

The Enemy Within: Firewalls and Backdoors

Bob Rudis and Phil Kostenbader 2003-06-09

Can your security infrastructure protect you when you've left the key under the mat?

As a modern IT professional you've done all the right things to keep the "bad guys" out: you protected your network with firewalls and/or proxies, deployed anti-virus software across all platforms, and secured your mobile workstations with personal firewalls. You may even be in the process of designing and deploying an enterprise-wide network and host intrusion detection framework to help keep an even closer eye on what's going on. Even with all this, are you *really* safe? Can your multiple-lines of defense *truly* protect your network from modern methods of intrusion?

This article presents an overview of modern backdoor techniques, discusses how they can be used to bypass the security infrastructure that exists in most network deployments and issues a wake-up call for those relying on current technologies to safeguard their systems/networks.

The Fundamentals of Firewalls

Before a discussion of modern backdoor techniques can take place, it is necessary to first look at what obstacles an attacker must get through. Firewalls are an integral part of a comprehensive security framework for your network. If they are relied on too heavily they can also be the weakest link in your defense strategy.

There are different flavors/combinations of "standard" firewalls to choose from depending on your environment:

Packet filters [1]

- Operates at Layer 3
- Also known as Port-based firewalls
- Each packet is compared to against a list of rules (source/destination address, source/destination port, protocol)
- Inexpensive and fast, but least secure
- 20-year old technology
- Breaks more complex applications (e.g. FTP)

- Example: router access control lists (ACL)

Circuit-level gateways

- Operates at Layer 4
- Relay TCP connections based on port
- Inexpensive but more secure than packet filters
- Generally requires work on the user or application configuration end to support
- Example: SOCKS-based firewalls

Application-level gateways [2]

- Operates at Layer 5
- Application-specific
- Moderately expensive and slower, but more secure and enables user activity logging
- Generally requires work on the user, network or application-configuration end to support
- Example: Web (http) proxy

Stateful, multi-layer inspection firewalls [3]

- Layer 3 filtering
- Layer 4 validation
- Layer 5 inspection
- High level of cost, security and complexity
- Example: CheckPoint Firewall-1

Some newer firewall technologies build upon these foundations and provide additional ways of securing both systems and networks:

"Personal"/host firewalls

This class of firewall has the ability to further enhance security by enabling granular control over what types of system functions and processes have access to networking resources. These firewalls can use various types of signatures and host conditions to allow or deny traffic. Some of the more common functions across personal firewall implementations include:

- *Protocol-driver blocking* - disallow "non-standard" protocol drivers to be loaded and used

by programs

- *Application-level blocking* - only allow certain applications or libraries to perform network actions or accept incoming connections
- *Signature-based blocking* - constantly monitor the network traffic and block all known attacks from making it to the host

The added control increases the difficulty of managing security due to the potentially large numbers of systems that may be individually firewalled. It also increases the risk of damage and exposure due to misconfiguration.

Dynamic Network Firewalling

Similar to the signature-based blocking features of personal firewalls, dynamic network firewalling marries the concepts of IDS, standard firewalls (see above) and emerging intrusion prevention techniques to provide "on-the-fly" blocking of specific network connections that fit a defined profile while allowing connections from other sources to the same port(s). This allows a firewall to proactively deny access to, say, clients that are issuing SQL worm attacks against your network while still allowing standard SQL traffic to flow.

The Basics of Backdoors

What is a backdoor? A backdoor is a "*mechanism surreptitiously introduced into a computer system to facilitate unauthorized access to the system,*"[\[4\]](#) and can be classified into (at least) three categories:

Active

Active backdoors originate outbound connections to one or more hosts. These connections can either provide full, fluid network access between the hosts (i.e. reverse tunnel-based) or be part of a process that actively monitors the compromised system, records information, sends data out in distinct "chunks" and receives both acknowledgements and/or commands from the remote systems.

Passive

Passive backdoors listen on one or more ports for incoming connections from one or more

hosts. Similar to the active backdoors, these programs can either be used to establish a forward tunnel into the compromised network or accept distinct commands and return the requested information.

Attack-based

This category of backdoor could also be classified as the "unknown backdoor." It generally arises from a buffer-overflow exploit of poorly-written programs resulting in some type (e.g. root/Administrator-level, user-level, fully-interactive, one-instruction) of command-level access to the compromised system.

There is one common element among the three types of backdoors - they all work by *circumventing the elaborate multi-layer security infrastructure* you have worked diligently to design and deploy. Most real (i.e. non-script-kiddies) hackers can determine almost immediately if it's worth attempting to meet your perimeter routers and firewalls with a head-on attack. Textbook methods can be relatively easily employed to help discover the types and configurations equipment protecting the borders of your network. Some of these discovery tools can even help detect the presence of proactive network intrusion detection systems (IDS). While there are still daily exceptions, most perimeter networks are configured well enough to make backdoors the emerging method-of-choice for deep-network penetration for a number of reasons:

They avoid immediate detection by well-configured firewalls, network & host IDS.

A perimeter attack will (or should) make your operations consoles light up like a Christmas tree. There is no such thing as a casual or accidental scan of open firewall ports. If you don't have a penetration test scheduled, chances are that you're being probed.

Some proactive environments will immediately lock-out the originating systems' IP address when these scans are detected. Even if this is not the case, risking detection removes the primary reason for getting into your environment: the ability to operate freely and without notice.

They don't rely on potentially hard-to-duplicate, specialized attack methods.

What is more difficult: constructing the precise SYN-Frag attack necessary to cause a buffer-

overflow in a CheckPoint firewall (that is two revisions behind the latest patch-level) to render it as helpless as a router without ACLs, or getting an unwitting user to open up an e-mail attachment?

To make it past the outer defenses, it might require the use of 4-6 of these specialized attack methods with no guarantees that one of them won't cause a crash and reboot, rendering the entire attempt useless.

They take advantage of the myriad of exploits available in the soft underbelly of an organization's internal network.

How many Microsoft Windows-based workstations and servers are in your organization? How many *nix systems do you have? How many users do you have with each of these types of systems? How many routers, firewalls and IDS systems do you have?

Chances are significantly higher that in most organizations a hacker will have a much easier time finding an un-patched Windows or *nix system to exploit than they will an un-patched and/or misconfigured piece of perimeter networking/security equipment.

An Inside Job

While this article has presented the concept of backdoors in the context of external penetration attempts, they are not limited to that narrow area of practice. Backdoors can be used by employees, contractors or planted-workers to provide less restrictive and undetectable "remote access" points all across your network.

Regardless of the type of backdoor, there are two primary ways of injecting them into your network. The first method involves getting a user to inadvertently load and run the program on their system(s). Extremely common examples of this include e-mail attachments that exploit un-patched vulnerabilities in client systems, web sites/downloads that have an unexpected/hidden payload, and programs that fall into the classification of "spyware". Unfortunately, these methods are all too common and can result in serious loss of confidentiality and privacy. In the case of "spyware", programs are installed, registry keys are inserted and browser cookies are set that enable the tracking of every network-based move a user makes. This tracking is not limited to Internet sites, which thus make it very easy for these systems to map out all the important places on a company's intranet. While the majority of the "spyware" programs are used to present and track your viewing of web ads, others can be crafted to be sentinels to

alert remote sites of your online/offline status, complete with current network connection information.

Even without loading malicious "spyware" backdoors, a user can still be susceptible to a more corporate form of backdoor. Real Networks player performs constant communication to its home network and is nearly impossible to deactivate without reinstalling. Microsoft XP users have the ability to be tracked by either enabling automatic updates or just having their time kept in sync by Microsoft's own time server.

The second method involves actually being on your network in the first place. A trivial example would be installing a custom-program which has a programmer-created backdoor embedded in it. These types of backdoors can be malicious, but they are usually coded as a means of circumventing standard software development processes in order to save time.

A more typical, network-level, generic example would be one which is used to bypass remote access restrictions. This may be the oldest form (relative to the early stages of the Internet) of backdoor, initially used to bypass inbound telnet/rlogin restrictions. The setup is rather straightforward: a user installs a program that doesn't require elevated privileges to execute, then the program is run and it waits for connections on a port that isn't blocked by upstream access control devices. This remote access could be to a multi-user system or to an individual's workstation. Initially only Unix-oriented, these types of programs can be difficult to detect.

These types of backdoors are easier to understand in the context of concrete examples:

| | |
|----------------------|---------------------------------------------------------------------------------------------------------------------|
| Program: | BindShell |
| Available at: | http://hysteria.sk/sd/f/junk/bindshell/bindshell.c |
| Type: | PASSIVE |

This program is easily modified to run on any defined port - for this example, TCP 1234 - and doesn't support a password, thus allowing anyone access. To access this service, the remote user simply starts a telnet session to the desired host and specifies a port number:

```
telnet some.insecure.host.org 1234
```

Variations of this program can also be found at <http://packetstormsecurity.nl/> which support UDP connections and encrypted sessions.

There are several techniques that can be used to attempt to detect this, none of which will provide simple or direct isolation. In all cases knowledge of the normal run state of the OS is necessary.

- '**netstat -a**' is a program that comes as part of the UNIX operating system and is used to display network port connection status. One would look for port usage that isn't part of the normal run state.
- '**nmap**'^[5] or '**strobe**'^[6] external port scanners could be used to identify active or listening ports. Again, knowledge of a normal run state would be extremely helpful.
- '**lsof -i**'^[7] a public domain program, can be used to list all open files and their resource usage. One would search the output for users running unusual programs that require the use of networking ports.

| | |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Program: | Sneakin |
| Available at: | http://packetstormsecurity.org/Exploit_Code_Archive/sneakin.tgz |
| Type: | ACTIVE |

This program requires elevated privileges and basically waits for two specially-crafted ICMP packets to arrive before starting something very similar to a reverse telnet session which establishes a connection to a remote machine. **Sneakin** requires LINUX and **netcat**^[8].

The "listening" state is just as difficult to detect as in the above example. A conventional external port scan will not work since the program intercepts and processes ICMP packets while still allowing access to them by the native operating system kernel. **LSOF**, however will show a process accessing the network adapter in promiscuous mode. In general, **LSOF** might be the best tool available to detect NICs in this state. **Netstat** will also provide a clue to this particular backdoor, as it will show two ICMP ports using the raw protocol. Once "**sneakin**" enters it's ACTIVE state, additional processes using network ports will show up in **LSOF** and **Netstat** output.

| | |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| Program: | GIFtpD |
| Available at: | http://www.security-express.com/archives/bugtaq/1999-q4/0443.html |

Type:**ATTACK**

GIFtpD is one of the standard examples of an attack-based backdoor. The premise behind it is simple: an attacker would take advantage of a few misconfigured features of an ftp server, allowing them to deposit and execute backdoor code, in this case **BindShell**. A weak inbound policy combined with un-proxied, weak outbound policies do the rest.

Sneakin and **bindshell** are classic tools used against weak inbound firewall policies. Many sites deploy extremely strong inbound policies, making it difficult to gain direct access to the listening ports. Without direct access, a large number of backdoors cannot be exploited. However, the strongest inbound policy can be easily defeated by active backdoors using "tunneling" methodologies. A tunnel, in the context of backdoors, is best explained as a program that sits on the inside of a protected network and establishes an outbound connection to an external host which results in the flow of bi-directional traffic between these systems and/or networks. This is a serious threat to even the most modern security architectures. A popular example of such communications would be to create an encrypted network connection between two hosts using VPN software.

Properly configured, a VPN tunnel will allow total and unrestricted access to the networks that the hosts are gateways for. When provided as a legitimate remote access tool for employees and business partners, VPNs can increase productivity, save time and reduce costs. When they are used to exploit gaps in the security architecture, they can have just the opposite effect.

VPN technology is still fairly new and requires more than casual knowledge to setup and maintain when used legitimately. The learning curve is even steeper when they are being used as a backdoor tool. You don't need a VPN for a tunnel. Taking a step back, it is possible to connect just two hosts using more traditional and widely known software - secure shell. Secure shell - or **SSH**[\[9\]](#) as it is more commonly referenced - can be used to establish a tunnel between two hosts by allowing the redirection of a port on the client (outside the firewall) to a port on the host (behind the firewall). For example, one could redirect a client port 2200 to host port 23. Assuming the user is currently accessing the client (outside the firewall), they would telnet to the localhost port 2200 and get port 23 on the remote host (behind the firewall). A weak outbound policy allows the connection to be generated from the host behind the firewall. This is a neat and popular trick.

In the same scenario it is also fairly straightforward to provide access to an organization's

internal web sites. The user would simply install a copy of a proxying agent - e.g. "**squid**"[\[10\]](#) web proxy or the Apache "**httpd**"[\[11\]](#) daemon with proxy support compiled in - on some internal system. The standard software configuration could be used for either agent. The user would then use SSH port redirection to connect client port 3128 to host port 3128. The client, again outside the firewall, now has proxied access to the organization's internal web servers thru proxy port 3128.

This example can be extended further to enable more than one external host to have access to the internal web sites. The addition of a simple port redirector[\[12\]](#) can make the tunneled, proxied connection available (on port 3128) to all users of the remote network.

Conventional techniques will not work in identifying the existence of this type of tunnel. Depending on the platform used, one could monitor network usage and look for consistent or seemingly permanent processes with established network connections to the outside. At a host level, identifying backdoors in this manner would necessitate the building and maintenance of a baseline network usage state (possibly using the tools mentioned earlier). It is also possible to query the boundary firewalls and monitor the connection state tables, focusing on these established connections. Either process is a daunting task in busy/large environments.

A Ready Defense

There is little one can do to completely defend their network from the use of backdoors. The current set of tools - whether it be host or network IDS - are difficult to configure, deploy and use effectively, especially in large organizations. Without the development of special-purpose tools, expressly designed to monitor systems and networks for the presence of backdoors, the only way to defend against these techniques is through a change in thinking. Security managers who think they can simply hide their networks behind a firewall, sit back and declare that "nobody can get in, I closed all the doors" need to take a hard look at their line of thinking. A good defense against backdoors needs to start with a change in network access philosophy. A solid beginning would be to develop strong Internet access policies and implement technologies that limit outbound access via well-configured firewall/outbound-access architectures.

At a network level, stopping backdoors means making it very difficult for them to establish connections outside of your infrastructure. One approach would be to use circuit-level gateways (i.e. SOCKS/port redirection) as a means of restricting backdoors from using high (or any) TCP ports. With simple port redirection, network requests destined for an external endpoint are

terminated internally at a device which makes the connection on its behalf to a pre-defined endpoint. While this limits the number of external resources applications can access, it can also create additional administrative and processing overhead and may not work for all applications. With modern SOCKS gateways, the administration can be done on a global policy level with little impact on performance and almost no impact on applications.

An alternative approach would be through the use of (again) highly restrictive outbound access policies - where very few direct outbound connections are allowed - and Web/application-specific proxies that force authentication before access is enabled. The goal is similar to port-redirection: stop unchecked access to external hosts. While most traditional security schools would like nothing more than to close all the doors and windows, modern businesses need access to external resources to function. Unfortunately, *almost* any outbound access mechanism can potentially be used to provide a conduit for backdoors. Proxy-based architectures enable granular control over what is allowed outside of your network since applications need to "speak the right language" to be permitted access. Tunnels can be established[13] through proxies (especially via SSL connections[14]) but they are much harder to configure, deploy correctly and rely on. With authentication thrown into the mix, a way now exists to identify all connections down to the source (user). All the pieces are then in place to deter (where possible), detect and discover backdoors.

Even with these techniques - which will require time and resources to implement - developing a network access architecture which makes it easy for users to get work done *and* difficult for backdoors to do their job is not a trivial endeavor. Fundamentally, your aim should be to design an infrastructure that makes it as efficient as possible to tie network connections to users while narrowing down the options for the backdoors.

[1]The Packet Filter: A Basic Network Security Tool - http://www.sans.org/rr/firewall/packet_filter.php

[2]Application-Level Firewalls: Smaller Net, Tighter Filter - <http://www.networkcomputing.com/1405/1405f3.html>

[3]Anatomy of a Stateful Firewall - <http://www.sans.org/rr/firewall/anatomy.php>

[4]Detecting Backdoors - <http://www.icir.org/vern/papers/backdoor/>

[5]nmap home - <http://www.insecure.org/>

[6]strobe source code - <http://www.packetstormsecurity.org/UNIX/scanners/strobe-1.04.tgz>

[7]lsof main distribution - <ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof>

[8]Netcat - The TCP/IP Swiss Army Knife - <http://www.sans.org/rr/audit/netcat.php>

[9]OpenSSH home - <http://www.openssh.org/>

[10]Squid Web Proxy Cache home - [href="http://www.squid-cache.org/"](http://www.squid-cache.org/)

[11]Apache httpd Project - <http://httpd.apache.org/>

[12]Port Forwarding Tools - <http://nucleo.freesevers.com/portfwd/tools.html>

[13]rwwwshell - <http://www.thc.org/releases/rwwwshell-2.0.pl.gz>

[14]ssh-tunnel.pl - <http://www.fwtk.org/fwtk/patches/ssh-tunnel.pl>

[Privacy Statement](#)

Copyright 2006, SecurityFocus