid is done only by root, and therefore only root needs access even to the hashed version of the password[23]. This helps tremendously, but it is still open to vulnerabilities where the attacker can cause the process doing the password comparison to segmentation fault, leaving a copy of the shadow password file in the core image.

Another solution is to use a different scheme for encrypting passwords. For instance, Red Hat 6.x offers the option of using MD5 to produce the password hash, rather than the traditional DES. While this is still vulnerable to a dictionary attack, the MD5 hash allows for passwords longer than 8 characters, and also is a slower hash, making dictionary attacks more computationally challenging.

6.3 X-Windows key sniffing

Programs are available to monitor the X events that are sent for screen refresh, mouse movement and keyboard activity, etc. If a user is running X on the target machine and using an xterm, a local user can watch the events and record (for instance) one side of an outgoing telnet or ssh session (or with more difficulty, monitor both the keystrokes and the window updates and catch both sides of the conversation). Similarly, programs like `xwd` (X window dump) dump an image of an X window to a file; this can be done even if a different user owns the X server.

In addition to simply monitoring the X events, an attacker can also generate new X events and insert them into the stream. This means that the attacker can execute commands on xterms owned by other users, opening up a host of new vulnerabilities.

[21] `http://pdos.lcs.mit.edu/asrg/02-14-2000.html`
[22] `http://oliver.efri.hr/~crv/security/bugs/mUNIXes/telnetd.html`
[23] `http://linuxsavvy.com/staff/jgotts/hack-faq/hack-faq-a.html#a04`

If the attacker manages to run `xhost +` or an equivalent command on the target machine, all of these attacks can be done remotely as well. (This means that users of X should be careful about which machines they trust.)

This is solved in part by using 'X cookies', an authentication mechanism to make sure that only truly authenticated hosts can interact with the X events. The X authentication system is complex enough that holes show up every so often, though.

6.4 Buffer Overflows

A popular flavor of attack is to exploit a binary on the system that has been set 'SUID root' – this means that the program runs with root privileges, and by taking control of this program, the cracker can run any arbitrary code (e.g., a root shell). From there, it's trivial to modify the system to open other holes or modify data.

A number of SUID root programs exist on most UNIX machines. Such files on a Redhat 6.1 box non-exclusively include:

```
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/newgrp
/usr/bin/crontab
/usr/bin/zgv
/usr/local/bin/ssh1
/usr/sbin/usernetctl
/usr/sbin/inndstart
/usr/sbin/sendmail
/usr/sbin/smbmount
/usr/sbin/traceroute
/usr/sbin/userhelper
```

Should any one of these programs fall victim to a buffer overflow, any local user can become root. Worse, some programs allow adversaries to overflow buffers from remote. Such an attack took linux boxes running `mountd` by storm in 1998[24].

Buffer overflows typically exploit poor array bounds checking in SUID root programs.

```
int foo (char *oflow)
{
  char vulnerable_buf[4];  /* Overflow to return address */
  strcpy (vulnerable_buf, oflow);

  return 0;
}
...
void main (int argc, char *argv[])
{
  ...
  foo (argv[1]);
  ...
}
```

This essentially translates to the following pseudo-assembly code with the memory address on the left and the code on the right[25]:

[24] `http://www.cert.org/advisories/CA-98.12.mountd.html`

[25] This is not real assembly, but close enough to get the idea.

```
main:
          ...
main + 3: push (main + 5)            /* return address of return point */
main + 4: push [address of argv[1]]  /* pushes the argument on stack */
main + 5: jump foo                   /* call foo() */
          ...

foo:      sub sp, 0xF                /* Allocate 4 bytes for vulnerable_buf */
foo + 1: push (foo + 5)              /* return address after strcpy */
foo + 2: push (sp + 0x10)            /* address of vulnerable_buf */
foo + 3: push (sp + 0x14)            /* address of argv[1] */
foo + 4: jump strcpy                 /* boom */
foo + 5: pop                         /* Pop the args */
foo + 6: pop
foo + 7: pop ra                      /* Pop return addr (normally main + 5)
                                        but now overflowed */
foo + 8: jump ra
```

We take advantage of the stack containing both return addresses and data. The strcpy function copies the input (argv[1]) to vulnerable_buf. But this buffer is too small. Since strcpy does not check bounds, it continues to write over the top of the stack – including the return address! A cleverly constructed input can overwrite the return address. Hence, the call to `jump ra` can actually jump to arbitrary locations in memory such as root shell exploit code.

There are a number of solutions to make buffer overflows less of a risk. First of all, Linux has a kernel patch to make the stack non-executable. While not completely fool-proof (bad programming can still open some vulnerabilities), this reduces a lot of the threats posed by buffer overflows that exploit the stack.

Another solution comes from Stackguard[26]. Stackguard is a compiler (actually, it's a patch to gcc) which puts wrappers around procedure calls to verify that the return address on the stack is correct. Again, this does not provide 'complete' protection, but it's much better than nothing.

[26] `http://www.immunix.org/products.html#stackguard`

6.5 Race Conditions

Sometimes programs perform actions that introduce timing vulnerabilities. For instance, a common error is to create a temporary file in /tmp for brief use, and then remove it. In fact, one such vulnerability for Redhat 6.1's initscripts was posted in December, 1999[27]. The process of determining a good name for this temporary file is often not atomic, meaning an attacker can try to predict the filename that a program will choose, and put a symbolic link in place at exactly the right time. This symlink might cause the program to overwrite a sensitive file, or read a file the attacker shouldn't have privileges to.

Matt Bishop gives an example in his race conditions paper. If the following program were SUID root, an attacker would have a window of opportunity between the access() and open calls to guess the temporary filename and symlink the filename to some privileged part of the disk (e.g., /etc/passwd).

```
void mktemp (char *tmpfile)
{
  if (access (tmpfile, W_OK) == 0) {
    /* Attack window */
    if ((if = open (tmpfile, O_WRONLY)) == NULL) {
      perror (filename);
      return (0);
    }
    /* write data to the file */
    ...
}
```

Bishop describes this as a Time-To-Check-To-Time-Of-Use flaw (TOCTTOU). There are tools to statically check for such flaws. However, from the study of computability theory we know that no solution exists for the general case of deciding if a program has an exploitable TOCTTOU flaw. Bishop explains the reasoning as follows. Given an arbitrary program, consider the existence of an exploitable TOCTTOU flaw. This property holds for at least one program (there are flaws) and it does not hold for at least one program (there are simple, secure programs). Hence, this is a non-trivial property of the language of programs (approximately Turing machines). That is, some programs have the flaw while other programs do not. Then by Rice's theorem, the set of programs with an exploitable TOCTTOCU flaw is undecidable[28]. That is, we can't even decide an answer for the general case given an infinite about of computation.

7 Root-level Local Attacks

7.1 Rootkits

Once an attacker has obtained root on a system, he can do pretty much anything he wants. However, many crackers don't have a good idea of how to hide their tracks once they're in, or how to open other security holes to make returning an easier process. Instead, they use packages called 'rootkits', which the cracker can simply transfer to the compromised machine and install. A typical rootkit will install new binaries on the target machine which make it more difficult to detect unusual behavior (e.g., extra processes running, or an extra user logged in), as well as changing the timestamps of files which might indicate a breakin, or even removing relevant lines from the system logs. More advanced rootkits are available to do a wide variety of cover-ups.

7.2 Kernel Modules

Perhaps the most insidious attack against a modern Unix machine is to install a new kernel module once you've compromised root. Kernel modules live in kernel-space rather than normal user-space, and they can

[27] http://www.l0pht.com/advisories/init_advisory.txt

[28] Bishop, Matt, and Dilger, Michael. "Checking for Race Conditions in File Accesses." USENIX Computing Systems, vol. 9, no. 2, pp. 131-152, Spring 1996.

intercept system calls and modify (or completely replace) their behavior. In this way, a cracker can truly take control of a machine and prevent himself from being noticed.

Of course, kernel modules exploits don't work on systems that don't support kernel modules. It's also much more difficult to build an attack that will work successfully on a machine where you don't have the source code for the kernel.

There are a number of preventative measures that can be taken to reduce the threats from kernel module exploits. One possibility is to introduce the concept of layered defense in the operating system, where the kernel takes away privileges even from itself after it's done certain boot procedures. Another alternative is to only allow new kernel modules to be loaded at boot, after which you load a module (or otherwise execute some code in kernel-space) which prevents the loading of any future modules. Of course, a clever cracker will configure his module so it will be installed upon boot, before the 'poison' module (the final module) is loaded.

8 Acknowledgments

We thank Bob Mahoney for managing and financing us, Geoff Goodell for investigating X cookies, and MIT's Network Security team for helping us in many ways. Ryan Thurston of Thurston Productions graciously volunteered to video tape this event. We also thank Marc Horowitz for providing the Internet Worm video footage.

9 About the authors

We are members of the MIT Network Security Team (net-security@mit.edu). Roger Dingledine '99 (6-3, 18, M.Eng 6) and Kevin Fu '98 (6-3A, M.Eng 6) are members of the MIT Network Security Team. Both were M.Eng students of Prof. Ron Rivest in the Lab for Computer Science (LCS).

Roger (arma@mit.edu) conducts research in anonymous distributed publishing systems. He is the project leader for both the Simple End-User Linux project (`seul.org`) and the Free Haven project (`freehaven.net`), and is an author of the upcoming O'Reilly book "Peer-to-peer." Currently he works as the Security Philosopher for Reputation Technologies, Inc. (`reputation.com`).

Kevin (fubob@mit.edu) is a doctoral student in the Parallel and Distributed Operating Systems group at LCS. His advisor is Prof. Frans Kaashoek. Kevin also coordinates the LCS Applied Security Reading Group (`http://pdos.lcs.mit.edu/asrg/`). His research interests include secure file systems (`www.fs.net`) and client authentication on the Web (`cookies.fooworld.org`).

We would love to hear your feedback about our documentation and presentation! Let us know what you think of the content, presentation, and scheduling.