Sidewinder™: Defense in Depth using Type Enforcement

Dr. Richard E. Smith
Secure Computing Corporation

Abstract

Sites use firewalls to defend against external attacks while providing necessary Internet services. Firewalls make a site safer: they present a smaller risk since they provide fewer services. However, most firewalls use standard computer operating systems. This can allow an attacker to overrun the firewall if a known security flaw is present. The ***Sidewinder*** firewall system overcomes this problem using type enforcement. Network server applications operate in independently controlled compartments called domains, each granted specific permissions to access particular types of files or communicate with other domains. If a server succumbs to an attack, type enforcement restricts the amount of damage an attacker can do. In particular, ***Sidewinder*** prevents an attack on an Internet server from accessing domains serving internal, protected networks. An attacker can not overrun a ***Sidewinder*** because the type enforcement restrictions can not be disabled while the system is handling network traffic.

Introduction

Internet firewalls are intended to protect sites from external attack. Experience has shown that attacks are based on fundamental weaknesses in Internet protocols or in the host computers providing the network services. ***Sidewinder*** counters these fundamental weaknesses using *type enforcement*, a distinctive security mechanism added to its standard BSD Unix kernel. Type enforcement provides a basic containment mechanism that prevents Internet attacks from spreading across the firewall to the internal, protected network. This makes ***Sidewinder*** resistant to attacks that would succeed against weaker systems.

Internet firewalls provide a valuable service: they focus the security problems of an Internet connection in a single box. This allows an organization to focus their network security efforts. The alternative has proven too costly: organizations always have more networked computers than they can afford to individually secure. A firewall is a dedicated Internet access device under the direct control of the individuals responsible for network security. Better firewalls also monitor Internet traffic, alerting administrators of trouble when it occurs.

Firewalls are intended to block attacks from the Internet. Known types of attacks have been published in a variety of places: Cheswick and Bellovin's well known book provides a thorough list (*Firewalls and Internet Security*, Addison-Wesley, 1994). Warnings related to recent security incidents are provided by the Computer Emergency Response Team (CERT); the CERT Advisories were initiated after the Internet Worm attack of late 1988.

In practice, few network security experts are surprised when a CERT Advisory is issued. There are numerous widely recognized weaknesses in the nature of Internet services and the hosts providing them. New attacks always exploit a known type of weakness, though generally in a new and clever way. For example, the "IP Spoofing" incidents reported in early 1995 were based on Internet protocol vulnerabilities described in reports ten years earlier. Experts know that it is only a matter of time before any particular weakness is going to be exploited by an attacker.

While firewalls are intended to defeat the attacks brought on by these basic weaknesses, many firewalls are themselves vulnerable to them. A firewall's first line of defense has often been surprise: the attacker attempted to hit a particular type of computer system (Unix or a PC with particular Internet software) but did not know how to attack the intervening firewall. History shows that any defense based on surprise or on the attacker's unfamiliarity will not last long. Furthermore, many firewalls are built atop vulnerable, standard Unix systems. Without a basic containment mechanism, an attacker can overrun a firewall and use it as a platform to freely attack the protected network.

Sidewinder™ and Type Enforcement

Sidewinder was specifically designed for commercial security applications. Secure Computing originally developed type enforcement for high security government applications. The original type enforced products are the only ones trusted by the US Government to transfer message traffic between the Internet and classified computer networks. This represents an unparalleled level of trust in a network security device. To reduce costs and complexity for commercial users, ***Sidewinder*** is hosted on a commercial grade Pentium tower system. The operating system is based on the BSDI version of Unix, security hardened using type enforcement and omitting military security labels.

Type enforcement allows ***Sidewinder*** to execute standard software in independent, restricted compartments. This provides a layered defense similar to watertight compartments on a large ship. Other compartments remain safe and dry even if some compartments are breached. Under type enforcement, software in a compartment, or *domain*, is granted access to all procedures and data it needs, and is forbidden to access anything else. If service software contains a security flaw, the type enforcement permissions limit the possible damage of an attack. In particular, a security breach in an Internet application's domain will not yield access to domains on the internal network. This protects the internal network even if application security failures occur.

The type enforcement mechanism also provides a "silent alarm" mechanism that automatically identifies and logs any attempts to violate type enforcement restrictions. This generally corresponds to attack behavior, since the type enforcement permissions are configured to permit all accesses that a software component will normally need to do. The type enforcement violation can automatically notify an administrator through electronic mail or a pager, and can be configured according to a variety of criteria.

An important feature of ***Sidewinder's***™ type enforcement is that it establishes *mandatory protection*, that is, programs are generally forbidden from changing security critical type enforcement permissions associated with programs or files. This is different from typical access protections in commercial systems. For user and software operating convenience, most operating systems allow users and programs to change access permissions under fairly broad conditions. Many systems even have "super user" modes in which a user or program may modify any permission setting on the system. ***Sidewinder's*** type enforcement protections are not so easily changed. Critical settings may only be changed when the system is in a special administrative mode with network access disabled. This prevents an external attacker from modifying mandatory access permissions.

The next two sections of this report will examine the general classes of security weaknesses inherent in network software and the general strategies to combat them. One strategy is reactive: it eliminates previously reported problems in detail without addressing the general problems allowing the attacks. Typically, it is difficult or impossible to fix a general problem in a large, mature software component like an Internet server. Another strategy is to try to replace or otherwise modify the server's interactions with the network in order to increase its security. This is somewhat proactive, but can not address all weaknesses. ***Sidewinder's*** strategy is to directly block the basic attacks using type enforcement. The last section describes how the type enforcement mechanism specifically blocks each type of attack.

Generic Network Software Weaknesses

Network software security flaws relevant to Internet attacks fall into four general categories. This summary is based on a variety of sources and analyses, including historical and theoretical evaluations as well as actual attacks and CERT reports. Often an attack exploits several weaknesses simultaneously to gain its objective.

1. **Weak authentication:** security decisions are based on vulnerable identification data.
2. **Excessive privilege:** network software operates in an excessively privileged state (i.e. as "super user").
3. **File vulnerability:** an attacker can view or modify security critical files and logs.
4. **Subverted execution:** program commands or instructions embedded in externally supplied data are somehow executed by the system.

These problems are the province of any host system connected to the Internet. These weaknesses increase the risk of an attack. They may also increase the risk that an attacker can completely overrun the host. The following paragraphs examine each weakness and how it is exploited by an attacker.

Weak authentication

A network sever makes many of its decisions according to the identity of the person asking for service. The server must correctly *authenticate* an individual asking for service to ensure that the user is correctly identified. If the authentication mechanism is weak, then security decisions based on identity are also weak. For many applications, weak authentication is not a problem. It becomes a problem when security critical decisions are based on unreliable data.

This problem depends largely on the protocols used to communicate between a host and its authorized users. The host must be able to reliably distinguish between messages from an authorized user and messages from an attacker. Otherwise, the host can not make appropriate security decisions on that user's behalf. Secret passwords used to be reasonably effective; they are no longer safe to use on the Internet. Unfortunately, secret passwords are the only authentication mechanism supported by older protocols like the File Transfer Protocol (FTP). This weakness can not be fixed with underlying host security capabilities. It must be fixed by improving the strength of the authentication protocols used, or by restricting the protocol to safe uses like anonymous FTP.

The “IP Spoofing” attack provides another example of weak authentication. The system under attack receives messages that claim to originate on a “trusted” host. In fact, the host is receiving messages from the attacker containing falsified return addresses. The host under attack becomes a victim if it trusts the return address, as is common in several protocols. The attacker then feeds the victim a message to open access for a broader attack.

Many systems worsen the threat of weak authentication by allowing users to perform practically any action from a weakly authenticated session. In earlier days this was considered liberating: systems could be administered remotely, saving a trip to work in the middle of the night. But attackers have exploited this convenience and have overrun systems by masquerading as an administrator.

Excessive privilege

Many network software components operate with most or all host security protections disabled. This is true of most network software on most commercial systems, even though most microprocessors can provide significant protection. Personal computers running DOS or Windows, or the Macintosh operating system do not always use the processor’s access control systems, so misbehaving software is capable of doing practically anything to other parts of the system.

Historically, Unix network server software has run in “root mode” with all Unix access protections disabled. This allowed the servers to operate on the behalf of a user based on messages received from the network. A more modern Unix server operates under more sensible restrictions, but the tradition has been hard to overcome in practice. Existing server software often fails when its presumed access permissions are unexpectedly restricted. Similar problems have been observed in other multiuser systems. Simply the existence of an all powerful “super user” mode is a security risk since an attacker operating in that mode can do unbounded damage.

File vulnerability

This weakness has led to numerous security breaches, most dramatically on multiuser systems. Systems often base their security decisions on the contents of data files. Those decisions can be altered by software that exploits those files. For many years, the most common way to breach a multiuser system was to find a weakly protected file with interesting security properties. A file might divulge secret passwords, or a list of permissions might not be fully protected from modification. An attacker can erase all traces of an attack by modifying records in poorly protected log files.

A well known trick to exploit this weakness is called the "Trojan Horse," a piece of software that surreptitiously reads or modifies a file when run by an individual with access to that file. Some strains of computer viruses play similar tricks. Multiuser systems often resist viruses, and instead fall to Trojan Horses hidden in a user's operating environment.

Subverted Execution

Network server software is designed so that the site can provide specific services to clients. Few organizations intend to provide arbitrary access to their computing system via the network servers. Unfortunately, security flaws can yield such behavior. In some cases, an attacker feeds data to the server (or to other network software) that gets passed to a system command interpreter like the Unix shell. In other cases, the attacker actually feeds binary executable data to the system and exploits a bug to cause that data to execute. In both cases the attacker can give the system arbitrary instructions or operating commands, causing arbitrary damage or mischief.

Interpreters are a fundamental part of all computer systems. Interactive command shells provide basic system services to users. Interpreted languages like LISP, PERL, sed, awk, and Postscript provide valuable services to their users. Commercial spreadsheets and database systems have elaborate built in interpreters for user customized programming and data analysis. However, the power of these systems allow them to be used to launch attacks.

Many systems also allow computer programs to construct and submit interactive commands on the user's behalf. The Unix command shell is a particularly good example: the "system()" procedure accepts a text string and passes it to the shell for execution. This mechanism makes it easy for one program to pass work to a separate program, simply by constructing the command with the appropriate list of arguments.

Unfortunately, this Unix shell interface has succumbed to clever abuse. A server program might extract data from an incoming message and pass it to another program using the "system()" function. If an attacker cleverly constructs the message's contents, the shell can be made to execute an arbitrary series of commands.

Other interpreters pose security risks, too. Any interpreter that can access and modify data could pose a threat, particularly if it is invoked automatically. This includes Postscript and the Multipurpose Internet Mail Extension (MIME), both of which could be programmed to access files. However, the services provided by interpreters are too valuable to be eliminated from computing systems. They provide vital services to users. They are easy to purchase, install and use. Their productive value usually justifies their existence, regardless of security risks they may pose.

Computer viruses are the most common case of binary execution attacks. Typically the virus is inserted into an executable file belonging to an application. When the application executes, a modified piece of the application's binary data redirects the instruction stream to the binary data comprising the virus. Once the virus has finished executing, it may branch back to the application, allowing the application to continue operation. Weak protections make viruses particularly common on personal computers.

While viruses have primarily infested personal computers, multiuser systems have also suffered from attacks with subverted binaries. The Internet Worm attack of 1988 included a style of penetration using binary data. A 1995 CERT Advisory reported a similar attack against certain World Wide Web servers. In these attacks, an unusually long message would overrun a data area and manage to overwrite an address pointing to executable binary code. The overwritten address would invoke binary code to perform the attack.

Typical virus-borne binary execution attacks simply act as a form of electronic graffiti, but such attacks could do serious damage to a system. The attack program could do anything to the system that any software is allowed to do. Such programs can access or modify files, reformat entire disks, access communications lines, or use any other unprotected resources the system may have.

Security Countermeasures

The weaknesses described above often provide ways to subvert network server software either on a service host or on a firewall. It is not enough to declare a host to be a firewall: it will still be vulnerable to attack unless special measures are taken to ensure its integrity.

Packet level filters are effective against certain specific weaknesses. A properly configured router, for example, can automatically detect and reject the spuriously addressed messages used in the “IP Spoofing” attack. However, reports during such attacks in early 1995 indicated that routers were not always correctly configured to detect and block the spurious packets. Attacks embedded in messages for an authorized service (electronic mail, for example) can flow right through packet level filtering and still attack a vulnerable server.

There are three general approaches to countering the weaknesses. The first is the reactive approach of fixing security holes as they are found, and hopefully before they are used to attack the site. The second is slightly more proactive: simplification of the software that faces the threat on the sound assumption that simpler software is more predictable. The third approach is to provide a fundamental security mechanism that overcomes the weaknesses. Each approach is described below. ***Sidewinder***™ is the only system that combines all three approaches, providing true defense in depth.

Reactive Defense

A reactive defense is one that tries to find and fix security bugs in standard programs (e.g. sendmail) as soon as they appear. This is the most common approach to computer and network security. The CERT Advisory mechanism is designed to support this: typical advisories identify software patches to fix the reported security problems. Systems following this approach deal with the weaknesses in an iterative fashion: as each weakness manifests itself in a new way, a fix is developed and applied to the software.

Computer security experts refer to this as the “Penetrate and Patch” cycle. Experience has shown that there is no way to tell when all problems have been found and fixed. This is especially difficult with a large suite of network server software. Each software component has almost certainly been developed by a separate team of people. Security requirements for both the server software and the protocol itself were often poorly specified or nonexistent. Security flaws inevitably appear under such conditions.

This approach suffers from two serious problems. First, each security flaw leaves the system is vulnerable to attack until the flaw is identified and a fix installed. Attackers could theoretically hit every system vulnerable to a particular flaw before the problem is diagnosed and fixed. In 1988 it often took months, if ever, for vendors to provide fixes for known security problems; the Internet Worm exploited this shortcoming to infect several thousand systems in a matter of hours. A second serious problem is that bug fixing is itself a source of risk. Computer industry experience is that an average of one new flaw is introduced for every two flaws fixed. It is also crucial to ensure that the changes made to the software are in fact intended to fix the flaw. A software “fix” from a questionable source represents a powerful form of attack.

Simplification

This approach replaces standard programs with simpler, less capable versions, reflecting the design wisdom that simpler is better. Both intuition and experience show that it is easier to do a simple task effectively. A simpler program will have fewer modes of operation, fewer execution paths, and make fewer decisions that may have security implications. This is more proactive than the bug fixing approach. The author of the safer program can apply security experience from the more complex program and generalize it to defend against a broader set of attacks.

Simplification captures the essential philosophy of firewalls in general. Unlike a general purpose server host, a firewall presents a limited face to the source of attacks. Unnecessary or risky services are disabled and omitted. The only software installed and executed on the system is the approved set of network services, filters, and security software. No unnecessary users have access to the system. The configuration is kept as simple as possible to ensure it is done securely and correctly.

A variant of this approach is to provide a layer of software between vulnerable server software and the source of attacks. The layer of software can perform security checks to potentially hostile network traffic, enforce access control restrictions, and collect security relevant information about server traffic in the system log. The extra software layer provides protection, but it can also interfere with the expected behavior of the protected software. This can introduce its own vulnerabilities.

Another simplifying approach is to host the server software on a really simple platform such as a DOS based personal computer. This achieves a degree of simplicity by eliminating the elaborate layers of services available in systems like Unix or VAX/VMS. There is no danger of improperly configuring access permissions if there are no access permissions to configure. The site can focus on configuring the software providing the actual network services since practically no other software is present on the system. Unfortunately, this approach leaves the system vulnerable to many threats associated with excessive privilege, since PC based systems can not effectively restrict software access privileges. This approach may also restrict the capabilities provided by network server software, since all capabilities might not port effectively to a PC environment.

Fundamental mechanism (i.e. Type Enforcement)

This approach implements a fundamental mechanism in the system to block the effects of security weaknesses. ***Sidewinder***™ uses type enforcement to do this. Computer security experts have long applied the concept of *mandatory protection* to situations requiring high security. It is the cornerstone of US Government standards for protecting classified information on shared computing systems. Mandatory protection refers to technical mechanisms that prevent programs from bypassing particular security controls. ***Sidewinder's*** type enforcement is a form of mandatory protection: it establishes the rules by which server software may access network connections, files, terminals, and so on. These rules can not be modified or violated while the system is processing network traffic.

Mandatory rules were originally established to prevent Trojan Horse attacks in systems designed to handle military secrets. The type enforcement concept extends mandatory protection to construct software compartments or *domains* within which programs can run. This prevents network server software from stepping outside its intended modes of operation. Thus, the system can detect an attempt by network software to access the system password file or other security relevant file by excluding those files from the domain's access permissions. Since these rules are mandatory, an attacker can not trick the network server software or ***Sidewinder***™ itself into disabling them as part of an attack.

Effectiveness of type enforcement

Experience has shown that security flaws will always exist in a system. The objective of a secure system is to prevent flaws or weaknesses from compromising its primary security objectives. For a firewall, the primary objective is always to protect the internal network from external attack. The following table reviews the likelihood that particular countermeasures will block future weaknesses of the different kinds. The type enforcement mechanism is the only countermeasure that is effective against all the generic weaknesses. The following paragraphs describe how type enforcement and other Sidewinder mechanisms are used to block the generic weaknesses.

Type of Generic Weakness or Vulnerability	**Firewall approaches to resolving vulnerabilities**			
	Unix Bug Fixing	**Unix with Simple Software**	**PCs with Simple Software**	**Sidewinder and Type Enforcement**
Weak Authentication	NO	NO	MAYBE	YES
Excessive Privilege	NO	MAYBE	NO	YES
Vulnerable Files	NO	MAYBE	NO	YES
Subverted Execution	NO	NO	NO	YES

Effectiveness of different firewall approaches to resolving vulnerabilities

Weak Authentication

This weakness is most dangerous when security decisions are made based on weak authentication. For example, Unix systems do not normally restrict access to “super user” status according to the physical properties of connections. It is more secure to restrict administration to directly connected terminals that are physically protected from unauthorized use. Although it is possible to implement stronger login measures, this could be disabled on conventional systems that do not have mandatory protections to control administrative access.

On ***Sidewinder***, a process can never perform administrative actions if the authentication is inadequate. ***Sidewinder*** uses its type enforcement for this in two ways. First, mandatory access permissions associated with type enforcement can only be modified by taking the system off of the networks and running administrative procedures via a directly connected terminal. No software on the system is capable of modifying these permissions while ***Sidewinder***™ is in its normal operating mode with the networks active. The system must be restarted in a “single user” mode before changes may be made.

Second, ***Sidewinder*** prevents other security critical accesses except through approved physical paths. Many administrative tasks may take place while the system is running; for example, it is impractical to shut down the system every time a new host or user identifier is added to access permission lists. The software to administer such things is configured so that it can only be executed via approved connections according to site requirements.

For example, some sites may require all administration to take place over directly connected terminals kept in locked machine rooms. Other sites may allow administration from one of the local networks. In all cases, ***Sidewinder*** will prevent administration from taking place via connections from the Internet. This prevents attackers from using administrative software to extend the scope of an attack.

Excessive privilege

Type enforcement defends specifically against this weakness. It applies protection to individual servers, protecting the system from further compromise even if an individual server suffers a security failure. ***Sidewinder*** also uses type enforcement to eliminate the risks inherent in “super user” access permissions capable of circumventing the system’s defenses.

The separate domains assigned to separate network servers allow specific assignment of privileges to individual server software components. Thus, the mail server has access to mail queues while the FTP or World Wide Web server do not. Furthermore, the permissions can reflect the expected access modes. For example, the Internet FTP and Web servers can be given read-only access to files approved for Internet release. The servers will not be able to deliver files to Internet users unless the files are of the correct type for Internet access. If either server suffers a failure or a security breach, the contents of released files are still protected from modification.

Sidewinder has no concept of a “super user” except when operating in the off line single user administrative mode. Thus, even if an attacker manages to subvert weak authentication through a network service, the attack can not proceed beyond the vulnerable server’s domain. The “super user” status acquired there is only effective within the server’s domain.

File vulnerability

Type enforcement addresses file vulnerabilities in several ways. Type enforcement permissions specify precisely what types of files may be accessed from each domain. As noted in the previous section, this can protect data files from modification by a network server, even if the server succumbs to a serious attack. This same approach is used to protect files that define access permissions enforced by server applications or by “filter programs” that pass information between servers on the Internet and those on the internal, protected networks. Only administrative software is granted permission to modify the files defining access permissions; the type enforcement database allows no modifications of such files from domains that run Internet server software. Network servers are granted no access at all to files they do not need.

Security critical files are attractive targets to attack since configuration errors occasionally leave them vulnerable. For that reason, ***Sidewinder***™ is built to detect unexpected attempts to access security critical files. Such attempts trigger a special type enforcement signal to log the event, or notify an operator, or invoke a special program. Typically, the event signals an operator who can initiate a trace of the connection to evaluate the user as a potential attacker.

Subverted execution

An attack exploits this weakness by feeding instructions or commands to the system to execute. Sidewinder blocks this type of attack in three ways. First, type enforcement restricts the domains in which command interpreters can execute. Second, it controls the accesses a process has, to limit the possible damage if the process goes completely out of control. Third, ***Sidewinder*** protects all of its executable binaries from modification so that subverted binaries can not remain in the system indefinitely.

An executable program, whether an interpreter or some other software component, can only execute in a given domain if the type enforcement database allows it to. For example, ***Sidewinder*** forbids Unix command shells from executing in server domains that do not actually require them. If a domain needs to use shells or other interpreters, ***Sidewinder*** limits its potential damage through file access controls and other measures previously described.

Conventional systems have no way to reliably prevent a particular program from running if the program is present on the system. Multiuser systems can make such attacks more difficult. The administrator can establish file permissions to restrict some users’ ability to execute particular files. However, many forms of attack can override these permissions.

Conclusion

Conventional systems do not provide fundamental protection against well known security weaknesses in network server software. ***Sidewinder*** uses type enforcement, a special set of mandatory access control rules, to block attacks based on these weaknesses. Type enforcement establishes a set of expectations for network server software behavior. The expectations identify which programs the server should execute, which files it will read, and what files it will create or modify. The server software is allowed to operate normally within the constraints of the type enforcement restrictions. If the server steps outside the bounds of its expected behavior, it violates a type enforcement restriction and generates a signal. ***Sidewinder*** uses this signal to look for potential attacks. Type enforcement is a form of *mandatory* access control because none of its normal operating software is allowed to change or bypass the type enforcement rules. This provides a level of protection unmatched by conventional systems.