Lectures on Distributed Systems

System Protection and Firewalls

Paul Krzyzanowski

INTRODUCTION

Computer security was an issue since the earliest days of digital computers. The earliest computers, such as the eniac[1] and the Colossus[2] were designed for military use, the former to compute ballistic firing tables and the latter to decipher German Lorenz[3] codes. These were primitive machines: running no operating system, running a single "process," and operated by a single user. Nevertheless, security of their data (or even of their existence, in the case of Colossus) was of paramount importance. This was achieved via *physical security* – guarded buildings, armed guards, face-to-face authentication.

Things have changed since then. We have sensitive data coexisting with data of others on the same file servers. Multiple processes run on the same machine. As we started to work a distance and use data communication networks, those networks became avenues for snooping: authentication sessions and data transfers could be grabbed. To make matters even more complex, we actually want to run other people's code on our machines. It's unlikely that we will write all the code that runs on our machines, so we rely on externally-produced operating systems, compilers, and applications. We are somewhat at ease when these come from trustworthy sources (trusted vendors, such as Adobe, Macromedia, Apple, et alia) but we also want to run applications from random parties – applets that are loaded with our web pages or miscellaneous games that we may spontaneously download. This can open up a Pandora's box of problems: how do we know that the applications are not performing malicious acts behind their innocent façade?

[1] Completed in 1945 at the Moore School of Electrical Engineering at the University of Pennsylvania.
[2] Completed in December 1943 at Bletchley Park in England.
[3] The Lorenz cipher was used by the German high command to send classified communications. The Lorenz cipher was based on the principles of a one-time pad, effectively xor-ing the plaintext with a stream of random text. Since one-time pads are not practical due to the difficulties of propagating keys, the Lorenz cipher employed a pseudo-random number generator. This, of course, turned out to be the Achilles heel of the system.

ATTACKS

With the advent of data networks, systems have become easier to attack. The ability to attack from the comfort of one's own home provides not only comfort, but a sense of invulnerability (the security guard won't catch you in the wrong place) and extra time (you don't have finish up before the security guard comes around). Scripting provides the ability to automate attacks that require brute-force trial and error. Social networking allows hackers to share information and techniques. One does not need to be a skilled programmer to write a virus: it is often as simple as downloading a virus kit. Software can also be obtained for snooping on the network, cracking passwords, and exploiting known security holes.

SYSTEM PENETRATION

There are numerous techniques for penetrating computer systems. This list only touches upon them.

Guess a password

Some systems and software comes with default passwords. Quite often, these are not changed. The installer's concern of the moment was on getting the software installed rather than thinking of the security implications of not changing the password. Knowledge of these defaults makes for trivial penetration.

A screen will appear and ask you for a User Name and Password. Enter admin (the default) in the *User Name* and *Password* fields. Then click the OK button.

Figure 1. Step B of the router configuration instructions from the Linksys RV082 VPN router instruction manual.

Left to their own devices, most people are not creative when it comes to selecting passwords. This creativity is further stifled by the vast preponderance of passwords that most people are forced to remember. Given the lack of creativity, guessing a password is often not as onerous a feat as one might imagine. One form of guessing is known as a *dictionary attack*. This involves trying words in a dictionary (Webster[4]'s, lists of names, or a list of common known passwords, for example), prefixing and suffixing the words with numbers or symbols, and trying "tricks" such as substituting '0'(zero) for 'o' or '1' (one) for 'l' (ell). As a last resort, one can try an exhaustive search over all combinations of letters, numbers, and symbols.

Social engineering

People tend to trust each other and avoid confrontational situations. This not just a matter of naïvete but a necessary underpinning of social interaction. Unfortunately,

[4] May Unix-derived systems come with a list of words from Webster's 7th Collegiate dictionary and the Brown Corpus in the file /usr/dict/words.

this trust extends to a lack of proper authentication. By studying an organization, learning some of the acronyms and buzzwords that are representative of the group one wishes to penetrate, one can pass for an insider. Programs such as *finger* as well as personal web pages allow one to learn about people in an organization: likely interactions, chain of command, responsibilities, and working hours. This can further aid in social infiltration. Would you trust the night shift administrator whom you've never met? How about a purported senior vice president who forgot her password and is threatening your immediate dismissal if you do not rectify the situation immediately?

A recent abuse of trust comes in the form of *phishing*: sending email that purports to come from a trusted party, such as a bank, asking the recipient to validate their account information by directing them to an authentic looking web page that is hosted by an imposter.

Trojan horse

Odysseus devised a plan. After ten years into the Trojan war, Greek ships were sent away and the Trojans believed that the Greeks have surrendered, leaving behind a gift: a giant wooden horse designed by the artist Epeius. The Trojans dragged the horse into their walled city. At night, the Greeks hiding in the horse slipped out, killed the Trojans and set fire to their city.

In computer parlance, a *Trojan horse* refers to a program that masquerades as another, trusted, program. The classic Trojan horse was a program that would display a login prompt, leading the user to believe that a computer (or terminal) was unused and awaiting a user login. The unwary user would enter his credentials (login name and password). The program would then stash them in a file or send them out via a network connection and then exit, printing a *login incorrect* message and executing the real login program. The user would simply assume that a mistake has been made, enter the credentials again (this time to the real program), and continue with life, unaware that his login name and password were compromised. This attack led Microsoft Windows to encourage the use of pressing *control-alt-delete* prior to logging in, since that key combination on PCs cannot be trapped by applications and forces a response by the kernel.

More recently, Trojan horses have appeared in the form of exploitations of Microsoft Internet Explorer to redirect traffic to different IP addresses. A novel form of a Trojan horse was exposed in November of 2004, also targeting Microsoft Internet Explorer and Windows XP SP2. In this attack, the dialogue window that warns users that they may be downloading malicious content could be disguised so that the unwitting user will click on an innocuous-looking message only to install harmful software on the computer.

Exploit software bugs

Virtually every non-trivial (and much trivial) program has bugs. Big programs tend to have more bugs than small programs. Poorly or hastily written programs tend to have many more bugs than well-written and carefully designed programs. Programs written by huge teams tend to be buggier than those written by a handful of good programmers. This empirical evidence does not bode well for much of commercial software, which is often created by large teams under tremendous time constraints due to market pressure.

Given that most software out there has bugs, some of those bugs may lead to security exploits. In the worst case, these exploits will allow a user to run any command, such as a shell. On Microsoft Windows systems, most users operate with administrative privileges and, as such, most programs run with administrative privileges. On Unix-derived systems, some programs are *setuid* programs. This means that, upon running, they assume the identity of the owner of the program, not that of the user running the program as is usually the case. If a user can exploit these programs then an ordinary user can attain administrative privileges[5]. Most server software runs with administrative privileges, making it ripe for exploit: the software is accessible via the network and offers powerful privileges if penetrated: an enticing combination.

One source of bugs is not really bugs at all. Programmers sometimes augment their software with undocumented options and back-doors to gain access to the system. Discovering these can make gaining entry into the system a snap. For example, poor checking on the syntax and disposition of a URL led to web server exploits where a remote user can execute an arbitrary program on the web server.

Buffer overflow

Of the real bugs in a system, none has attracted more attention than the *buffer overflow* bug. A buffer overflow occurs when a program is reading data into a buffer (e.g., input from a user) but the amount of data that is read is greater than the amount of memory allocated for the buffer. A well-written program would ensure that no additional data will be written to the buffer beyond its size. However, that means having another variable to measure the remaining size of the buffer and who wants to bother coding that when you *know* that the input data will never exceed your already generous buffer size. It did not help matters any that one of the more commonly-used functions in the C standard i/o library is *gets*, whose purpose is to fill a given buffer with a line read from the standard input. Unfortunately, the programmer can only supply the starting address of the buffer, not its size[6].

[5] Some setuid programs that were exploited in the past include *uucp*, *lpr*, *sendmail*, *mount*, *mkdir*, and *eject*.

[6] The similar function, *fgets*, which allows a programmer to specify a FILE pointer instead of using stdin, does require the programmer to specify the buffer size. This function should be used in place of *gets*.

When a buffer is declared locally within a function (in a language that supports reentrant procedures, such as C or C++), it is allocated on the stack. Writing beyond the limit of this buffer may overwrite not just other local variables but also the memory location on the stack where the return address is stored. When the function returns, the modified return address will transfer control to the location of the intruder's choosing, such as code that was placed in the buffer.

Some software attempts to safeguard against this by checking that the return address has not been modified during the execution of the function. A buffer overflow attack that loads arbitrary executable code is difficult to carry out on some processors whose memory management unit (MMU) allows execution privileges to be turned on or off per page. Processors such as the Sun SPARC, HP Alpha, IBM PowerPC, AMD Opeteron and Athlon 64,Transmeta Efficeon, and Intel's IA-64 architecture support this feature (which is often known as the NX bit). Unfortunately, today's dominant Intel Pentium-family processors do not support this.

* * *

The network itself has turned out to be a rich source of attacks and an enabler for penetrating systems.

Fake ICMP or RIP packets

Some routers perform dynamic routers and rely on messages from other routers in the form of ICMP (Internet Control Message Protocol) and RIP (Router Information Protocl). Sending these messages may allow a router to believe that the best route for packets is via an intruders (or compromised) system or router.

Address spoofing

Some applications base their authentication on the source IP address of a message (for example, *rsh*, the remote shell program). An intruder can change a machine's address to that of another system or spoof an IP packet stream to appear to come from another address. This is best done in conjunction with modifying routes or taking a machine that rightfully owns the masqueraded address out of service.

ARP cache poisoning

The key to communicating on an IP network is to find the hardware address (MAC) that corresponds to a given IP address. Systems do this via the Address Resolution Protocol, or ARP. A machine broadcasts an ARP query and trustingly accepts any response. If the response comes from an imposter, it will address its IP packets to that imposter. Alternatively, an attacker can broadcast ARP responses to non-existent MAC addresses, effectively cutting off access to other systems (and routers!).

Ping of death

The ping of death is a denial of service attack that is meant to take a machine out of service by sending it an IP datagram that is larger than the allowable limit of 65,535 bytes. Packets can be fragmented and hackers realized that many operating systems did not know what do to when the fragments added up to a packet that was larger than the 64K limit and simply froze up. Most operating systems were patched for this deficiency by 1997.

SYN flooding

Another technique used for making a machine appear dead to the network is known as *SYN flooding*. The TCP/IP protocol uses a three-way handshake to set up a connection. In the first step, the originator sends a SYN packet. In the second step, the receiver sends an acknowledgement (a syn/ack packet). In the final step, the initiator acknowledges the receipt of this acknowledgement and the session is established.

Because TCP is a stateful protocol that supports retransmission and in-order delivery of packets, the recipient needs to allocate a certain amount of memory for each connection. Upon receiving the SYN, the receiver allocates memory for the new connection. To avoid using up all resources, there is a limit to the number of outstanding requests. Additional SYN messages will be directed to a backlog queue. Since the backlog queue is also finite, any further SYN packets get dropped.

The SYN flooding attack works by sending SYN packets that are masqueraded to appear to come from an unreachable host. The receiver sends a SYN/ACK to this unreachable host and eventually times out waiting for the final response. This time-out period before the pending session is cleaned up is often around 75 seconds.

TCP session hijacking

A *sequence number attack* is one where the intruder masquerades as one of the endpoints of a TCP connection. Traffic to the target session is monitored (or imposter packets are sent to generate response traffic) to find the sequence numbers for the TCP packets of that session. The original machine is then taken out of service (e.g., via SYN flooding) and the intruder steps in, masquerading as the original machine and continuing the session with the correct TCP sequence numbers.

UDP spoofing

Spoofing udp packets is trivial compared to spoofing tcp packets. There is no connection setup and no sequence numbers to deal with. Packets can be grabbed of the network and replayed at a later time or spurious packets can be inserted into an existing communication session at will.

* * *

Many network services themselves have holes (bugs or lapses in authentication). A sender's address can be masqueraded with SMTP (Simple Mail Transfer Protocol). While not an intrusion, this is useful for social engineering. The *sendmail* program, still one of the most popular email servers, has been debugged for decades. The *finger* dæmon is useful for social engineering – finding out about people by running the *finger* command. It also used to be vulnerable to a buffer overflow attack because of its use of *gets*.

Remote Procedure Calls

Remote procedure calls are another source of problems. Unauthenticated, they can allow any client to execute a remote procedure on the machine. The easiest, and most common, way to use RPC is without authentication. In fact, authentication was not even present in its earliest instantiation from Sun. If an intruder has control of a system offering RPC services, it may be able to fake the port mapper or replace the real service with that of an imposter. Finally, the RPC subsystem itself may be vulnerable. A number of security holes were uncovered in 2003 in Microsoft's Distributed Component Object Model (DCOM) where hackers could send an improperly formatted RPC message to cause a buffer overflow and allow them to run their own code on that system.

Other network services

In addition to exploiting RPC holes on the server, NFS's stateless design lets you use a file handle in the future, even if the file system is no longer exported. Moreover, even if authentication is enabled, NFS data is not encrypted. Someone capable of snooping on the network may see unencrypted file chunks of files.

Services such as *rlogin* (remote login), *rsh* (remote shell), and *ftp* (file transfer protocol) send their authentication credentials (login and password) in plain text. Anyone who can snoop on the network can grab the user's login and password. Moreover, as a convenience *rsh* and *rlogin* allow the use of .rhosts and /etc/hosts.equiv to list trusted hosts and short-circuit the annoying process of asking for a login and password. If intruders can either masquerade as a trusted machine or modify one of these files then they can gain access to the system.

A client-server based windowing system such as X-windows is potentially vulnerable to TCP sequence number attacks. For example, an intruder can tap into an open terminal session and take it over.

Internet Explorer

At the time of this writing, Microsoft's Internet Explorer (IE) web browser is the most-used browser, enjoying over 90% of the browser market. As such, it enjoys the status of probably being the world's most actively hacked program.

The benefits of finding holes in are realized if you can get someone to navigate to your website and download something that will open up their computer to either allowing you to run an arbitrary command, redirect their URLs or modify their registry. In recent years, bugs capable of system penetration were found in malformed URLs, buffer overflows, ActiveX flaws, bugs with PNG and JPEG graphics, Jscript, and the processing of XML object data tags.

Many more
The number of services exploited is too voluminous (and pointless) to mention. Other services include remote administration servers (e.g., Microsoft BackOffice), Java applets, Visual Basic scripts, shell scripts, and many, many others.

WORMS AND VIRUSES

One form of intrusion involves not targeting a specific computer but rather spreading an attack through a vast number of systems. Programs that behave in this way fall under the category of worms and viruses.

A *worm* is a process that spawns copies of itself. It is a self-contained program that attempts to connect to other systems (exploiting some security hole) and replicate itself onto that system, *ad infinitum*. In the best case, it is a harmless process that does nothing but propagate. More commonly, it will use up system resources, possibly spawning multiple copies on the same machine. In the worst case, it can run malicious software that will destroy files, send spam, or launch a distributed denial of service attack[7].

The worm that brought worms to the public eye was Robert Tappan Morris, Jr.'s Internet worm of 1988. It started off by exploiting the *gets* buffer overflow bug in the *finger* daemon to load a small bootstrap program onto the target machine (this program was only 99 lines of C code). The bootstrap program then connected back to the originator and downloaded the full worm. This worm then searched for other machines to which it could propagate through one of four methods:

- It searched user directories for .rhost files that designated other trustworthy machines that would not need a password for logging in via *rsh* or *rlogin*.
- It attempted to exploit the buffer overflow bug in the *finger* daemon.
- Some versions of *sendmail* had a DEBUG mode that allowed a remote user to get a shell with administrative privileges. It tried to connect to the mail server on other machines on the network and exploit this bug.

[7] A *distributed denial of service* (DDOS) attack is an attack where a number of computers gang up on one system, saturating it with network activity to render it dead (unresponsive) to the rest of the world.

- It tried guessing users' passwords via a dictionary attack, using 432 common passwords and combinations of the account name and the user's full name.

The worm was written only for DEC VAX systems running the 4.x BSD system and Sun 3 systems running SunOS. It was not intended to cause damage but a bug in the code caused it to spread repeatedly within a machine, filling up its process table.

* * *

A *virus* does not run as a self-contained process. Instead, it is code that is augmented onto some other body of code. The classic form of a virus has been a *file infector*, where the virus attaches copies of itself to one or more programs on the computer. When those programs are run, they execute the virus code and may then choose to resume executing the original program (for example, the virus might check the date and, if it is not April 1st, just continue with the progam). File infectors have been problems primarily on systems without adequate protection mechanisms. If a user does not have permission to modify installed programs and system files, the virus, running under the user's credentials, will not be able to affix itself to any of these programs. Unfortunately, the world's dominant operating systems (Windows 3.1, Windows 95, 98, …) were such systems. Another place where viruses might install themselves would be in the boot sector of a disk. This gives them a chance to run at boot-time, before the operating system is run.

These days, the most common viruses are Visual Basic scripts sent via email attachments. These attachments are often disguised to appear to be photos or other innocuous material.

PENETRATION FROM WITHIN

The classic threat model assumes that penetration attempts come from adversaries from the outside. The other threat that an organization has to deal with is damage control from the effects of a successful intrusion. Once malicious software is loaded onto a computer, that computer becomes vulnerable to whatever actions that software undertakes. It has full access to the internal network, other computers, and data on the system.

Adware

Adware is a program that displays ads to the user, either through pop-up windows, on the application, or via browser redirection. In its most innocent instantiation, it is an intrinsic part of a program that the user installed and exists for defraying programming costs (you pay less or nothing for the software but are forced to see ads – migrating the TV and radio model to the computer). In other cases, it is a separate

program that is installed without the user's knowledge. In some cases, it may pass on information about the user or monitor the user to make a decision on which ads to serve.

Dialers

A *dialer* is a program that uses the computer's modem to create a connection to the Internet. It is an essential piece of software for those still using modems. Malicious software can change the dialing preferences[8] to use premium-rate (ripoff-rate) long distance telephony services. This software is often installed via a web-page download after ActiveX controls are enabled via a Visual Basic script in an email message. In some cases, a user may inadvertently install a dialer, falling prey to advertising that claims an improved network connection or extra services. Needless to say, all these attacks target Microsoft Windows systems.

Spyware

Spyware encompasses the class of programs that send any information about the user over the network without the user's knowledge. In some cases, this is coupled with adware; either for sending user information over the network or for targeting specific ads for the user. One particularly insidious form of spyware is the *key logger*. A key logger is a program that records every keystroke (and mouse movement as well). Microsoft Windows provides a generic mechanism, called a *book*, that programs can use to intercept system calls. These hooks can be chained, so that a request to execute a system call may pass through several hooks. A key logger installs hooks via the *SetWindowsHookEx* Win32 API function to capture key up, key down (WM_KEYBOARD) and mouse events (WH_MOUSE).

PROTECTION BY THE OPERATING SYSTEM

The principal purpose of an operating system is to provide controlled and authorized access to resources. For example, the process scheduler provides access to the CPU. The memory management unit (hardware that is configured by the operating system) provides controlled access to memory, allowing each process to have its own address space and disallowing one to access another's memory. Device drivers provide controlled, exclusive access to system devices. Certain devices are never accessible to normal processes. File systems provide an abstraction for accessing logical regions of persistent data, ensuring that access permissions are obeyed. Sockets provide a controlled mechanism to send and receive messages over a network.

[8] Again, an example of insufficient protection on the part of the operating system.

ACHIEVING SECURITY: THE FOUR A'S

Security has several aspects to it. We can think of it as the four A's: Authentication, Authorization, Accounting, and Auditing.

AUTHENTICATION

Authentication refers to the process of identification: validating that the user is not an imposter. The traditional method used by operating systems to authenticate a user via a login process, where the user supplies credentials consisting of a login name and a password. This is a bad idea if done over a network without having established a covert communication channel first since the entire session can be snooped upon. A covert channel can be established by generating a common key with a Diffie-Hellman key exchange or using public keys to exchange a session key. In either case, unless the public key comes from a trusted source, no authentication has been performed. However, the resultant session key will not be known to other parties and can be used to set up an encrypted session (using a symmetric encryption algorithm) that can carry out the login-password authentication.

Other authentication techniques that may be used include:

- one-time passwords (S-key, RSA, SecureID).
- Challenge-response.
- Shared secret keys and nonce-based authentication, although this requires that the keys have been distributed via a secure channel (e.g., SKID).
- Public key authentication. This requires that the public key come from a trusted source or be a digital certificate that is signed by a verifiable, trusted party.

AUTHORIZATION

Authorization is responsible for access control once a user has been identified. A decision has to be made whether a request (to get a shell, access a file, access a server, et al.) is accepted or denied. Operating systems generally rely on *access control lists* (ACL) for regulating access to objects within the system. An access control list is associated with each object (e.g., a file) and enumerates a list of users and groups and the access permissions that each one has. The user and group IDs are associated with the user's processes upon login.

In a networked environment, a network service is often a process that is running under its own user ID (or, worse yet, with administrative privileges). It is responsible for the veracity and proper authentication of network messages that it processes. In some cases, neither the process nor the system may even know of the user. This is fine for anonymous services (e.g., accessing a web page) but problematic if authentication is required. In this case, the server may rely on a trusted third party that will

grant credentials to the session. Kerberos is an example of such a system that is based on shared secret keys and an untrusted network. When the user's process presents the server with a Kerberos ticket, Kerberos can validate that the user has been granted access to the service simply by decrypting the message to extract a session key and validating that the user indeed possesses that key.

In an environment where the network (at least, the internal network) is trusted, a system such as RADIUS[9] can be used. RADIUS is a trusted server that can store arbitrary authentication and authorization credentials. A service passes the collected user information to the radius server, which validates the authentication credentials and returns authentication information to the requesting service. It may also log the transaction for accounting purposes.

ACCOUNTING

Should security be breached, the first questions that pop up are: *what happened? Who did it? How did it happen?*

Do answer these questions requires the ability to examine events of the past. To do this, events of potential interest must be logged. What to log is very much dependent on the level of paranoia in an organization, performance issues, and the amount of logs one is willing to have. Some useful things to log are user logins, user logouts, network requests for service, the creation of new accounts or deletion of old ones, automatic execution of jobs (e.g., cron jobs), and any system configuration changes. More detailed items to log are database transactions and a command trace per user.

Since a skilled intruder is likely to clear any logs, it helps to send the logs to a remote logging server (hopefully one that is well secured from attacks).

Intrusion detection software has been written to scan through various system logs., actively monitor system activity and analyze network traffic. The administrator is alerted when "unusual" activity is detected. The challenge, of course, is defining what is meant by "unusual activity." Thus far, these systems have not been very successful.

AUDITING

Auditing is the process of validating the system configuration and going through the software source code and searching for security holes. It requires access to the source code for all applications (and operating system) that need to be audited as well as highly experienced staff. More importantly, it requires time. Auditing does not ensure or prove security but is form of testing the software and eliding constructs that are known to have security holes. Few systems have been thoroughly audited. The OpenBSD operating system installation is one of them.

[9] Remote Authentication Dial-In User Service. It is described in RFC 2138. Microsoft's implementation of this is called the Internet Authentication Service (IAS).

Complex systems are, of course, much more difficult to audit. For example, Windows 2000 is estimated to contain between 35 and 60 million lines of code and the prospect of a thorough audit is dim.

DEFENDING THE SYSTEM

Defending the system from malicious software that may already have entered the computer involves:

- Restricting access privileges. An ordinary user should not run with administrative privileges. This will decrease the operations that malicious software can perform and the files that a virus can infect.
- Virus checking. There is no magic in detecting whether a program has been modified with a virus. All that can be done is to scan the program for bit patterns that correspond to instructions present in any of the tens of thousands of viruses that have been identified thus far. New viruses will come out next year and the virus scanning software will need to be augmented with those patterns.
- Personal firewall. Since malicious software is likely to send data over the network, it becomes necessary to monitor and restrict the individual applications that can access the network. A personal firewall, such as Zone Alarm from Zone Labs, installs hooks in the operating system to intercept operations that access the network and allow users to allow or deny them explicitly. The problem with this, of course, is that users need to be somewhat educated in understanding which applications can and should access the network legitimately and which ones should not.

CODE SIGNING

To guard against installing code from untrusted parties, one can require that the software be digitally signed by a trusted provider and validated prior to installation. This is what Microsoft does with their Authenticode code signing system. A software publisher registers themselves with the VeriSign certification authority, creates an RSA public key pair, and obtains a class 3 software publisher's certificate. Prior to being made available to the public, a signature is generated by hashing the software and encrypting the hash with the publisher's private key, This signature and the associated digital certificate are added to the code as part of the signature block. Prior to installing the software, the recipient validates the certificate and validates that the code hashes to the same value as the decrypted signature.

DEFENDING THE NETWORK

While we'd like to communicate with all machines: those on our network and other systems, we do not want systems trying to penetrate our network. Certain network services, such as database access or perhaps even a network login, may need to be present on some of our machines but should not be accessible by others outside of our local network. We basically have two domains of machines: a trusted domain (our machines on our network or networks) and an untrusted domain – all the external machines.

The next step is to *isolate* this trusted group of machines from the rest of the untrusted world. In some cases, this may be easy: move all the machines onto a private network and do not allow untrusted people onto the machines or the network. Unfortunately we are often not content to remain disconnected from the rest of the world (there's a lot of cool stuff out there). A solution is to protect the junction between a trusted internal network of computers from the external network with a *firewall*.

The two major approaches to building a firewall are *packet filtering* and *application proxies*.

PACKET FILTERING

Packet filtering is the selective routing of packets between internal and external hosts. This firewall serves as a form of a gateway that forwards packets from one network to another, allowing only certain packets to flow into the internal network. Packets can be accepted or blocked from transmission in a way that reflects the security policy of a site. A router that can perform packet filtering is known as a *screening router* (Figure 2).

Ordinary routers simply look at the destination address of each packet and pick the best interface on which to send the packet to its destination. A screening router, on the other hand, determines the route (destination) and also decides *whether* the packet should be routed or dropped. Most routers and firewall products support packet filtering.

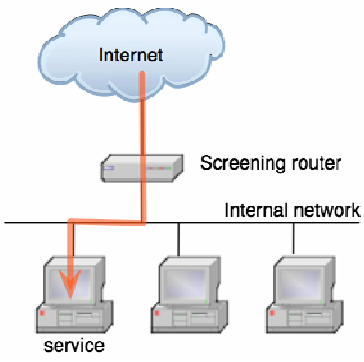

Figure 2. Screening router

Various fields of each packet header are examined:

- IP source address
- IP destination address
- Protocol (TCP, UDP, ICMP)
- TCP/UDP source port
- TCP/UDP destination port
- ICMP message type

Any or all of these may be used to accept or deny a packet. In addition, since the router knows the interface on which the packet arrives and the interface on which the packet goes out. Packet filtering allows the decision of allowing or blocking a packet to be based on any or all of these eight components. To set up packet filtering rules effectively, one needs to have an intimate knowledge of TCP or UDP port utilization.

Filtering rules are generally processed a line at a time, which each line representing a set of IP header parameters that need to be matched. A match on line indicates whether the packet should be allowed to be routed, dropped, or whether additional rules should be processed.

For example, to block all incoming connections from systems outside the internal network except those for the SMTP (mail) server, we may choose to do the following:

```
allow dest_port=25
deny  interface=external
```

The first rule matches any packets whose destination port is 25 (the port on which the SMTP server listens) and allows them to be routed. The second rule matches all other packets that arrive from the external interface and drops them.

We may also choose to allow connections from specific machines or networks. For instance, let us suppose we want to do the following:

> Allow any machine to connect to the web server (TCP port 80 for HTTP; port 443 for HTTPS) that we have running on the system with address 12.34.5.6 and allow any machine to connect to the mail server (port 25) that is running on the machine with address 12.34.5.7. We also want only machines in the subnet 19.8.*.* to be able to connect to the telnet port (23) on any machine within 12.34.5.*. All other packets should be dropped.

We can set up the following rules:

#	Type	source address	source port	destination address	destination port	Interface	action
0	*	123.34.5.0/24	*	*	*	1	deny
1	tcp	*	>1023	12.34.5.6	80	1	allow
2	tcp	*	>1023	12.34.5.6	443	1	allow
3	tcp	*	>1023	12.34.5.7	25	1	allow
4	tcp	19.8.0.0/16	>1023	12.34.5.0/24	23	1	allow
5	*	*	*	*	*	0	allow
6	*	*	*	*	*		deny

The above table identifies two interfaces: "1" refers to the external interface (connection to the Internet) and "0" refers to the internal interface (connection to our local area network). Note the addition of rules 0 and 5. Rule 0 instructs the packet filter to

drop any packets coming from the Internet whose source address is that of the local network. This prevents IP masquerading, where an external packet is disguised to look like an internal one. Rule 5 allows any packet from the internal network to be routed to the external network.

We may set filters to block any connections from certain systems or block access to all "dangerous services" such as tftp, X windows, rpc, r-commands (*rlogin*, *rsh*, *rcp*)[10].

One problem with the blind filtering provided by screening routers is that it is done with no context – each packet is examined on its own with no regard to previous packets. Without maintaining state, a router cannot tell, for example, whether a return packet is in response to an established connection. For instance, if you establish a connection to a mail server, it should be able to talk back to you over the address and port that matches the original port. A simple packet filter would simply have to keep all ports open in the outbound direction. In other cases, we might want to block TCP data packets until a valid TCP connection has been established. Again, the packet filter would need to maintain state to know that a connection has been established between two endpoints. Finally, as a more complex example, FTP has a client connect to the server (on port 21) but any data transfers require that the server establish another connection back to the client. If we had a "smart" packet filter, it could expect and allow the secondary connection only if the first one has been accepted. Packet filters that can keep track of connection state and make decisions based upon it are called *stateful packet filters*[11].

A screening router will generally sit between the internal and external network. It has an enormous responsibility. It needs to do routing as well as make filtering decisions. It is the only protection for the internal network: if its security fails or is compromised, the internal network is exposed. For this reason, the software that comprises the firewall should be simple (less code yields less bugs) and offer as few services as possible. It will not run network services, compilers, or have a lot of accounts. It will know the addresses of its routers on both the inside and outside and not issue queries for them. It will not allow packets to enter that appear as if they are from the internal network (masquerading; see rule 0 in the table on the previous page). For detailed information on firewalls and advice on installing them, take a look at Bill Cheswick and Steven Bellovin's book *Firewalls and Internet Security*.

PROXY SERVICES

A screening router (packet filter) can allow or deny access to a service, but it cannot protect operations within a service. These are known as *protocol attacks*. An example is

[10] These are considered *dangerous* because they perform username, password authentication in cleartext over an unencrypted channel.
[11] The service provided by such a filter is often called SPI: Stateful Packed Inspection.

trying to enter debug mode in *sendmail* or sending large amounts of bogus data trying to create a buffer overflow.

Proxy services are a set of specialized applications or server programs that run on a firewall host. This host is typically a *dual-homed* host. This means that the host has two network adapters: one interface on the internal network and another on the external network. This allows the host to know explicitly which packets are coming from the outside and not worry about forged source addresses. The dual-homed host on which the proxy service runs on is also a *bastion host*. This is a term for a machine where extra effort has been taken to make it highly secure because it is exposed to the Internet and is vulnerable to attack. Securing every machine on the system is difficult but certain machines, such as proxies, deserve the extra effort. A bastion host will not have many user accounts and will have only the minimal system installation needed to run services (no compilers). The less tools that intruders have to work with, the less damage they can do.

Proxy programs take requests for Internet services (e.g. FTP) and forward them to the actual services. Proxies provide replacement connections and act as gateways for these services. Another term for a proxy is an *appliaction-level gateway*.

Proxies sit between a user on the inside (internal network) and a service on the outside (Internet); see Figure 3. Ideally, proxies should be transparent and present the illusion that the user is dealing with a real server. Proxies are effective only in environments where direct communication is restricted between internal and external hosts. Dual-homed machines or packet filters can accomplish this. A proxy makes sure that the final server never gets packets that came from the Internet – those packets are picked up by the proxy; the final server only talks with the proxy.

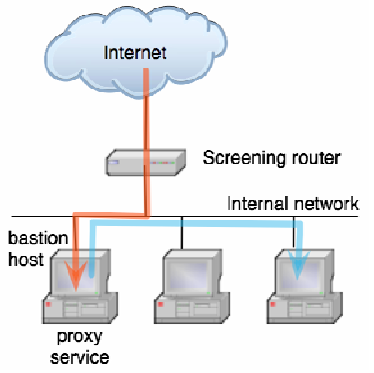

Figure 3. Dual-homed host proxy

Proxies can provide *stateful inspection* of data. They can examine a session and have the ability to *look at the data* (content) on the session. In addition, a proxy can provide authentication facilities for applications that have weak or no authentication.

As an example of content-specific filtering, consider SMTP mail filtering on Checkpoint Software Technologies' Firewall-1™ product. It provides the following capabilities:

- mail address translation. The original *From* address of outgoing mail is rewritten to a generic address (to conceal internal network structure).
- redirect *To* addresses.
- drop mail originating from a given set of addresses.
- strip MIME attachments of specific types.

- strip the Received information from outgoing mail (also to hide the internal structure).
- drop mail messages above a given size.
- perform anti-virus checks on incoming messages/attachments
- do not allow the application to connect to the ever-buggy *sendmail* inside the network. Instead, enqueue the messages locally into a cache on the firewall, and dequeue them onto *sendmail*.

This particular product, representative of high-grade commercial firewalls, also has knowledge of other applications, such as *telnet*, *ftp*, *rlogin*, *rsh*, and *http*. It also allows support for a number of authentication schemes.

FIREWALL ARCHITECTURES

DUAL-HOMED HOST ARCHITECTURE

A *dual-homed host architecture* is built around a dual-homed host computer with at least two network interfaces. The ability to route between the two networks is disabled so that IP packets from one network (e.g., Internet) are not routed to the other (e.g., internal). See Figure 3.

Services are provided by proxies or by having users log into the dual-homed host directly. Two problems with this architecture are that user accounts on the dual-homed host present significant security problems and that proxies may not be available for all services.

SCREENED HOST ARCHITECTURE

A *screened host architecture* (Figure 4) provides services from a host attached to the internal network. Primary security is provided by packet filtering. Only certain connections are permitted (e.g., deliver email). Connections may also be disallowed from the outside to any internal hosts except for a bastion host running proxy services. The bastion host will be connected on the same internal network as other internal machines.

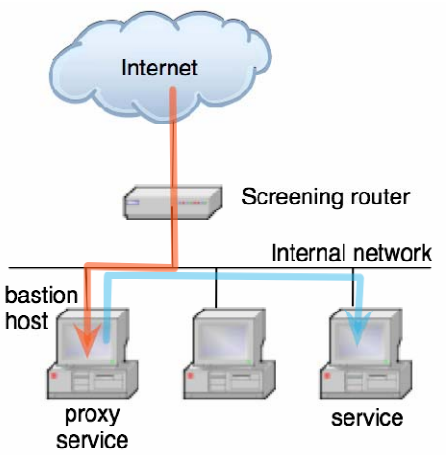

Figure 4. Screened host architecture

SCREENED SUBNET ARCHITECTURE

We can add an extra layer of security to the screened host architecture by adding a perimeter

network that further isolates the internal network. This perimeter network is known as a *DMZ*, or Demilitarized Zone (Figure 5).

The reason for doing this is that bastion hosts are the most vulnerable machines on your network. They are most likely to be attacked because they *can* be attacked (they're the only ones accepting any packets from the outside network). With a screened host architecture, there is no defense between the bastion host and other internal machines: should the bastion host be penetrated, the entire internal network is instantly vulnerable. By isolating the bastion host(s) on a perimeter network, you can reduce the impact of a break-in.

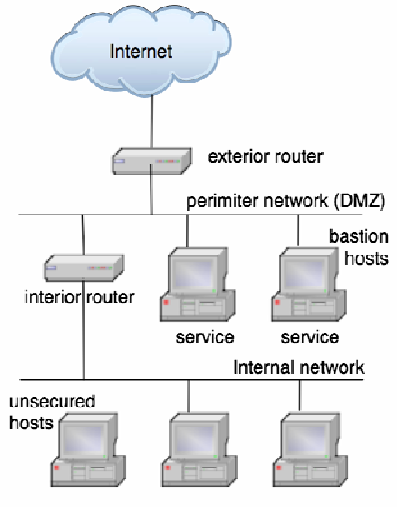

Figure 5. Screened subnet

The design of a screened subnet architecture consists of two screening routers:

- one sits between the perimeter network and the internal network
- the other sits between the perimeter network and the Internet

An attacker would have to get through both routers to penetrate the internal network. There is no single vulnerable point that will compromise the internal network.

It is also possible to create a layered series of perimeter networks between the outside world and the interior network. Less trusted (more vulnerable) services are placed onto perimeter networks, removed from the interior network.

The two screening routers serve the following functions:

Interior router (choke router)

- protects the internal network from the Internet and DMZ. It performs most of the packet filtering for the firewall
- allows selected services outbound from the internal network to the Internet (services that don't go through proxies, such as telnet or ftp).
- limits the services between a bastion host and the internal network. This reduces the number of machines that can be attacked from the bastion host if it is compromised.

Exterior router (access router)

- protects both the DMZ and the internal network from the Internet.
- generally allows almost anything outbound from the DMZ (performing little packet filtering)
- generally performs the same outbound rules as the interior router (allowing any internal packets to get out)
- this router is often provided by the ISP and should have rules sufficient to protect the machines in the DMZ, disallow forged packets

(which appear to come from the internal network), and allow access only to the services provided by the DMZ.

To support proxies, the interior router should allow packets from the internal network if they are communicating with the bastion host. The exterior router should allow packets only from the bastion host to the outside.

The basic principles of firewalling are:

- It is easier to secure one or a few machines than a large number of machines on a LAN.
- Focus effort on the bastion host(s) since only they are accessible from the external network.
- All traffic between the outside and inside must pass through a firewall.
- The internal (private) network should never see security attacks.

TCP WRAPPERS

TCP wrappers are a mechanism orthogonal to firewalls and packet filters that is used to restrict access to TCP services on a particular host. The idea is to allow only certain originating IP addresses to execute authorized services. All requests can be monitored and logged.

On most Unix environments, a process called *inetd* listens for connections to an enumerated set of services. These services may include ftp, rpc, rsh, rlogin, telnet, rcp, and others. When a client connects to the listening *inetd* process, *inetd* simply executes the requested process (based on port) and gives the socket to that process. With TCP wrappers in place, *inetd* is replaced with a new daemon process, *tcpd*. This process behaves just like *inetd* but instead of blindly passing connections to the requested services, *tcpd* first checks permission files to validate that the requesting host is indeed allowed access to the service or is explicitly disallowed access.

VIRTUAL PRIVATE NETWORKS (VPN)

A Virtual Private Network (VPN) is an alternative to a dedicated communication line between two points (a private network). A VPN provides users with the illusion that they are directly and privately connected to a remote network via a private network even though a public Internet infrastructure is used in the connection.

Let us first take a look at private networks. If two networks are connected via a private line, the router at each network will be configured to route packets destined to the other network over the private line.

A virtual private network relies on *tunneling*. Tunneling links two network devices such that the devices appear to exist on a common, private backbone. It is accomplished by taking all IP packets destined to machines on the remote network and

encapsulating them within an IP header directed to a router that listens on the remote side. As far as the connecting [public] network is concerned, all data is just a set of IP datagrams going to one port on one remote machine.

When the remote system (router) gets the packet, it extracts the data, which happens to be a full IP packet that belongs on that network. It is then placed on the network and sent to its destination within the network.

The Internet standard for VPNs is known as IPSEC and is described in RFCs 1825-1827. It provides IP-layer security mechanisms for both IPv4 and IPv6. These include authentication and encryption. IPSEC comprises two parts: the authentication header (AH) and the encapsulating security payload (ESP). The authentication header is essentially a digital signature for the packet. It is an extra header that is added to the packet that contains an encrypted hash of the unchanging part of the packet (e.g., fields such as a decrementing time-to-live are not part of the hash). IPSEC uses a shared secret key for its encryption. The ESP allows each outgoing IP datagram to be encrypted. This encrypted data is placed within an unencrypted IP header for routing the datagram to its destination over the public network. Upon receipt, the data is decrypted and the resultant, encapsulated, IP datagram is routed within the private network. A protocol that is conceptually similar to IPSEC is PPTP, the point-to-point tunneling protocol. This is an extension to PPP (point-to-point protocol) that was developed by Microsoft. It encapsulates not just IP packets, but also IPX and NetBEUI packets. Because of its past flawed security and the less prevalent support among routers, it is not as widely used as IPSEC.

REFERENCES

Building Secure and Reliable Network Applications, Kenneth P. Birman, © 1996 Manning Publications Co.

Firewalls and Internet Security, William R. Cheswick and Steven M. Bellovin, Addison-Wesley © 1994 AT&T Bell Laboratories.

Internet Authentication Service for Windows 2000, White Paper, © 2000 Microsoft Corporation.

Sequence Number Attacks, by Rik Farrow, UnixWorld.
www.networkcomputing.com/unixworld/security/001.txt.html