The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment

David R. Safford, Douglas Lee Schales, and David K. Hess
Supercomputer Center
Texas A&M University
College Station, TX 77843-3363

1. Abstract

Texas A&M University (TAMU) UNIX computers came under coordinated attack in August 1992 from an organized group of internet crackers. This package of security tools represents the results of over seven months of development and testing of the software currently being used to protect the estimated 12,000 networked devices at TAMU (of which roughly 5,000 are IP devices). This package includes three related sets of tools: “drawbridge,” a powerful bridging filter package; “tiger,” a set of easy to use yet thorough machine checking programs; and “netlog,” a set of intrusion detection network monitoring programs.

2. Introduction

A Brief History of the Incidents

On Tuesday, 25 August 1992, the Texas A&M University Supercomputer Center (TAMUSC) was notified by the Ohio Supercomputer Center that a specific Texas A&M University (TAMU) machine was being used to attack one of their computers over the internet. The local machine turned out to be a Sun workstation in a faculty member's office. Unfortunately, this faculty member was out of town for a week, so rather than trying to gain access to the machine through the department head, it was decided to monitor network connections to the workstation and, if necessary, disconnect the machine from the net electronically. This decision to monitor the machine's sessions rather than immediately securing it turned out to be very fortunate, as this monitoring provided a wealth of information about the intruders and their methods.

The initial monitoring tools were very simple, but as the significance of what was occurring became apparent, the tools were rapidly improved to the point that the intruder's entire session could be watched in real time, keystroke by keystroke. This monitoring led to the discovery that several outside intruders were involved and that many other local machines had been compromised. One local machine had even been set up as a cracker bulletin board machine, which the crackers would use to contact each other and discuss techniques and progress!

By Thursday, 27 August, there was enough information about which machines had been compromised and how they had been broken into to allow an effective cleanup. In addition, the severity of the modifications the intruders were making, particularly on the bulletin board machine, made it imperative to stop the intrusions. The respective system managers, therefore, were contacted, arrangements made to shut down all machines, and a system cleanup scheduled for the next day.

On Friday, 28 August, the known affected machines were worked on, closing the security holes that had been used to break in, and all were brought back up on the network.

On Saturday, 29 August, an emergency call was received from one of the system managers, saying that the intruders had broken back into the cracker bulletin board machine. Concerned about the integrity of their research data, they asked for their machines to be physically disconnected from the rest of the network.

On Monday, 31 August, the logs of the new break-in were analyzed and it was determined that the crackers were much more sophisticated than originally believed and that many more local machines and user accounts had been compromised than initially realized. Several files were found containing hundreds of captured passwords, including ones on major (supposedly secure) servers. It appeared that there were actually two levels of crackers. The high level were the more sophisticated with a thorough knowledge of the technology; the low level were the "foot soldiers" who merely used the supplied cracking programs with little understanding of how they worked. Our initial response had been based on watching the latter, less capable crackers and was insufficient to handle the more sophisticated ones.

After much deliberation, it was decided that the only way to protect the computers on campus was to block certain key incoming network protocols, re-enabling them to local machines on a case by case basis, as each machine had been cleaned up and secured. The rationale was that if the crackers had access to even one unsecure local machine, it could be used as a base for further attacks, so it had to be assumed that all machines had been compromised, unless proven otherwise.

The recommendation to filter incoming traffic was presented to the Associate Provost for Computing on Monday afternoon and approved. The necessary equipment for the filter and monitor machines was bought or borrowed late that afternoon, and the design and coding of the filter proceeded through the night. Particular effort was made in the design to achieve the necessary security with the minimum of impact to local users. The filter was completed and installed by 5 PM Tuesday, 1 September.

At this point, the major task of analyzing all of the detailed logs and captured files was restarted. It was discovered that over 40MB of the cracker's tools had been captured, tools that they had FTP'ed onto some of the broken machines. These tools included Crack, network monitoring tools, all SunOS, Ultrix and Dynix source code (so they could replace any executable on the system), and cracking programs for virtually every CERT announced vulnerability. The logs showed that the crackers routinely placed back door and trojan login binaries on each broken system and used programs to set the timestamp and checksum of the replaced binaries to avoid detection.

On Thursday, 3 September, TAMUSC monitor logs showed an obviously automated attack by ftp that was sequentially probing every machine on campus. Here again it was decided to monitor this attack, as it was not clear what it could accomplish. This decision to observe, rather than immediately block, turned out to be very fortunate.

Shortly after midnight on Friday, 4 September, TAMUSC received a report from another site via the Computing Emergency Response Team (CERT) at Carnegie Mellon that the crackers had broken back into TAMU machines. The logs were immediately analyzed, and it was determined that the crackers had used ftp to install a program that allowed them to tunnel past the TAMU filter's blocks. In addition, even though they knew we were aware of their original intrusions, they continued their pattern of breaking in and replacing key system binaries.

At this point, the filter was completely redesigned to keep the crackers out, and the new version was installed by 5 AM Saturday. The new version changed the filter approach from "deny" based filtering (let everything in unless it is specifically denied) to "allow" based filtering (block everything unless it is specifically allowed). This new version, while providing much greater security, was unfortunately also more visible to valid users.

Since the new filter was installed, no successful intrusion attacks against TAMU machines have been observed, despite continued logging of probes and continued attempts. Recent efforts have centered in three areas: improving the ease of use and throughput of the filter, reducing the manpower requirements of the monitoring tools, and developing a program to help local system managers check their machines for proper security configuration.

Highlights of the Cracker Sessions

While all techniques used by the crackers aren't specified, the following section shows some of the more interesting things that were discovered. In all cases, references to specific machines have been changed; all of the spelling errors have been left in. The first fact is that the crackers seemed to have a compulsion to discuss their exploits with other crackers. While IRC seemed to be the preferred technique, many of the better crackers preferred less obvious methods, such as simply cat'ing directly to each other's ttys.

This snippet records an unnamed cracker on host1 talking with NMN (No Means No) concerning having run "ch", a brute force password cracking program on all 10 HP Snake machines in a Kent State lab. This conversation occurred on host1.tamu.edu using "cat >/dev/tty…".

NMN: "Well the people who run all 10 of the HP's will core when all thier logs show NMN.and especially if they find password crackers on them all.. me writing it is one thing ,but installing it is ANOTHER"

host1: "How would they find NMN on the hp.You lost me."

NMN: "Theyd see me Ftp to acct nmn on host2"

host1: "Possibly.Not liekly ..Unless they suspect something.Kent is the least concerened about security of any host i a have a ever been on.Oh maybe bsides .ai.mit.edu systems."

NMN: "Yeah ok"

host1: "Anyways.. dont forget to check up on the ch im running at host3.. its PID 16684 (ps -p 16684 will see if its still there) At host3, it should take 12.59712 days. *grin* BUT it sould crack it.. its running right now, it could crack it tonight, could crack it next week, who knows."

One extreme form of communication involved one cracker setting up a clandestine conversation bulletin board called LIMX (Local Immediate Message eXchange) on host4.tamu.edu, a machine that they had broken. LIMX was implemented as a passwordless back-door login by replacing the login executable. Here is the logout message from LIMX:

> "Connect to us again sometime, dont forget to spread the word about our system (host4.tamu.edu/128.194.xx.xx) and the account name (limx) to all your friends. If you dont have any friends, thats ok, tell your family members! Imagine the fun at the dinner table if Mom, Dad, sister and brother (and of course your dog fido) all had computer terminals and were connected online to the BACKDOOR LIMX, all chatting about the latest gossip, sports reports, fashon tips, and the shocking crimes being committed (which of course aren't related to our wonderful hypocrisy 'democracy'), and of course the wonderful meal moms been slaving all day over a hot kitchen microwave making, which none of you can comprehend. So spread the word! And connect to us again! Anonymous Cyberpunk Number One Oh yeah, and thanks to NoMeansNo for making this wonderful program! Sure beats IRC!"

3. Package Overview

Response Overview

Response to the intrusion incidents has three major thrusts: filtering, monitoring, and cleaning. The first line of defense is the bridging filter package *drawbridge*, which is used to filter all packets to or from the internet. *Drawbridge* allows internet access to be controlled on a machine by machine and port by port basis on a full T1 bandwidth basis. Using the filtering built into the TAMU WAN router (cisco) was initially considered, but it was determined that our requirements, particularly in the need for supporting potentially different filtering to each of the roughly 5,000 IP machines in the TAMU class B network, were too complex for the router. In addition, something was needed that could handle full T1 bandwidth, was itself very secure, and could be implemented rapidly. While other firewall configurations are known to be stronger, *drawbridge* provides a level of compromise between security and availability more acceptable to the university environment and provides much needed flexibility and throughput for the TAMU large scale network.

Realizing that *drawbridge* was a compromise between convenience and security, a set of monitoring tools was developed to look for intrusions that might be attempting to circumvent the filter. These tools continuously monitor the internet link, checking for unusual connections, patterns of connections, and for a wide range of specific intrusion signatures.

The third major thrust has been the development of the *tiger scripts*, an automated tool for checking a given machine for signs of intrusion and for network security related configuration items.

Figure 1 shows an overview of the filter and monitor implementation. In traditional secure gateways, a filter and secure bastion host are used and all traffic to or from internet is forced through them. This typically means that users need proxy clients for external access, such as for telnet and ftp, so that they all do not have to log on to the bastion host for external access. At TAMU, the filter allows arbitrary protocol filtering on a host by host basis, so that each department can set up its own authorized hosts with their own service configurations (subject to the campus wide minimum standards). This provides a reasonable level of both security and flexibility for educational and research requirements. For a UNIX host to be enabled at all beyond the default incoming permissions for mail, it must pass the *tiger scripts*, as described later. The monitor node is placed outside the filter so that it can record connection attempts which are blocked by the filter. This placement has been crucial to recognizing intrusion attacks, but does place the monitor itself at risk. To minimize this risk, both the filter and monitor are placed in a controlled access machine room and the monitor is configured for secure network access. The filter is similarly programmed only to respond to secure filter update requests, which are not routeable.

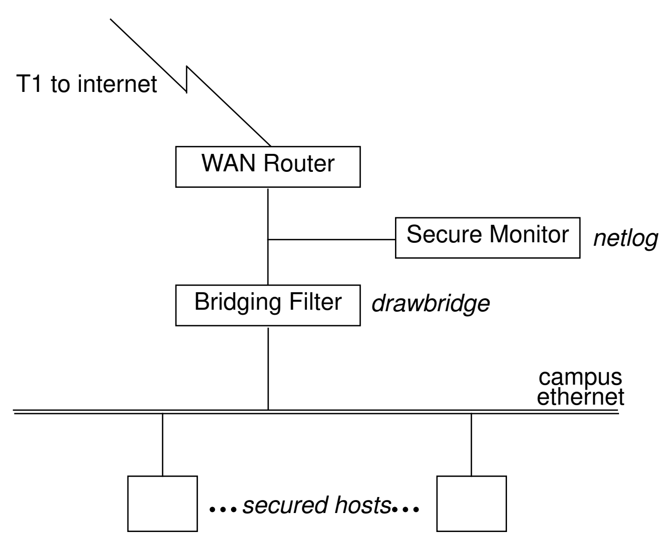

Figure 1. Overview of Components.

Filter (drawbridge)

Chapman [1] presented an interesting analysis of the limitations of current filter implementations at the Third UNIX Security Symposium. The drawbridge program, along with its support filter specification language and compiler, address some of his critical recommendations with respect to both functionality and ease of specification.

The filter approach chosen was based on a PC with two SMC 8013 (AKA Western Digital) ethernet cards. The first software implementation was based on pcbridge by Vance Morrison. This initial version was soon rewritten from scratch in C (compiled with turbo C++) to make the addition of needed features somewhat easier. The current filter design provides "allow" based filtering per host with separate incoming and outgoing permissions.

For both performance and configuration management, the filter tables are created on a support workstation, based on a powerful filter configuration language, and then securely transferred to the filter machine, either at boot time or dynamically during operation. The support machine does all the hard work of parsing the configuration file, looking up addresses, and building the tables, so that the filter itself need only perform simple O(1) table lookups at run time. Updating the tables dynamically is made secure with a Data Encryption Standard (DES) authentication.

The current default configuration allows any outgoing connection, but basically allows in only smtp (mail). Several campus and departmental servers have been checked and set up as hosts that are able to receive incoming telnet, ftp, nntp, and gopher requests.

Monitor

The goal of monitoring is to record security related network events by which intrusion attempts can be detected and tracked, particularly in those services allowed through the filter. This is a very difficult problem in general. The communication data rates make this problem somewhat like trying to take a sip of water from a fire hose; TAMU has some 30 terabytes of internal data transfer per day, and its internet connection is on the order of 4 gigabytes per day, with an average of 100,000 individual connections during that period. Clearly, monitoring needs to be both very selective and flexible, and automated tools are needed for reviewing even these resultant logs. Another problem is that of monitor placement. It is important that monitors be placed so that critical segments can be observed and so that the monitors themselves are secure.

Our solution includes the programs *tcplogger*, *udplogger*, *etherscan*, *nstat*, and some associated support programs. The tcp and udp loggers basically log a one line summary for all connection attempts. The associated analysis programs report on suspicious connections or patterns of connections. In addition, these logs have been very useful in analyzing details of security events after the fact. The *etherscan* program goes much further, actually scanning all packets and their contents, looking for a specific set of intrusion signatures, such as root login attempts from off campus. The *nstat* program collects statistics on all traffic to the filter and is useful both for capacity planning and for detecting unusual activity patterns. *Nstat* detected a clandestine FSP server on campus that was providing a repository of pirated commercial software, simply by noting a large transfer rate on a specific UDP port.

Machine Cleanup (the tiger scripts)

The phrase 'Tiger Scripts' comes from the concept of a 'tiger team.' A tiger team is a group which locates problems in a security system and demonstrates this problem by using it to circumvent the security system. By doing this, it is hoped that any weakness will located and corrected. The 'Tiger Scripts' perform the first part of this task in the UNIX environment. They search through a UNIX system and report any elements of it which may represent a security risk.

After a series of intrusions were discovered at Texas A&M University, it became apparent that a large and unknown number of machines attached to the campus network had been compromised. A filtering bridge was installed between the campus network and the Internet in order to protect the machines at the campus. It was still necessary though to clean up the machines that were involved. Most (if not all) of these machines were UNIX systems, but there were only a few people available at the university with the knowledge to locate and correct the problems on these machines. The 'Tiger Scripts' were developed to search out and report these types of problems. The scripts are also used as a means of verifying the security of machines to which access is allowed through the filtering bridge. Because of continuing development, the scripts have grown beyond just this internal use.

There were several goals for the ‘Tiger Scripts.'

- Ease of use
- Robustness
- Portability
- Functionality

It was essential that the scripts be easy to use, as they were to be used by persons who possibly had little UNIX systems management background. Towards this end, no user configuration of the scripts are required. Once unpacked, all that is necessary is to run the script ‘tiger.' This generates a report which contains any possible problems found. All messages are tagged with severity levels. These severity levels are used to indicate whether the system would be cleared through the filtering bridge. At the university, the manager of a system simply provides a copy of the report when requesting access.

It was also essential that these scripts be robust. Since they would be run on many different systems, the scripts had to be written in order to handle the nuances of the many administrators. This is of most concern when parsing system configuration files, as this is often where security problems manifest themselves.

In addition to these goals, portability had to be considered as well. The flavors of UNIX running on machines at Texas A&M is diverse. The initial release of the scripts (October 1992) was targeted toward machines running SunOS 4.1.x, as these machines were the primary target during the intrusion. The second release (April 1993) incorporated support for SunOS 5.x and NeXTOS 3.0. Future releases will include support for other UNIX derivatives such as AIX 3.x, HP-UX, IRIX, and UNICOS. The scripts are designed, however, to perform a "best of their ability" attempt even when specific support is not provided for a system. The only primary requirement is that the systems Bourne shell support the defining of shell functions.

The scripts accomplish these goals, while at the same time providing the following functionality. System configuration files are checked for problems, system binaries are checked for alterations, or known security problems, and known signs of an intrusion are checked. There is also support for the extension of the checks as new problems are reported.

For ease of use, the *tiger scripts* label all outputs with an error classification:

ALERT	A positive sign of intrusion was detected.
FAIL	The problem that was found was extremely serious.
WARN	The problem that was found may be serious, but will require human inspection.
INFO	A possible problem was found, or a change in configuration is being suggested.
ERROR	A test was not able to be performed for some reason.

As an aid to ease of use, an explanation facility is provided with the *tiger scripts*. Explanations can be requested for specific messages, or an explanation report can be generated for the security report. The explanation report can automatically be inserted into the security report, with explanations following each of the security messages. The explanations describe the situation, why it is a problem, and how to correct it. Figure 2 shows an excerpt from a security report with explanations inserted.

The checking performed covers a wide range of items, including items identified in CERT announcements and items observed in the recent intrusions. The scripts use cryptographic checksum programs to check for both modified system binaries (possible trap doors/ trojans), as well as for the presence of critical security related patches.

At the present time, the *tiger scripts* have been configured for SunOS 4.1.x, SunOS 5.x, Nextstep 3.0, AIX 3.2, HP-UX, IRIX 4.0, and UNICOS 7.0 releases. The programs are largely table driven for ease of porting, and ports to other platforms are being worked on.

```
--WARN-- [acc012w] Login ID sundiag has uid == 0.

The listed login ID has a user ID of zero (0) and is not the 'root' account.
This should be checked to see if it is legitimate.  In any case, having login
ID's with a user ID of zero tends to lead to security problems, and should be
avoided (except for 'root')

--WARN-- [acc004w] Login ID csehlhp is disabled, but has a .rhosts file

The listed login ID is disabled in some manner ('*' in passwd field, etc), but
a non-zero length .rhosts file.  This can allow the login ID to continue to be
used.  Unless this has been specifically set up to provide some service, it
should be removed.

 --INFO-- [acc002i] Login ID vu18368 is disabled, and has a shell of /bin/login.

The listed login ID is disabled, but has a potentially valid shell. These can
usually be safely ignored, but should be checked.
```

Figure 2. Sample *tiger scripts* security report with explanations inserted.

Policies

The policies and procedures need to provide both security and flexibility. The resultant decision was to filter incoming traffic other than mail to all machines and then allow case by case requests for authorized hosts status, based on successful demonstration of basic security configuration with the *tiger scripts*. Special requests for allowing incoming requests to special servers that are not easily checked, such as for embedded robot controllers, have been made. In these cases, the connections have been allowed, but special monitors have been implemented on these services.

Long term policy questions that remain unanswered include how to handle updates in response to critical CERT announcements and how to handle OS updates. Obviously, some way to coordinate both periodic and quick response host reviews is needed. This filter configuration language does support machine classes, so it would be possible to do something such as “disable ftp to all SunOS 4.1.1 machines” in response to a CERT announcement of a respective problem, but it would be nice to have a mechanism to communicate such announcements to the respective managers *before* cutting off access. The problem on a large campus is maintaining a contact list for a large number of machines, given the high rate of turnover in student managers. In addition, the information in the filter configuration file may rapidly become outdated, as managers update their machines’ hardware and software. The current plan is to require periodic (annual) security checks with the current *tiger scripts*, enforced with the possible loss of IP authorization. In the case of aperiodic security events or announcements, an attempt will be made to evaluate the time criticality of responding and require appropriate event specific checking. As the *tiger scripts* are easy to run, it is anticipated that this requirement will not be a significant burden to system managers.

A recent case in point was the announced security problem with the wuarchive anonymous ftp code. In this case, it was known exactly which machines had ftp authorized, and the respective managers were contacted immediately. The managers updated their software so rapidly that it was not necessary to block access, and the limited number of authorized machines avoided the need for an immediate *tiger* update.

4. Filter (drawbridge)

Design

Drawbridge is different from any of the current standard firewall configurations. Using the categorization of firewalls developed by Ranum [2], *drawbridge* compares best to a filtering router firewall configuration as shown in figure 3. In a filtering router firewall, a router which has packet filtering support is used to filter packets to and from hosts on the “inside” of the router. This is used to establish a policy where hosts are provided more or less access depending on the decisions of the network managers.

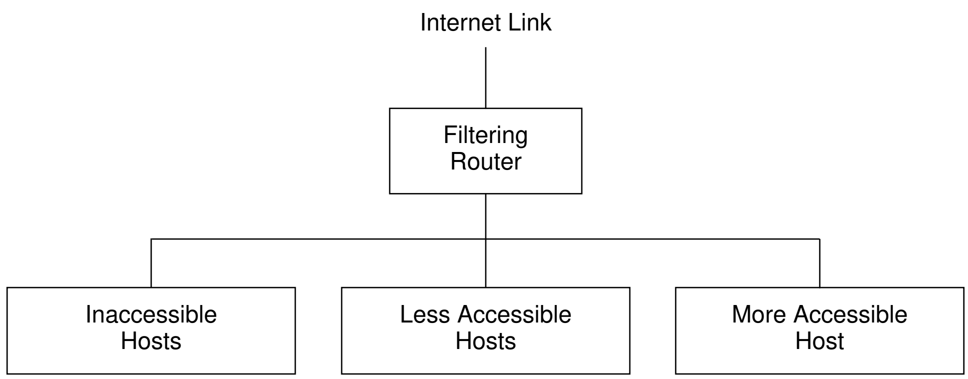

Figure 3. Typical filtering router firewall.

In an university environment, access typically would be based on the department the machine is located in, who manages/uses the machine, and an initial security audit, which is hopefully repeated on a regular basis. At TAMU, security audits are performed using the *tiger scripts* package, which was developed for this purpose. Any hosts that are never “registered” for access through the filter would receive some type of default access that would be defined by the network managers.

While this type of firewall is theoretically weak in comparison to other firewall methods, in practical use it does provide a useful increase in security. The points of attack have been greatly reduced and casual intruders are quickly discouraged.

A typical *drawbridge* firewall configuration is related to a filtering router firewall as shown in figure 4. The difference is that instead of using a filtering router as the firewall, the filtering function is moved from the router into *drawbridge* which acts as a bridging filter. Note, however, that figure 3 describes just a typical setup; a router is not a necessary component of a *drawbridge* configuration.

Comparison to Other Filtering Methods

Chapman [1] is an excellent source of information about packet filtering issues. He discusses the concepts behind packet filtering and some of the problems associated with it. He also discusses the problems with current implementations of packet filtering found in some current routing products.

Some of these problems include:

- Complex configuration language
- Difficult verification
- Lack of filtering on key parameters, such as source port or direction

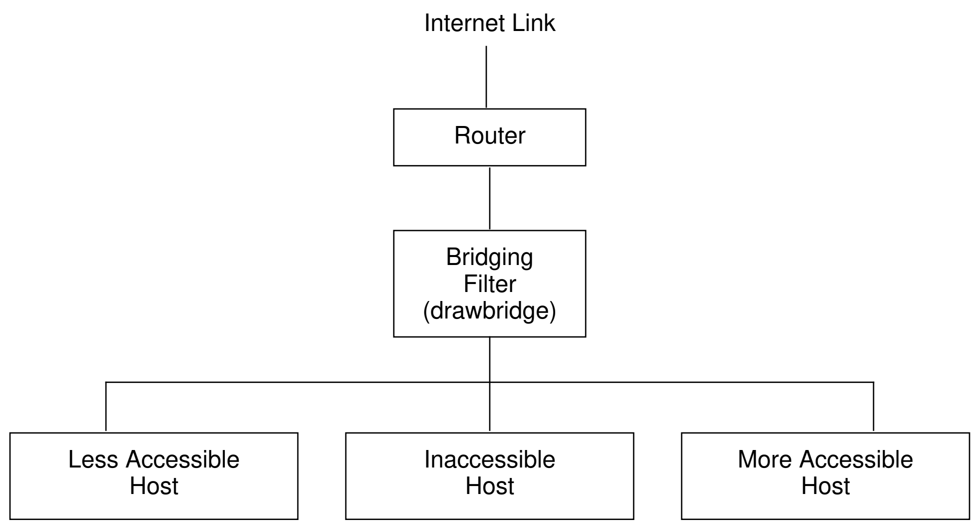

Figure 4. *Drawbridge* firewall.

Rather than repeat that material, it will be assumed that the reader is familiar with packet filtering and a discussion of how *drawbridge* tries to address some of these problems mentioned by Chapman [1] will be presented.

One of the first problems with current packet filtering implementations is that they are difficult to configure. They use a simple syntax that is designed for efficient implementation, not for effective configuration by an administrator. On a university campus, there is a need for many different filtering configurations to satisfy the diverse needs of the many users. Also, while the needs of administrators are usually defined in terms of connections, filters usually are defined in terms of packets only; the semantics of connections must be tediously mapped on to them.

These issues are addressed in *drawbridge* through the use of compiled tables. One table is defined for each (entire) IP network with each host address in that network being a single entry in the table. This allows a powerful source language to be designed that administrators can easily use and that is flexible enough to define complex sets of filters. In addition, *drawbridge*, under TCP, is not restricted to filtering in terms of packets but also filters in terms of connections. This makes configuration easier for the administrator and *drawbridge* more efficient.

This table design allows arbitrarily complex filters to be defined with little penalty. In conventional filtering routers, as filters are added, the performance begins to quickly drop due to how they implement the filtering rules. In *drawbridge*, arbitrary numbers of complex filters can be set up and the performance remains almost constant since simple look ups are performed and only connection establishment packets are filtered for TCP.

A second problem with most filtering implementations is that testing filter configurations is difficult. *Drawbridge* remedies this by allowing the administrator to check the results of a compiled configuration file to see if the correct filtering rules have been applied. Since *drawbridge* is less algorithmic than current filtering implementations, it is sufficient to investigate the compiler output. The administrator can look at the class that a host has been assigned and at the filtering lists defined for each subtable in that class.

A last problem that *drawbridge* addresses is the need for support for source port filtering. *Drawbridge* specifically defines an entire subtable to support TCP source port filtering (UDP source port is not currently supported). Since source port filtering does allow the possibility of tunneling, *draw-*

bridge does add the restriction that the destination port must be greater than 900; 900 was chosen due to certain FTP implementations that happen to use FTP data ports beginning at around TCP port 900 rather than following the BSD convention of starting at 1,024.

Physical Structure

Drawbridge is physically structured as shown in figure 5. The PC running *filter* is placed between the external network (Internet link) and the internal network (campus) that will be protected. Optionally, a Sun workstation can be used to communicate with and manage *filter* on the PC. *Filter* acts as a filtering bridge between the external and internal networks. *Filter* performs bridging but does not conform to bridging standards, e.g., it has no support for Spanning Tree Protocol.

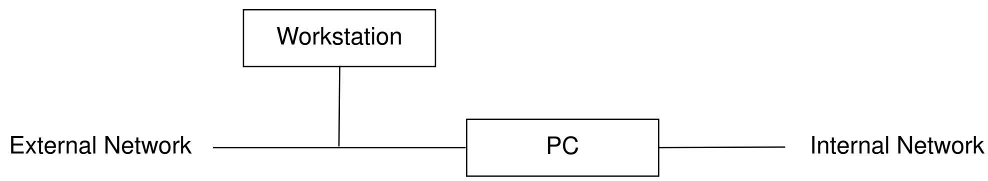

Figure 5. Ideal *drawbridge* Configuration.

In the ideal configuration, the workstation would be placed outside of *filter* so that monitoring of connection attempts from the external network can be performed. This is a good way to look for attacks and probes that are attempted against your internal network (and are hopefully blocked by *filter*). Since this also restricts the workstation's access to the internal network, a workstation will have to be committed specifically for this purpose. If a spare workstation isn't available, a machine on the internal network can be used to perform the management.

As mentioned above, *filter* is a table based filtering bridge. This approach was taken to improve the performance of filtering. The tables are generated by the following process (see figure 6). First, a source file containing filtering specifications in a special language is generated and maintained by an administrator. This file is then passed through *fc*, which generates the tables used by *filter*. These tables can be loaded via *fm* or by floppy disk.

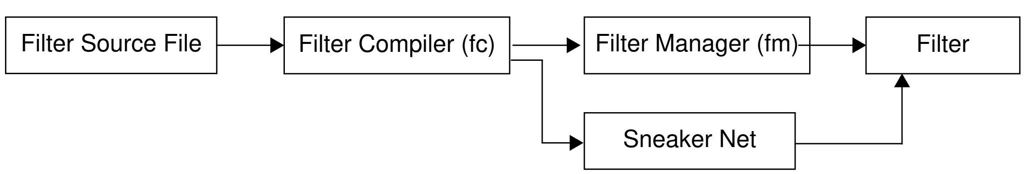

Figure 6. *Drawbridge* Compilation and Loading Process.

Filter Compiler Language

The language used by the compiler contains constructs for creating the various tables used by the filter. Constructs exist for specifying the network access on a per host basis, on a network or on a subnetwork basis. Groups of services can be created. These groups can be used in cases of related services or to group related machines. Access to particular external sites can also be granted, and access from certain sites can be denied. These constructs are shown in figure 7.

```
host (<hostname>|<ip_address>) <list of service_entries>
network (<ip_address>|<ip_address> - <ip_address>) <netmask> <list of service_entries>
define <group_name> <list of service_entries>
allow <ip_address> <netmask> <list of service_entries>
reject <ip_address> <netmask>
```

Figure 7. Constructs contained in the *filter* compiler language.

Hosts and networks can be granted network access using service specifications or group names. As hosts and networks are processed, the classes used by the filter are created. Hosts with equivalent network access (real access, not syntactic) will belong to the same class.

A group is a list of comma separated service specifications or other previously defined groups. Groups can be used to relate services or to categorize machines, allowing quick global changes to a category of machines. The special group "default" specifies the default access for any machine that does not match any of the networks loaded into *filter*.

Constructs also exist for building the allow and reject tables used by the filter. The allow table allows internal machines access to a restricted external service. The reject table is used to block all incoming packets from a host or network.

The basic element of the language is a service specification. The service specification contains four pieces of information: the service, protocol, source or destination, and traffic direction. The service can be either an entry from /etc/services or a numeric port. Service ranges can also be used. The protocol specifies the protocol the service uses. The source or destination indicates whether the filter should use the source port or the destination port. Finally, the traffic direction indicates whether this is for outbound packets, inbound packets, or both. The grammar defining a service specification is shown in figure 8.

An example configuration file is shown in figure 9.

```
<service_entry>  ::= <  (src=|dst=) <service_desc> (in|out|inout) > |
                     <! (src=|dst=) <service_desc> (in|out|inout) > |
                     <group_name>
<service_desc>   ::= <service> | <service_range>
<service>        ::= <port_number>                |
                     <port_number>  / <protocol>  |
                     <service_name>               |
                     <service_name> / <protocol>
<service_range>  ::= <port_number> - <port_number>               |
                     <port_number> - <port_number> / <protocol>
<protocol>       ::= <protocol_number> | <protocol_name>
```

Figure 8. Service entry grammar.

```
# Defaults for any machine not listed in this file.
define default  <1-65535/udp in>, <!tftp/udp in>, <!sunrpc/udp in>,
                   <!2049/udp in>, <1-65535 out>, <src=ftp-data in>,
                   <smtp in>, <auth in>, <gopher in>;

# Admin requested no access in/out for this subnet
network 123.45.58.0 255.255.255.0 <!1-65535 in-out>;

# NNTP host and CSO phonebook server
host mailnews.tamu.edu          default,
                                <nntp in>, <time in>,
                                <csnet-ns in>, <domain in>,
                                <finger in>;

# Machine (PC) in library which uses tftp to do document transfers
host sender.tamu.edu            <1-65535/udp in>;

# Has to have X
host arrow.tamu.edu             default, <ftp in>, <6000 in>;


# No TCP access in/out
host bee.tamu.edu               <!1-65535 in-out>;
```

Figure 9. Example configuration file.

How filter Works

fc generates four different kinds of tables (see figure 10):

- Multiple "network" tables, with one entry per host address,
- A "class" table with one permission list per distinct service class,
- A global "allow" table, and
- A global "reject" table.

The network tables have an entry for each host in the network. The host portion of an address (ignoring any subnetting) determines the index into the table. The value in the table defines the “class” that will be applied to a host when a packet is to be filtered. Only class B and C networks are currently supported as *filter* does not have the capability to use any memory above 1 MB.

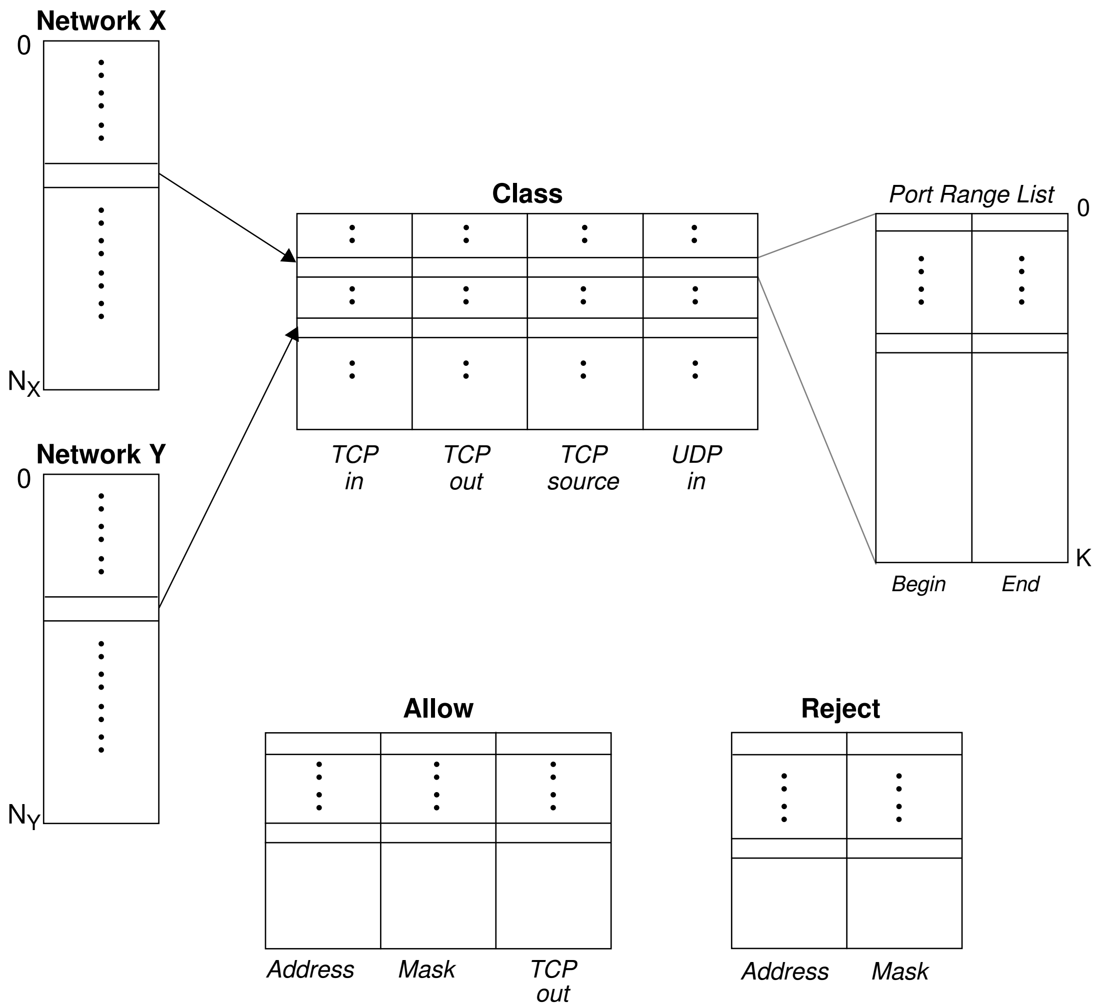

Figure 10. The four tables generated by *fc*.

Note that the network and class tables are defined in terms of a host on the internal network. No filtering is done based on the address of a host outside of the filter except on a global basis for reject and allow. It is assumed that an inside host will control which outside hosts are allowed to access its services, e.g., using TCPWrapper. *Filter* only controls which internal host's services are open, not which external hosts may access an internal host's services.

The host's class is used as an index into the class table. This second table is composed of four sub-tables: TCP in, TCP out, TCP source and UDP in. The subtables are composed of lists that contain port number ranges. A class specifies a list out of each subtable that defines a host's filtering. It is important to note that the TCP filtering only occurs when ACKless SYNs (connection initiation) are detected in a TCP header. All other packets of a TCP session are not filtered. Also, all UDP packets are filtered on an incoming basis only.

The last two tables are the allow and reject tables. The allow table globally allows packets out from any machine on the inside of the filter to the list of addresses in the allow table using the supplied list of port number ranges. The reject table globally rejects packets coming in from any machine on the outside of the filter with an address corresponding to an address in the reject table.

Figures 11 and 12 are flowcharts describing how *filter* uses these tables to check connections for incoming and outgoing requests, respectively.

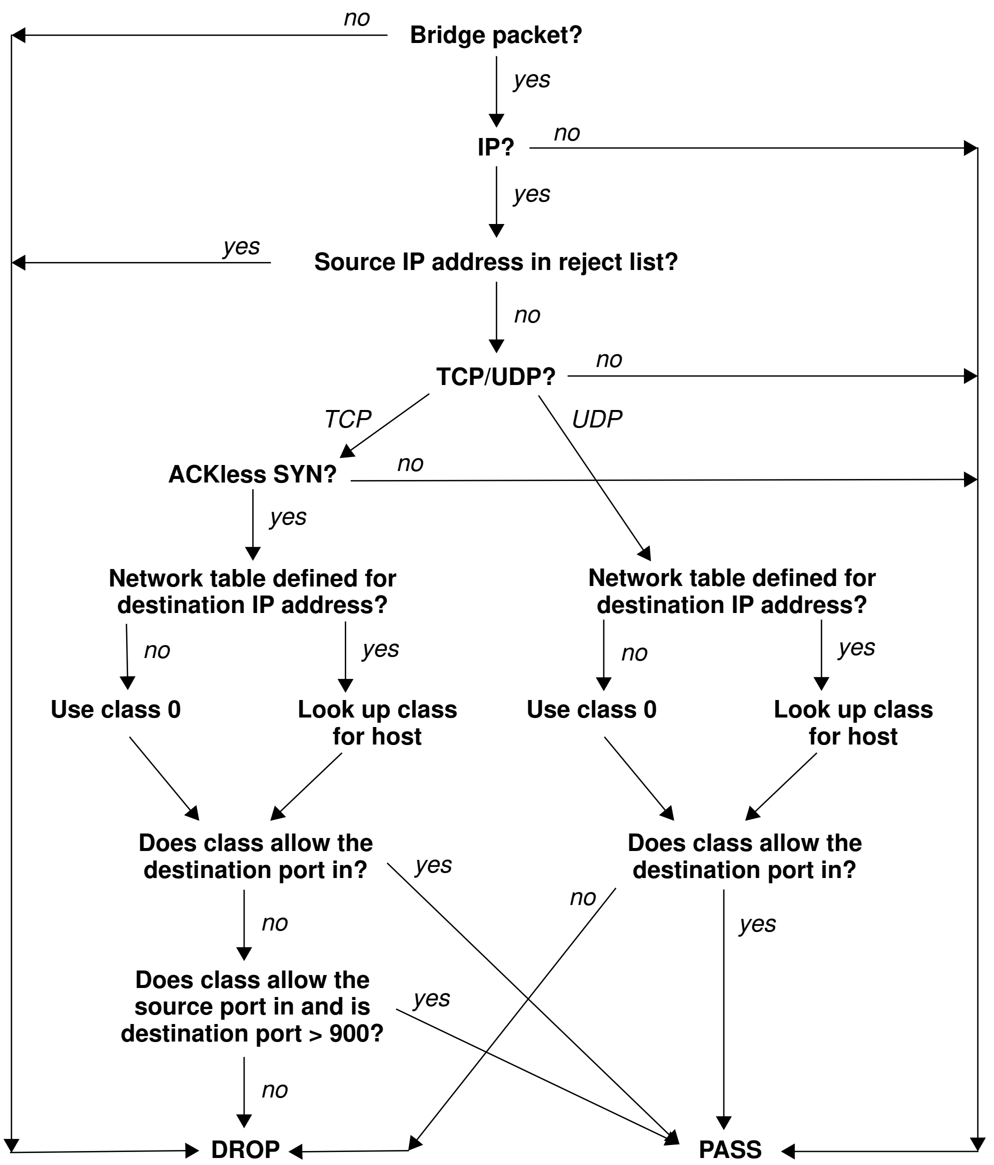

Figure 11. Incoming Packet Filtering Algorithm.

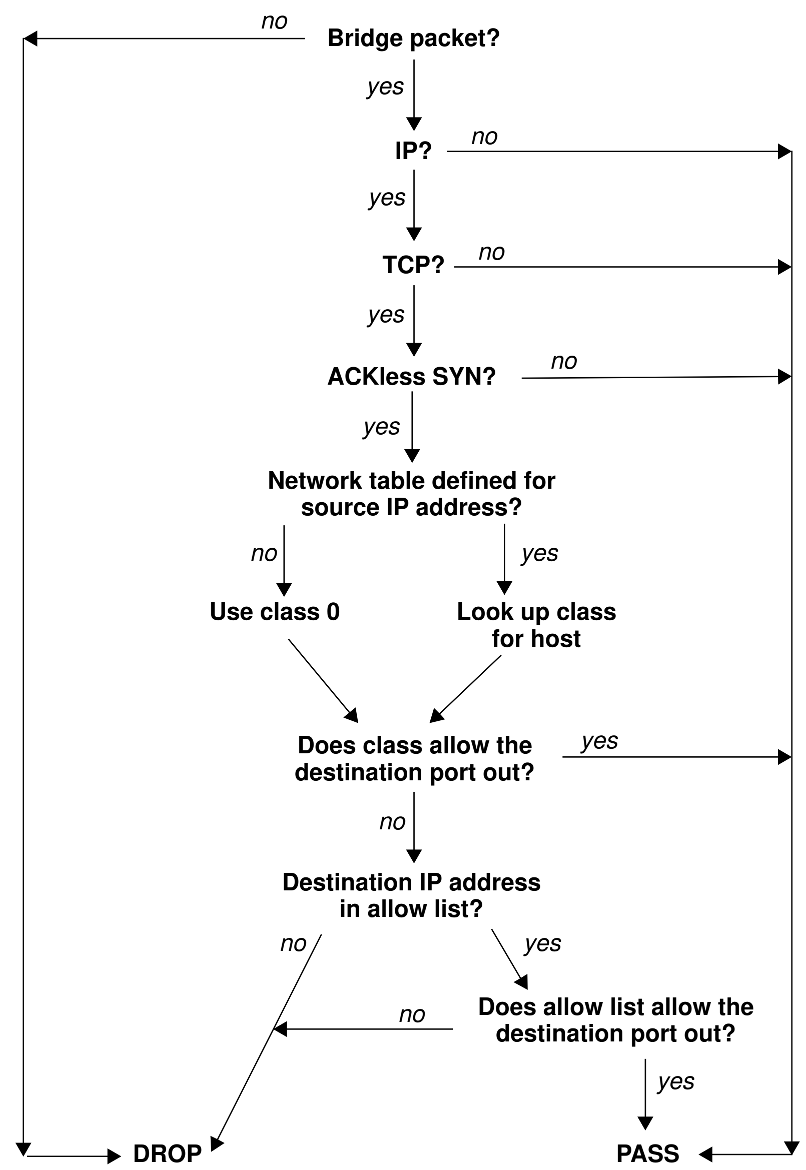

Figure 12. Outgoing Packet Filtering Algorithm.

Implementation

Filter started out as a simple modification to PCBridge, a public domain program written in assembly. It is now over 3,000 lines of C code. However, the efficiency of PCBridge has been retained and even improved upon. *Filter* double buffers outgoing packets and has code to perform adaptive bridging using a hash table.

Figure 13 shows the performance of *filter* under different loads. The input packet rates were generated and monitored with a combination of two workstations and an ethernet analyzer with the packets generated on one side of the filter and measured on the other. These packets were built such that all of them would pass through the filter (not be bridged) and would not experience any higher level protocol backoffs. A set of fixed sizes and a sample distribution of sizes were chosen with the sample distribution being based on the average distribution of packet size on our current campus ethernet backbone. While the filter peaked at about 2.5 Mb/s for the sample distribution, our campus backbone averages around 1.3 Mb/s. Our conclusions from these tests are that the filter is limited by the PC hardware and not the software. The configuration tested was a 33 MHz 486 with two 16 bit SMC ethernet cards on an 8 MHz ISA bus. By simply increasing the speed of the ISA bus to 11 MHz, we saw the performance threshold of the filter on 1514 byte packets jump by 1.0 Mb/s. We feel that 32 bit ethernet cards on an EISA bus would increase the performance greatly.

Memory usage depends greatly on the number of network tables that are loaded into *filter*. At TAMU we currently have one class B IP network that is filtered. *Filter* uses approximately 300 KB of memory in this configuration.

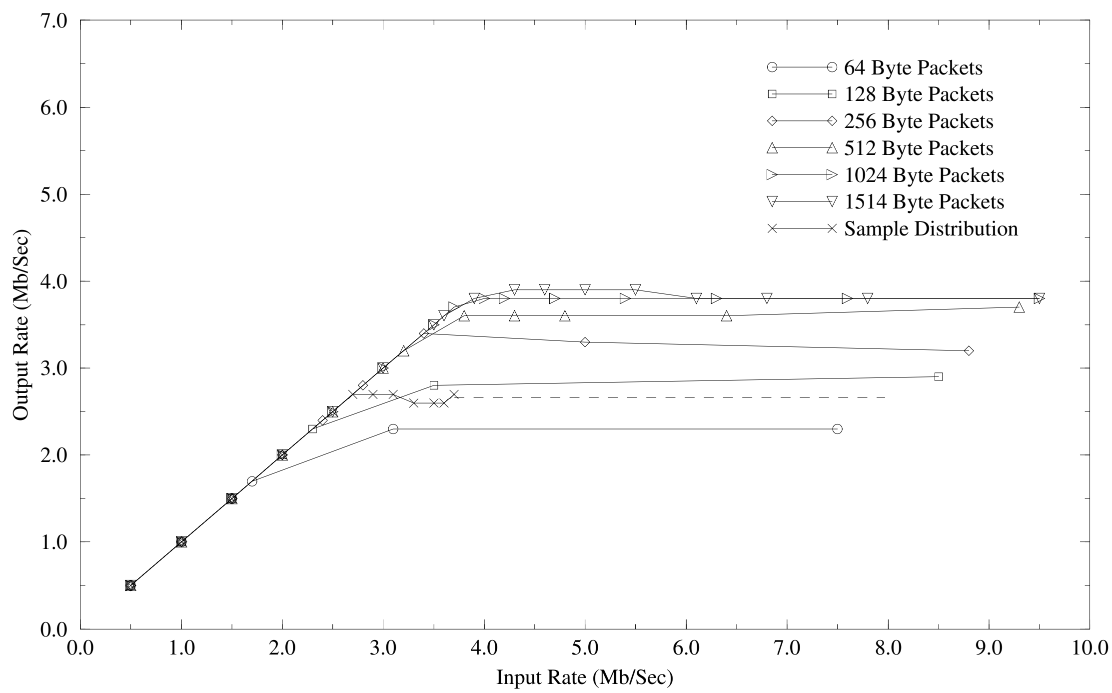

Figure 13. Performance of *filter* under different loads.

5. Monitor

Service Initiation Logging

The first tool, which is actually two tools (*tcplogger* and *udplogger*), records the initiation of a TCP session or UDP session. The start of a TCP session is indicated when the FLAGS field of the packet has the SYN flag set with no other flags set. A record is written to a log file, ASCII or binary, for each session. For a binary log file, the format of the record is:

```
struct sessinit {
      struct timeval start_time; /* Initiation time */
      unsigned long  ip_src;     /* IP source address */
      unsigned long  ip_dst;     /* IP destination address */
      unsigned long  tcp_seq;    /* TCP sequence number */
      unsigned short srcport;    /* Source port */
      unsigned short dstport;    /* Destination port */
};
```

The TCP sequence number is recorded so that duplicate packets can be removed in a post-processing phase. This simplifies the recording tool.

Detecting the initiation of a UDP session is not as simple. The UDP logging system uses a heuristic to determine the start of a session. Whenever a UDP packet is received, a table of “active” sessions is searched. If the packet belongs to an active session, that session’s active time stamp is updated. If no active session is found, then a new “active” session is created and the packet is logged, using the same record structure as used by the TCP logging system. Since UDP packets do not have sequence numbers, the sequence number field is set to zero. After a packet has not been received for a session for an (user specifiable) amount of time, that session is deleted.

Both tools use the SunOS 4.1 Network Interface Tap (nit) with packet filters (nitpf). The use of the packet filter significantly enhances the performance of these two tools.

Service Initiation Log Processing

The binary log files created by *tcplogger* and *udplogger* contain records of all the TCP and UDP sessions that have occurred. This is, of course, a large number. On a normal day, there are over 100,000 TCP sessions and around 50,000 UDP sessions occurring on the TAMU Internet link. A tool was developed for extracting only those records of interest. The tool, *extract*, uses an “awk”-like language for selecting records and printing them. Records can be selected based on source or destination port, host, and network, date, and time. Selectors can be grouped using the boolean “and” and “or” operators. An example *extract* script is shown in figure 14. *Extract* can generate ASCII or binary log files. The binary log file uses the same format as the TCP and UDP logging tools, thus allowing further processing to occur. Since there is no information in the binary log file to indicate whether it is a TCP or UDP entry, the two can not be mixed, and *extract* must be informed of the type of file it is processing.

In practice this restriction is not a problem. There is generally a lot of noise with UDP traffic (traceroutes, FSP, etc.). If mixed together with the TCP log data, a TCP connection might be lost in the noise. Separating them into two log files eliminates this problem. As we normally maintain two logs for this reason, tagging each record by type so that the logs can be combined, is not of significant benefit.

```
#!/usr/local/etc/extract -f
#
#  Print out interesting TCP events coming from the Internet
#
srcnet = 128.194.0.0 {next}  # Skip sessions originating from A&M
dstport = shell ||
  dstport = exec ||
  dstport = 6000 {print; next}
srchost = terminus.lcs.mit.edu ||
  srchost = nyx.cs.du.edu {print; next}
dstport = smtp || dstport = telnet || dstport = finger ||
  dstport = 113 || dstport = ftp || srcport = ftp-data {next}
dstport = nntp && dsthost = news.tamu.edu {next}    | |
dstport = 2000 && (
    dsthost = mud1.tamu.edu ||
    dsthost = mud2.tamu.edu) {next}
dstport > 1023 {next}
{print}  # Print anything that makes it to here
```

Figure 14. Example *extract* script.

Protocol Signature Analysis

While the TCP/UDP logging tools allow us to detect when someone is probing the campus machines for tftp, or some related activity, they don't tell us what happens when someone connects to a system via telnet or some other TCP/IP service. The *etherscan* tool provides this capability. *Etherscan* monitors certain protocols for unusual activities. These protocols are the ones normally allowed through the filtering bridge, i.e., telnet, ftp, smtp. The specifics of what is watched for will not be discussed here, as we do not want potential intruders to know exactly at what we are looking. One example though is attempts to login using system account names, e.g., “root.” Another feature of *etherscan* is the ability to detect and report FSP servers. FSP is a UDP based file transfer system which has found favor among those wishing to keep their activities hidden from network and system administrators. A sample listing of the output of *etherscan* is shown in figure 15. At TAMU, there are approximately 20 records recorded per hour. Most of these are due to people attempting to login as the user ‘guest' or ‘anonymous'.

As with *tcplogger* and *udplogger*, *etherscan* uses the SunOS 4.1 Network Interface Tap and the packet filtering mechanism for performing packet captures.

```
05/26/93 08:35:54.19 [smtp] some.host.edu.2992 HOSTAT.TAMU.EDU cmd help
05/26/93 08:36:00.08 [smtp] some.host.edu.2992 HOSTAT.TAMU.EDU cmd help data
05/26/93 08:36:04.19 [smtp] some.host.edu.2992 HOSTAT.TAMU.EDU cmd help mail
05/26/93 08:36:19.84 [smtp] some.host.edu.2992 HOSTAT.TAMU.EDU cmd help rcpt
05/27/93 11:16:49.44 [udp] AHOST.UDE.EDU.21 possible FSP server.
05/27/93 13:23:30.42 [ftp] 128.194.180.66.27935 out.there.edu attempted login
                           `USER'
05/27/93 13:23:31.87 [ftp] 128.194.180.66.27935 out.there.edu 530 User USER
                           access denied..
05/27/93 13:55:37.29 [telnet] 128.194.225.211.33633 over.there.edu attempted
                              login as `system'
```

Figure 15. Example output from *etherscan*.

Traffic Analyzer

The final tool, *nstat*, is used to locate changes in network traffic patterns. This tool was originally written in order to gather statistics on the usage of various protocols on the Internet link. By recording these levels on an hourly basis, changes in the usage of a protocol can indicate someone attempting to bypass system security. Using this, an unauthorized FSP server was discovered on the campus when the UDP port that was being used suddenly increased in utilization (this was before support had been added to *etherscan* for detecting FSP servers).

Nstat has a raw output in ASCII format, although it is not really intended for direct viewing. Two programs, *nsum* and *nload*, are provided to analyze the raw *nstat* output. *Nsum* is a PERL program that summarizes the utilizations statistics, giving an ASCII histogram of the top ten usages at the ethernet, IP, TCP, and UPD levels. *Nsum* has support for analyzing certain time periods, such as morning, afternoon, etc., and for selecting weekday versus weekend times. The second analysis program, *nload*, is a simple awk program which produces data suitable for graphing with a tool such as xvgr. A parameter file for xvgr is included. Figure 16 shows a section of the raw *nstat* output. Figure 17 shows a histogram produced by *nsum*.

Ethics

Many may question the ethics and legality of such monitoring. We feel that our current system is not a privacy intrusion. The TCPLOGGER and UDPLOGGER are simply the network equivalent of process accounting, as they log routine network events, but none of the associated user level data associated with the event. Etherscan similarly reports unusual network events, which is the network equivalent of logging failed login attempts.

```
#Start Wed May 19 18:48:01 1993
#Stop  Wed May 19 19:22:52 1993
#550641 packets, 78169154 bytes, 9178 802.3, 2 runt, 3320 missed.
e 600       # 175        b 15882
e 800       # 457927     b 67569221
e 806       # 235        b 14104
e 889       # 184        b 11452
i 1         # 819        b 63686
i 6         # 446119     b 66005034
i 9         # 87         b 104942
i 17        # 10642      b 1375485
t 20        # 40416      b 16027884
t 21        # 1263       b 96426
t 23        # 81314      b 6522485
t 25        # 18540      b 3092486
t 37        # 8          b 480
t 53        # 58         b 4104
t 70        # 11347      b 4266897
t 79        # 262        b 22230
t 113       # 26         b 1596
t 119       # 108098     b 16296687
u 42        # 6          b 360
u 53        # 6634       b 628649
u 111       # 4          b 536
u 123       # 3622       b 325980
u 125       # 67         b 7102
u 127       # 67         b 7102
u 161       # 156        b 13950
u 213       # 24         b 1776
u 513       # 92         b 9960
u 514       # 3          b 348
u 517       # 6          b 552
u 518       # 137        b 17142
u 520       # 4864       b 954544
```

Figure 16. Raw *nstat* output.

```
Utilization: 68.44%

ETH IP        (50%/57%):#################################################
ETH DECIVDNA  ( 3%/ 4%):###
ETH DECLAVC   ( 3%/ 4%):###
ETH oldIPX    ( 2%/ 3%):##
ETH DECLAT    ( 1%/ 1%):#
ETH Banyan    ( 0%/ 0%):
ETH DECRCONS  ( 0%/ 0%):
ETH DECLBM    ( 0%/ 0%):

IP  TCP      (56%/47%):########################################################
IP  UDP       ( 3%/ 3%):###
IP  ICMP      ( 0%/ 0%):

TCP ftp-data (18%/14%):##################
TCP nntp     (17%/13%):#################
TCP 2000     ( 7%/ 6%):#######
TCP telnet   ( 5%/ 4%):#####
TCP 175      ( 3%/ 2%):###
TCP smtp     ( 3%/ 2%):###
TCP 6667     ( 2%/ 1%):##
TCP 1023     ( 1%/ 1%):#

UDP route    (17%/ 1%):#################
UDP domain   (14%/ 1%):##############
UDP ntp      ( 6%/ 0%):######
UDP 2074     ( 3%/ 0%):###
UDP 1081     ( 2%/ 0%):##
UDP 1025     ( 1%/ 0%):#
```

Figure 17. Example *nsum* output.

6. Machine Cleanup (the tiger scripts)

Structure

The *tiger scripts* consist of one primary "driver" script named *tiger*, a set of scripts which check various components of the system, support scripts, and support files. The driver script, *tiger*, executes each of the component scripts. Its usage is:

```
tiger [-B tigerhomedir] [-d loggingdir] [-w scratchworkdir]
```

The configuration file, *tigerrc*, can be used to enable or disable the different component checks. This allows certain fast executing components to be executed frequently, while other, longer executing components can be executed less frequently.

The driver and component scripts make use of many support scripts and data files. These are used to make the scripts portable. The support scripts are used to set internal variables, such as the pathnames to the UNIX commands used by the scripts, and to convert system configuration files into the formats the main scripts can parse. For example, one of these scripts is used to generate '/etc/passwd'-like input from the various locations that this information is obtained on a particular system. They are also used to provide functionality that a particular system may be lacking. The data files contain information that is compared against information found on the system. For example, one of the files contains the permissions expected on system files and directories.

Component Scripts

The checks performed by the *tiger scripts* are broken out into several component scripts. Ordinarily, these are all executed by the driver script *tiger*, but they can all be executed directly.

Many of the scripts check the ownership and access permissions for files. There are two different types of checks. They will be distinguished in this paper by referring to them in the following manners.

A check of a ***pathname*** means that all components of the pathname are checked. If the support module ‘realpath' is available (i.e., it compiled), any symbolic links are handled as well. As an example, on SunOS 4.x systems, ‘/usr/spool' is a symbolic link to ‘/var/spool'. Therefore, the pathname ‘/usr/spool/cron' contains the components ‘/usr', ‘/var', ‘/var/spool', and ‘/var/spool/cron' which will be checked. All “incorrect" ownerships or access permissions along the pathname are reported in detail.

A check of a ***filename***, ***directory*** or ***file*** means that only the ownership and access permissions of the file referred to by the filename are checked.

The ownership and permissions requirements are user configurable. The default configuration requires that pathnames to system executables and files be owned by root and writable only by root.

check_accounts

The ‘check_accounts' script examines user accounts for the following:

- passwordless accounts
- checks home directory ownership and access permissions
- checks ownership and access permissions of configuration files
- disabled account with .rhosts file, cron entries or .forward file that executes a program.

```
--WARN-- [acc004w] Login ID nag is disabled, but has a .rhosts file
--WARN-- [acc006w] Login ID dave's home directory has group
         `apcis' write access.
```

check_aliases

The ‘check_aliases' script examines the system mail aliases file. It reports any program aliases, and also checks the pathnames to the executable for proper ownership and access permissions. It also verifies that pathnames to any included files have proper ownership and access permissions (the entries in the included files are also processed).

```
--INFO-- [ali005w] Alias `drawbridge-server' contains a program
         entry: |"/xpub/secure/mailserver"
```

check_anonftp

This script checks for an anonymous FTP setup, and if one is found checks the integrity of it. Ownership and permissions of critical files and directories are checked. It also reports on any writable directories that are found.

```
--WARN-- [ftp010w] ~ftp/xpub/ftp/bin is writable by 'ftp'.
```

check_cron

The ‘check_cron' script checks user ‘cron' files. It examines the ownership and permissions of the pathnames used by each cron entry. Executables without absolute pathnames are also reported. The script attempts to be smart and interpret complex cron entries.

```
--FAIL-- [cron003] cron entry for root uses `/var/netlog/bin/logit'
         which contains `/var/netlog' which is group `wheel' and
         world writable.
         /var/netlog/bin/logit stop
```

```
--WARN-- [cron002] cron entry for root uses '/home/accts/doug/
         tiger-2.1.1/tigercron' which contains '/home/accts/doug/
         shop' which is not owned by root (owned by doug).
         /home/accts/doug/tiger-2.1.1/tigercron -B
         /home/accts/doug/tiger-2.1.1 -l
         /var/spool/tiger -w /var/spool/tiger/work -b
         /var/spool/tiger/bin > /dev/null 2>&1
```

check_exports

This script checks the server side of an NFS machine. Filesystems exported with "root" access, are reported, although the script does understand diskless clients and is (usually) quiet about these. It also reports exporting of the root directory (/), unrestricted exports, etc.

```
--WARN-- [nfs011w] Unprotected directory /fs is exported with root
         access to host(s) HOSTX.TAMU.EDU.

--WARN-- [nfs011w] Unprotected directory /export/export1/sunos5 is
         exported with root access to host(s) thea, and
         poshost.

--INFO-- [nfs010i] Directory /export/export1/root/Xkernel.sun3x is
         exported with root access to host(s) xhost.
```

check_group

The 'check_group' script cross references all of the sources of 'group' information for consistency. Any conflicting group ids or group names are reported. The group files are also checked for correct format.

```
--WARN-- [grp004w] GID conflict for group 'system' between
         /etc/passwd (gid = 2) and NIS (gid = 128).

--WARN-- [grp005w] Groupname conflict for gid 8 between
         /etc/passwd (group staff) and NIS (group grads).
```

check_inetd

The 'check_inetd' script examines the configuration files '/etc/inetd.conf' and '/etc/services.' It looks for things such as mismatched entries (i.e., telnetd on a port other than 23). It also reports any services which have been added from the standard distribution. The pathnames of executables are checked for ownership and access permissions.

```
--WARN-- [inet005w] Service login is using /xpub/etc/in.rlogind
         instead of /usr/etc/in.rlogind.

--FAIL-- [inet009] inetd entry for login service uses '/xpub/etc/
         in.rlogind' which contains '/xpub/etc' which is group
         'daemon' and world writable.

--WARN-- [inet005w] Service shell is using /xpub/etc/in.rshd
         instead of /usr/etc/in.rshd.

--FAIL-- [inet009] inetd entry for shell service uses '/xpub/etc/
         in.rshd' which contains '/xpub/etc' which is group 'dae-
         mon' and world writable.
```

check_known

This script tests for known signs of an intruder. Directories known to be used by intruders are checked for unexpected files. These directories include ‘lost+found' directories, system mail-spool directories and window server directories. The setuid(2) system call is also checked to verify that it is working properly.

```
--ALERT-- [kis001a] /usr/uucp/.sys is a directory."

--ALERT-- [kis002a] /usr/spool/uucppublic/.hushlogin is not zero-
          length."

--WARN-- [kis008w] File ".stuff" in the mail spool, owned by ‘bin'.
```

check_netrc

The ‘check_netrc' script examines the .netrc files in user home directories. It reports if the permissions on the file are incorrect (i.e. world readable). It also reports any entries in the .netrc file which contain passwords which are not anonymous ftp entries.

```
--WARN-- [nrc002w] User imauser's .netrc file contains passwords
          for non-anonymous ftp accounts.

--FAIL-- [nrc001f] User urauser's .netrc file is readable and
          contains passwords for non-anonymous ftp accounts.
```

check_passwd

The ‘check_passwd' script performs the same functionality as the ‘check_group' script, except it works with "passwd" sources. Sources of password information are cross referenced for conflicts and also check for correct format.

```
--WARN-- [pass004w] UID conflict for login ID ‘smith' between
          /etc/passwd (uid = 125) and NIS (uid = 1388).

--WARN-- [pass005w] Username conflict for uid 6 between /etc/passwd
          (login ID sys) and NIS (login ID joeuser).
```

check_path

The ‘check_path' script checks the PATH variable as set in the various shell startup files. It checks for ‘.' (dot) in the PATH, correct ownership and access permissions of pathnames in the PATH and correct ownership and access permissions of executables in the PATH. By default, the script only checks the PATH for the root account. It can be configured to check all users, though the author is not comfortable with this.

```
--INFO-- [path008i] Setuid program /usr/bin/uux in root's PATH from
          .cshrc is not owned by root (owned by uucp).

--WARN-- [path002w] /usr/bin/ls in root's PATH from .profile is not
          owned by root (owned by bin).

--INFO-- [path006] The PATH set in root's .profile contains
          ‘/usr/bin' which contains ‘/usr' which is not owned by
          root (owned by bin).
```

check_printcap

The script ‘check_printcap' checks the pathnames to filters used by printers for proper ownership and access permissions. Since not all systems use the BSD print system, this script is invoked only for systems on which it is used (there currently isn't an equivalent System V print system checking script).

```
--WARN-- [pcap001w] Print control ‘if' for printer ‘lw2' uses
```

```
          '/usr/local/lib/topsif' which contains '/usr/local/lib'
          which is not owned by root (owned by bin).
```

check_perms

The ‘check_perms' script checks the ownership and permissions of system files. A database (specific to the platform) describes which files to check and what the expected permissions are. Files and directories such as ‘/', ‘/etc', ‘/etc/passwd', ‘/etc/group' and ‘/etc/aliases' are a few examples of the many files which are checked.

```
--WARN-- [perm019w] The owner of /etc should be root.
--WARN-- [perm019w] /etc should not have group write.
--WARN-- [perm003w] /sbin should not have group write.
--WARN-- [perm003w] /usr should not have group write.
```

check_rhosts

The ‘check_rhosts' script examines the .rhosts files in user home directories. It checks the permissions on the files and examines the files, reporting entries with a '+', attempted comment entries, entries that are only partially specified. This is a remote hostname with no remote username. There is no direct security problem here, but it can lead to one as people carry a .rhosts file from site to site.

In addition, it is possible to configure the script to report hostnames that do not match a set of regular expressions. A version of this script written in PERL is provided as well which attempts to verify that the remote user is the same person as the local user (using ‘finger').

For large sites, with user home directories distributed across multiple NFS servers, it is possible to configure this script to only check accounts with home directories on local filesytems. This can greatly increase performance. This can also be done for ‘check_netrc' and ‘check_accounts'.

The ‘/etc/hosts.equiv' file is also examined. All trusted hosts are reported. Any included netgroups are expanded and reported as well.

```
# Checking accounts from /etc/passwd...
--WARN-- [rcmd004w] User snag's .rhosts file has a '+' for user
          (host ourhost.tamu.edu).
# Checking accounts from NIS...
--WARN-- [rcmd006w] User imauser's .rhosts file has group 'tamug'
         and world read access.
--WARN-- [rcmd006w] User urauser's .rhosts file has group 'onet'
          read access.
```

check_signatures

This script is used to validate system binaries. It does this through the use of a data file which contains digital signatures, generated from distribution media, for important system binaries. There is currently support for two different signature methods: the XEROX Secure Hash Function signatures, commonly referred to as Snefru, and the RSA Data Security Inc., MD5 Message Digest Algorithm. An output block size of 8 is used (256 bits) for the SNEFRU hash. The script auto-detects the type of signature and generates the appropriate one for comparison.

The script reports any system binaries which do not match the stored signatures. It also reports system binaries which are out of date in regards to security patches (using signatures generated from the replacement binaries in the patches). This provides a means of determining quickly whether critical security patches have been installed on a system. A system is being planned which will allow up to date signature databases to be retrieved from a central site(s).

```
--WARN-- [sig004w] None of the following versions of /usr/bin/login
         (-rwsr-xr-x) matched the /usr/bin/login on this machine.
         >>>>>> Sun Patch ID 100630-01
         >>>>>> Sun Patch ID 100631-01
         >>>>>> Sun Patch ID 100633-01
         >>>>>> SunOS 4.1.2 (security patch is 100630)
--WARN-- [sig015w] /usr/bin/mail is from Sun Patch ID 100224-03
         (current is 100224-06)
--WARN-- [sig004w] None of the following versions of /usr/etc/
         rpc.yppasswdd (-rwxr-xr-x)matched the
         /usr/etc/rpc.yppasswdd on this machine.
         >>>>>> SunOS 4.1.2
```

find_files

The ‘find_files' script searches through the file systems and locates files that might present a security problem. These are

- setuid executables
- device files
- symbolic links to system files
- world writable directories
- files with an undefined owner or group
- files with unusual filenames

Setuid files are checked to see if they are scripts, or if they contain relative pathnames (an admittedly crude check). It also compares the list of setuid files against a list prepared from distribution media and reports any new setuid programs.

Any device files found in non-standard locations (i.e., not in /dev) are reported. The script understands diskless clients and will automatically ignore the device directories for these as well. Other directories can also be added to this list by setting a variable in the configuration file.

Any symbolic links to system files (such as /etc/passwd) are reported as well. While not directly a security problem, an ill-placed ‘chown -R' or ‘chmod -R' (or equivalent) could create one.

World writable directories are reported, as they are often used by intruders as a place to store log files. Unowned files are often an indication of a break-in. They also can lead to unexpected access for an account.

The unusual file names includes files with spaces in them, leading '.' or other characters. The list of file names is customizable via a variable which can be set in the configuration file.

check_embedded

The ‘check_embedded' script examines files and extracts any apparent pathnames which are embedded in the files. These pathnames are checked for “proper" ownership and access permissions. These files indicated by these pathnames are then in turn checked for embedded pathnames. This process is continued until no new pathnames are found, or a user specified search depth is reached.

The initial list of files searched comes from two sources. The first is a static list specific to a platform. For example, on SunOS 4.x systems, this would include the /etc/rc.* scripts. The second source is from the other 'tiger' scripts. When the scripts are run from 'tiger' or 'tigercron', they generate lists of files that should be checked. For example, ‘check_path' will request that any executable in root's path be checked, and ‘check_inetd' requests that all the servers defined in /etc/inetd.conf be checked.

```
--WARN-- [embed002w] Path '/usr/sbin/fsck' is not owned by root
          (owned by  bin).
          Embedded references in:
          /sbin/mountall->/etc/init.d/MOUNTFSYS
          /sbin/mountall->/etc/init.d/buildmnttab
          /sbin/mountall->/etc/init.d/nfs.client
```

```
--WARN-- [embed003w] Path '/usr/sbin/ypinit' contains '/usr/sbin'
         which is group 'bin' writable.
         Embedded references in:
         /usr/lib/netsvc/yp/ypbind->/usr/sbin/sysidnet->
         /etc/init.d/sysid.net
```

Miscellaneous checks

In addition to these standard checks, miscellaneous checks specific to a system are also performed. Items such as the use of the ‘securenets' file on SunOS NIS servers and the removal of the "_writers" property for printers on NeXTOS are examples of the checks performed.

```
--WARN-- [misc004w] The PROM monitor is not in secure mode.
```

```
--FAIL-- [misc003f] No /var/yp/securenets file.
```

crack_run

The script ‘crack_run' is used to perform password cracking. No password cracker is provided with the 'tiger' system. It is expected that a tool such as Alec Muffett's 'Crack' will be used. ‘crack_run' collects all password sources and runs the password cracking tool on them, and reports the results. Since this can take days to complete (or longer), 'tiger' by default does not wait for this to complete.

```
--WARN-- [crk001w] The following login id's have weak passwords:
          imauser urauser
```

Isn't this just COPS?

One common question is how does 'tiger' compare to the COPS package by Dan Farmer There is a lot of overlap between the two packages. Much of this is intentional. Dan Farmer allowed us to borrow ideas and code from his COPS package. We are giving Dan (and anyone else) the same ability in regards to 'tiger'.

There are advantages and disadvantages between the two packages. Note that most of this is subjective and hence will no doubt be biased.

We feel that 'tiger' is easier to use than COPS for a person who is not familiar with systems administration. As stated earlier, once unpacked, all that is necessary is to run 'tiger' and a report will be generated. The explain facility allows the administrator to get a better idea about the entries in the report. One area that 'tiger' is lacking in is user documentation.

Another area we feel 'tiger' is at an advantage is based on part of the design structure, in which no system files are referenced directly. This feature is what allows the auto-checking of multiple information sources (for example /etc/passwd, NIS, NetInfo, etc) without having to alter the scripts which perform the checks. This makes running the scripts easier for the administrator.

There are other areas that we feel 'tiger' is better than COPS. These include the use of digital signatures for checking for security patches, the complete checking of pathnames containing symbolic links, and the more thorough examination of system configuration files.

We recognize that COPS does have advantages over 'tiger'. The primary one is that COPS is a "proven product". It has been used by thousands for two to three years now. Also, though 'tiger' will attempt to run on platforms for which no configuration files exist, COPS is more likely to succeed in running. The use of newer shell features, primarily shell functions, prevents the use of 'tiger' on any UNIX system which has an older Bourne shell. There are also miscellaneous checks performed by COPS that have not been integrated into 'tiger'. COPS also includes the 'kuang' expert system checker. There is no equivalent functionality in 'tiger'.

7. Observations

We have been using this combination of filtering, monitoring, and checking almost a year, with very positive results. While our network monitoring tools continue to show incoming intrusion attempts (unfortunately along with some outgoing attempts), we have had no major incidents of the type we experienced last summer. The combination of approaches seems to have struck an appropriate balance between security and availability for our academic environment.

Our monitoring tools have produced some interesting statistics. During the last four weeks, for example, we have observed the following number of incoming security events:

e-mail forgery	6
knob-turning	48
TCP attacks/events (X11, DNS zones, rshell ...)	30
UDP attacks/events (TFTP, SNMP ...)	7

Of these 91 incidents, 49 (or 54%) originated from .edu sites. Edu sites account, however, for only 30% of all internet hosts (according to the July 1993 Internet Domain Survey). This means that a minority of hosts are accounting for a disproportionate majority of intrusion activity. One would hope that other university networking groups would be actively trying to reduce these incidents, yet of the requests for the etherscan tool, less than 25% have come from university sites.

8. Conclusions

A set of policies and tools for filtering, monitoring and checking has been developed in response to a significant series of intrusions from internet. Each of these three areas has proved critical: the filtering for its ability to protect machines from attack, the monitoring because it augments the filter and has yielded significant information about the intruders and their methods, and the checking tools for their ability to automate the task of checking and cleaning a large number of machines. With these tools and associated policies, we have achieved an appropriate balance between security and availability in an academic environment.

9. Availability

Drawbridge, the *tiger scripts*, and all monitoring tools other than *etherscan* are now available via anonymous ftp in sc.tamu.edu:pub/security/TAMU. Due to export restrictions, the DES routines used in *drawbridge* have been put in a separate tar file and are readable only by U.S.A. sites. Other sites should have no problem either running the filter without encryption or dropping in their own favorite encryption package.

The distribution of *etherscan* has been hotly debated within the TAMUSC group. One argument is that *etherscan* should be freely released, as the crackers already have equivalent knowledge and tools (they do) and restrictions would only hurt valid administrators. The counter argument is that free availability of the intrusion signatures would enable the crackers to design better intrusions and the availability of sources would provide novice crackers a significant help. Our resultant compromise will be to provide copies to Network Information Center registered site contacts, given an official request on respective letterhead. Requests should be sent to:

Dr. Dave Safford, Director
Supercomputer Center
Texas A&M University
MS 3363
College Station, TX 77843-3363

10.References

[1] D.B. Chapman. Network (In)Security through IP Packet Filtering, *Proceedings of the Third UNIX Security Symposium*, September 1992.
(available from ftp.greatcircle.com as pub/pkt_filtering.ps.Z)

[2] Ranum. "Thinking about Firewalls", available on the Internet

[3] Violino, Bob. "Are Your Networks Secure?" Information Week, April 12, 1993, page 30.

The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment

David R. Safford, Douglas Lee Schales, and David K. Hess
Supercomputer Center
Texas A&M University
College Station, TX 77843-3363

1. Abstract

Texas A&M University (TAMU) UNIX computers came under coordinated attack in August 1992 from an organized group of internet crackers. This package of security tools represents the results of over seven months of development and testing of the software currently being used to protect the estimated 12,000 networked devices at TAMU (of which roughly 5,000 are IP devices). This package includes three related sets of tools: “drawbridge,” a powerful bridging filter package; “tiger,” a set of easy to use yet thorough machine checking programs; and “netlog,” a set of intrusion detection network monitoring programs.

2. Introduction

A Brief History of the Incidents

On Tuesday, 25 August 1992, the Texas A&M University Supercomputer Center (TAMUSC) was notified by the Ohio Supercomputer Center that a specific Texas A&M University (TAMU) machine was being used to attack one of their computers over the internet. The local machine turned out to be a Sun workstation in a faculty member's office. Unfortunately, this faculty member was out of town for a week, so rather than trying to gain access to the machine through the department head, it was decided to monitor network connections to the workstation and, if necessary, disconnect the machine from the net electronically. This decision to monitor the machine's sessions rather than immediately securing it turned out to be very fortunate, as this monitoring provided a wealth of information about the intruders and their methods.

The initial monitoring tools were very simple, but as the significance of what was occurring became apparent, the tools were rapidly improved to the point that the intruder's entire session could be watched in real time, keystroke by keystroke. This monitoring led to the discovery that several outside intruders were involved and that many other local machines had been compromised. One local machine had even been set up as a cracker bulletin board machine, which the crackers would use to contact each other and discuss techniques and progress!

By Thursday, 27 August, there was enough information about which machines had been compromised and how they had been broken into to allow an effective cleanup. In addition, the severity of the modifications the intruders were making, particularly on the bulletin board machine, made it imperative to stop the intrusions. The respective system managers, therefore, were contacted, arrangements made to shut down all machines, and a system cleanup scheduled for the next day.

On Friday, 28 August, the known affected machines were worked on, closing the security holes that had been used to break in, and all were brought back up on the network.

On Saturday, 29 August, an emergency call was received from one of the system managers, saying that the intruders had broken back into the cracker bulletin board machine. Concerned about the integrity of their research data, they asked for their machines to be physically disconnected from the rest of the network.

On Monday, 31 August, the logs of the new break-in were analyzed and it was determined that the crackers were much more sophisticated than originally believed and that many more local machines and user accounts had been compromised than initially realized. Several files were found containing hundreds of captured passwords, including ones on major (supposedly secure) servers. It appeared that there were actually two levels of crackers. The high level were the more sophisticated with a thorough knowledge of the technology; the low level were the “foot soldiers” who merely used the supplied cracking programs with little understanding of how they worked. Our initial response had been based on watching the latter, less capable crackers and was insufficient to handle the more sophisticated ones.

After much deliberation, it was decided that the only way to protect the computers on campus was to block certain key incoming network protocols, re-enabling them to local machines on a case by case basis, as each machine had been cleaned up and secured. The rationale was that if the crackers had access to even one unsecure local machine, it could be used as a base for further attacks, so it had to be assumed that all machines had been compromised, unless proven otherwise.

The recommendation to filter incoming traffic was presented to the Associate Provost for Computing on Monday afternoon and approved. The necessary equipment for the filter and monitor machines was bought or borrowed late that afternoon, and the design and coding of the filter proceeded through the night. Particular effort was made in the design to achieve the necessary security with the minimum of impact to local users. The filter was completed and installed by 5 PM Tuesday, 1 September.

At this point, the major task of analyzing all of the detailed logs and captured files was restarted. It was discovered that over 40MB of the cracker’s tools had been captured, tools that they had FTP’ed onto some of the broken machines. These tools included Crack, network monitoring tools, all SunOS, Ultrix and Dynix source code (so they could replace any executable on the system), and cracking programs for virtually every CERT announced vulnerability. The logs showed that the crackers routinely placed back door and trojan login binaries on each broken system and used programs to set the timestamp and checksum of the replaced binaries to avoid detection.

On Thursday, 3 September, TAMUSC monitor logs showed an obviously automated attack by ftp that was sequentially probing every machine on campus. Here again it was decided to monitor this attack, as it was not clear what it could accomplish. This decision to observe, rather than immediately block, turned out to be very fortunate.

Shortly after midnight on Friday, 4 September, TAMUSC received a report from another site via the Computing Emergency Response Team (CERT) at Carnegie Mellon that the crackers had broken back into TAMU machines. The logs were immediately analyzed, and it was determined that the crackers had used ftp to install a program that allowed them to tunnel past the TAMU filter's blocks. In addition, even though they knew we were aware of their original intrusions, they continued their pattern of breaking in and replacing key system binaries.

At this point, the filter was completely redesigned to keep the crackers out, and the new version was installed by 5 AM Saturday. The new version changed the filter approach from “deny” based filtering (let everything in unless it is specifically denied) to “allow” based filtering (block everything unless it is specifically allowed). This new version, while providing much greater security, was unfortunately also more visible to valid users.

Since the new filter was installed, no successful intrusion attacks against TAMU machines have been observed, despite continued logging of probes and continued attempts. Recent efforts have centered in three areas: improving the ease of use and throughput of the filter, reducing the manpower requirements of the monitoring tools, and developing a program to help local system managers check their machines for proper security configuration.

Highlights of the Cracker Sessions

While all techniques used by the crackers aren't specified, the following section shows some of the more interesting things that were discovered. In all cases, references to specific machines have been changed; all of the spelling errors have been left in. The first fact is that the crackers seemed to have a compulsion to discuss their exploits with other crackers. While IRC seemed to be the preferred technique, many of the better crackers preferred less obvious methods, such as simply cat'ing directly to each other's ttys.

This snippet records an unnamed cracker on host1 talking with NMN (No Means No) concerning having run "ch", a brute force password cracking program on all 10 HP Snake machines in a Kent State lab. This conversation occurred on host1.tamu.edu using "cat >/dev/tty…".

NMN: "Well the people who run all 10 of the HP's will core when all thier logs show NMN.and especially if they find password crackers on them all.. me writing it is one thing ,but installing it is ANOTHER"

host1: "How would they find NMN on the hp.You lost me."

NMN: "Theyd see me Ftp to acct nmn on host2"

host1: "Possibly.Not liekly ..Unless they suspect something.Kent is the least concerened about security of any host i a have a ever been on.Oh maybe bsides .ai.mit.edu systems."

NMN: "Yeah ok"

host1: "Anyways.. dont forget to check up on the ch im running at host3.. its PID 16684 (ps -p 16684 will see if its still there) At host3, it should take 12.59712 days. *grin* BUT it sould crack it.. its running right now, it could crack it tonight, could crack it next week, who knows."

One extreme form of communication involved one cracker setting up a clandestine conversation bulletin board called LIMX (Local Immediate Message eXchange) on host4.tamu.edu, a machine that they had broken. LIMX was implemented as a passwordless back-door login by replacing the login executable. Here is the logout message from LIMX:

> "Connect to us again sometime, dont forget to spread the word about our system (host4.tamu.edu/128.194.xx.xx) and the account name (limx) to all your friends. If you dont have any friends, thats ok, tell your family members! Imagine the fun at the dinner table if Mom, Dad, sister and brother (and of course your dog fido) all had computer terminals and were connected online to the BACKDOOR LIMX, all chatting about the latest gossip, sports reports, fashon tips, and the shocking crimes being committed (which of course aren't related to our wonderful hypocrisy 'democracy'), and of course the wonderful meal moms been slaving all day over a hot kitchen microwave making, which none of you can comprehend. So spread the word! And connect to us again! Anonymous Cyberpunk Number One Oh yeah, and thanks to NoMeansNo for making this wonderful program! Sure beats IRC!"

3. Package Overview

Response Overview

Response to the intrusion incidents has three major thrusts: filtering, monitoring, and cleaning. The first line of defense is the bridging filter package *drawbridge*, which is used to filter all packets to or from the internet. *Drawbridge* allows internet access to be controlled on a machine by machine and port by port basis on a full T1 bandwidth basis. Using the filtering built into the TAMU WAN router (cisco) was initially considered, but it was determined that our requirements, particularly in the need for supporting potentially different filtering to each of the roughly 5,000 IP machines in the TAMU class B network, were too complex for the router. In addition, something was needed that could handle full T1 bandwidth, was itself very secure, and could be implemented rapidly. While other firewall configurations are known to be stronger, *drawbridge* provides a level of compromise between security and availability more acceptable to the university environment and provides much needed flexibility and throughput for the TAMU large scale network.

Realizing that *drawbridge* was a compromise between convenience and security, a set of monitoring tools was developed to look for intrusions that might be attempting to circumvent the filter. These tools continuously monitor the internet link, checking for unusual connections, patterns of connections, and for a wide range of specific intrusion signatures.

The third major thrust has been the development of the *tiger scripts*, an automated tool for checking a given machine for signs of intrusion and for network security related configuration items.

Figure 1 shows an overview of the filter and monitor implementation. In traditional secure gateways, a filter and secure bastion host are used and all traffic to or from internet is forced through them. This typically means that users need proxy clients for external access, such as for telnet and ftp, so that they all do not have to log on to the bastion host for external access. At TAMU, the filter allows arbitrary protocol filtering on a host by host basis, so that each department can set up its own authorized hosts with their own service configurations (subject to the campus wide minimum standards). This provides a reasonable level of both security and flexibility for educational and research requirements. For a UNIX host to be enabled at all beyond the default incoming permissions for mail, it must pass the *tiger scripts*, as described later. The monitor node is placed outside the filter so that it can record connection attempts which are blocked by the filter. This placement has been crucial to recognizing intrusion attacks, but does place the monitor itself at risk. To minimize this risk, both the filter and monitor are placed in a controlled access machine room and the monitor is configured for secure network access. The filter is similarly programmed only to respond to secure filter update requests, which are not routeable.

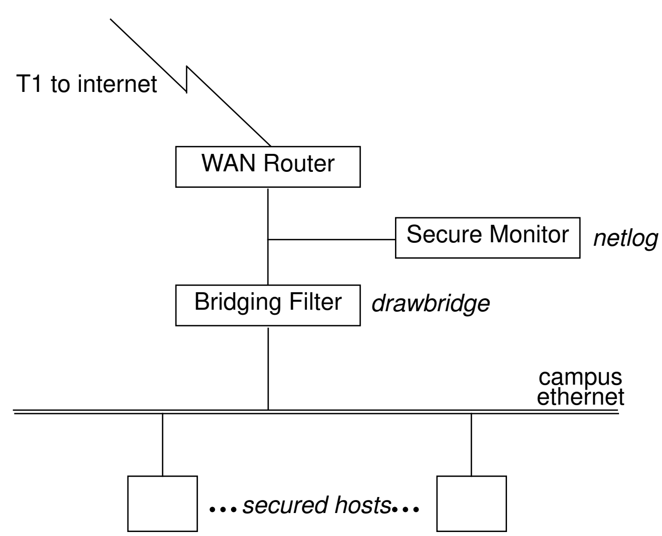

Figure 1. Overview of Components.

Filter (drawbridge)

Chapman [1] presented an interesting analysis of the limitations of current filter implementations at the Third UNIX Security Symposium. The drawbridge program, along with its support filter specification language and compiler, address some of his critical recommendations with respect to both functionality and ease of specification.

The filter approach chosen was based on a PC with two SMC 8013 (AKA Western Digital) ethernet cards. The first software implementation was based on pcbridge by Vance Morrison. This initial version was soon rewritten from scratch in C (compiled with turbo C++) to make the addition of needed features somewhat easier. The current filter design provides “allow” based filtering per host with separate incoming and outgoing permissions.

For both performance and configuration management, the filter tables are created on a support workstation, based on a powerful filter configuration language, and then securely transferred to the filter machine, either at boot time or dynamically during operation. The support machine does all the hard work of parsing the configuration file, looking up addresses, and building the tables, so that the filter itself need only perform simple O(1) table lookups at run time. Updating the tables dynamically is made secure with a Data Encryption Standard (DES) authentication.

The current default configuration allows any outgoing connection, but basically allows in only smtp (mail). Several campus and departmental servers have been checked and set up as hosts that are able to receive incoming telnet, ftp, nntp, and gopher requests.

Monitor

The goal of monitoring is to record security related network events by which intrusion attempts can be detected and tracked, particularly in those services allowed through the filter. This is a very difficult problem in general. The communication data rates make this problem somewhat like trying to take a sip of water from a fire hose; TAMU has some 30 terabytes of internal data transfer per day, and its internet connection is on the order of 4 gigabytes per day, with an average of 100,000 individual connections during that period. Clearly, monitoring needs to be both very selective and flexible, and automated tools are needed for reviewing even these resultant logs. Another problem is that of monitor placement. It is important that monitors be placed so that critical segments can be observed and so that the monitors themselves are secure.

Our solution includes the programs *tcplogger*, *udplogger*, *etherscan*, *nstat*, and some associated support programs. The tcp and udp loggers basically log a one line summary for all connection attempts. The associated analysis programs report on suspicious connections or patterns of connections. In addition, these logs have been very useful in analyzing details of security events after the fact. The *etherscan* program goes much further, actually scanning all packets and their contents, looking for a specific set of intrusion signatures, such as root login attempts from off campus. The *nstat* program collects statistics on all traffic to the filter and is useful both for capacity planning and for detecting unusual activity patterns. *Nstat* detected a clandestine FSP server on campus that was providing a repository of pirated commercial software, simply by noting a large transfer rate on a specific UDP port.

Machine Cleanup (the tiger scripts)

The phrase ‘Tiger Scripts' comes from the concept of a ‘tiger team.' A tiger team is a group which locates problems in a security system and demonstrates this problem by using it to circumvent the security system. By doing this, it is hoped that any weakness will located and corrected. The ‘Tiger Scripts' perform the first part of this task in the UNIX environment. They search through a UNIX system and report any elements of it which may represent a security risk.

After a series of intrusions were discovered at Texas A&M University, it became apparent that a large and unknown number of machines attached to the campus network had been compromised. A filtering bridge was installed between the campus network and the Internet in order to protect the machines at the campus. It was still necessary though to clean up the machines that were involved. Most (if not all) of these machines were UNIX systems, but there were only a few people available at the university with the knowledge to locate and correct the problems on these machines. The ‘Tiger Scripts' were developed to search out and report these types of problems. The scripts are also used as a means of verifying the security of machines to which access is allowed through the filtering bridge. Because of continuing development, the scripts have grown beyond just this internal use.

There were several goals for the ‘Tiger Scripts.'

- Ease of use
- Robustness
- Portability
- Functionality

It was essential that the scripts be easy to use, as they were to be used by persons who possibly had little UNIX systems management background. Towards this end, no user configuration of the scripts are required. Once unpacked, all that is necessary is to run the script ‘tiger.' This generates a report which contains any possible problems found. All messages are tagged with severity levels. These severity levels are used to indicate whether the system would be cleared through the filtering bridge. At the university, the manager of a system simply provides a copy of the report when requesting access.

It was also essential that these scripts be robust. Since they would be run on many different systems, the scripts had to be written in order to handle the nuances of the many administrators. This is of most concern when parsing system configuration files, as this is often where security problems manifest themselves.

In addition to these goals, portability had to be considered as well. The flavors of UNIX running on machines at Texas A&M is diverse. The initial release of the scripts (October 1992) was targeted toward machines running SunOS 4.1.x, as these machines were the primary target during the intrusion. The second release (April 1993) incorporated support for SunOS 5.x and NeXTOS 3.0. Future releases will include support for other UNIX derivatives such as AIX 3.x, HP-UX, IRIX, and UNICOS. The scripts are designed, however, to perform a "best of their ability" attempt even when specific support is not provided for a system. The only primary requirement is that the systems Bourne shell support the defining of shell functions.

The scripts accomplish these goals, while at the same time providing the following functionality. System configuration files are checked for problems, system binaries are checked for alterations, or known security problems, and known signs of an intrusion are checked. There is also support for the extension of the checks as new problems are reported.

For ease of use, the *tiger scripts* label all outputs with an error classification:

ALERT	A positive sign of intrusion was detected.
FAIL	The problem that was found was extremely serious.
WARN	The problem that was found may be serious, but will require human inspection.
INFO	A possible problem was found, or a change in configuration is being suggested.
ERROR	A test was not able to be performed for some reason.

As an aid to ease of use, an explanation facility is provided with the *tiger scripts*. Explanations can be requested for specific messages, or an explanation report can be generated for the security report. The explanation report can automatically be inserted into the security report, with explanations following each of the security messages. The explanations describe the situation, why it is a problem, and how to correct it. Figure 2 shows an excerpt from a security report with explanations inserted.

The checking performed covers a wide range of items, including items identified in CERT announcements and items observed in the recent intrusions. The scripts use cryptographic checksum programs to check for both modified system binaries (possible trap doors/ trojans), as well as for the presence of critical security related patches.

At the present time, the *tiger scripts* have been configured for SunOS 4.1.x, SunOS 5.x, Nextstep 3.0, AIX 3.2, HP-UX, IRIX 4.0, and UNICOS 7.0 releases. The programs are largely table driven for ease of porting, and ports to other platforms are being worked on.

```
--WARN-- [acc012w] Login ID sundiag has uid == 0.

The listed login ID has a user ID of zero (0) and is not the 'root' account.
This should be checked to see if it is legitimate.  In any case, having login
ID's with a user ID of zero tends to lead to security problems, and should be
avoided (except for 'root')

--WARN-- [acc004w] Login ID csehlhp is disabled, but has a .rhosts file

The listed login ID is disabled in some manner ('*' in passwd field, etc), but
a non-zero length .rhosts file.  This can allow the login ID to continue to be
used.  Unless this has been specifically set up to provide some service, it
should be removed.

 --INFO-- [acc002i] Login ID vu18368 is disabled, and has a shell of /bin/login.

The listed login ID is disabled, but has a potentially valid shell. These can
usually be safely ignored, but should be checked.
```

Figure 2. Sample *tiger scripts* security report with explanations inserted.

Policies

The policies and procedures need to provide both security and flexibility. The resultant decision was to filter incoming traffic other than mail to all machines and then allow case by case requests for authorized hosts status, based on successful demonstration of basic security configuration with the *tiger scripts*. Special requests for allowing incoming requests to special servers that are not easily checked, such as for embedded robot controllers, have been made. In these cases, the connections have been allowed, but special monitors have been implemented on these services.

Long term policy questions that remain unanswered include how to handle updates in response to critical CERT announcements and how to handle OS updates. Obviously, some way to coordinate both periodic and quick response host reviews is needed. This filter configuration language does support machine classes, so it would be possible to do something such as “disable ftp to all SunOS 4.1.1 machines” in response to a CERT announcement of a respective problem, but it would be nice to have a mechanism to communicate such announcements to the respective managers *before* cutting off access. The problem on a large campus is maintaining a contact list for a large number of machines, given the high rate of turnover in student managers. In addition, the information in the filter configuration file may rapidly become outdated, as managers update their machines’ hardware and software. The current plan is to require periodic (annual) security checks with the current *tiger scripts*, enforced with the possible loss of IP authorization. In the case of aperiodic security events or announcements, an attempt will be made to evaluate the time criticality of responding and require appropriate event specific checking. As the *tiger scripts* are easy to run, it is anticipated that this requirement will not be a significant burden to system managers.

A recent case in point was the announced security problem with the wuarchive anonymous ftp code. In this case, it was known exactly which machines had ftp authorized, and the respective managers were contacted immediately. The managers updated their software so rapidly that it was not necessary to block access, and the limited number of authorized machines avoided the need for an immediate *tiger* update.

4. Filter (drawbridge)

Design

Drawbridge is different from any of the current standard firewall configurations. Using the categorization of firewalls developed by Ranum [2], *drawbridge* compares best to a filtering router firewall configuration as shown in figure 3. In a filtering router firewall, a router which has packet filtering support is used to filter packets to and from hosts on the "inside" of the router. This is used to establish a policy where hosts are provided more or less access depending on the decisions of the network managers.

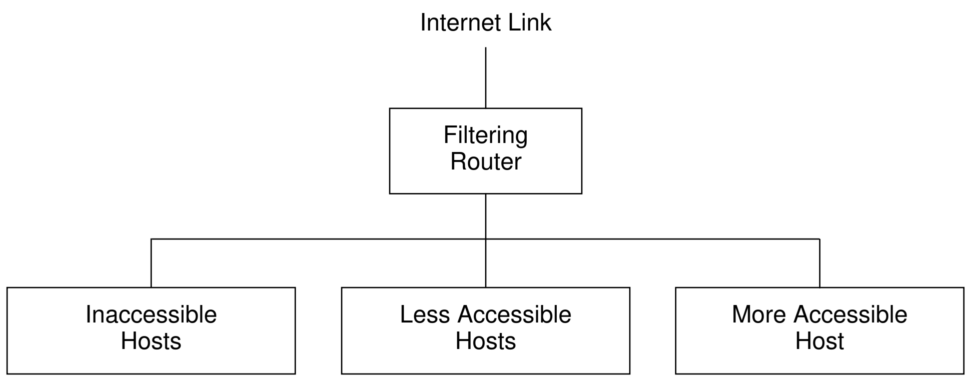

Figure 3. Typical filtering router firewall.

In an university environment, access typically would be based on the department the machine is located in, who manages/uses the machine, and an initial security audit, which is hopefully repeated on a regular basis. At TAMU, security audits are performed using the *tiger scripts* package, which was developed for this purpose. Any hosts that are never "registered" for access through the filter would receive some type of default access that would be defined by the network managers.

While this type of firewall is theoretically weak in comparison to other firewall methods, in practical use it does provide a useful increase in security. The points of attack have been greatly reduced and casual intruders are quickly discouraged.

A typical *drawbridge* firewall configuration is related to a filtering router firewall as shown in figure 4. The difference is that instead of using a filtering router as the firewall, the filtering function is moved from the router into *drawbridge* which acts as a bridging filter. Note, however, that figure 3 describes just a typical setup; a router is not a necessary component of a *drawbridge* configuration.

Comparison to Other Filtering Methods

Chapman [1] is an excellent source of information about packet filtering issues. He discusses the concepts behind packet filtering and some of the problems associated with it. He also discusses the problems with current implementations of packet filtering found in some current routing products.

Some of these problems include:

- Complex configuration language
- Difficult verification
- Lack of filtering on key parameters, such as source port or direction

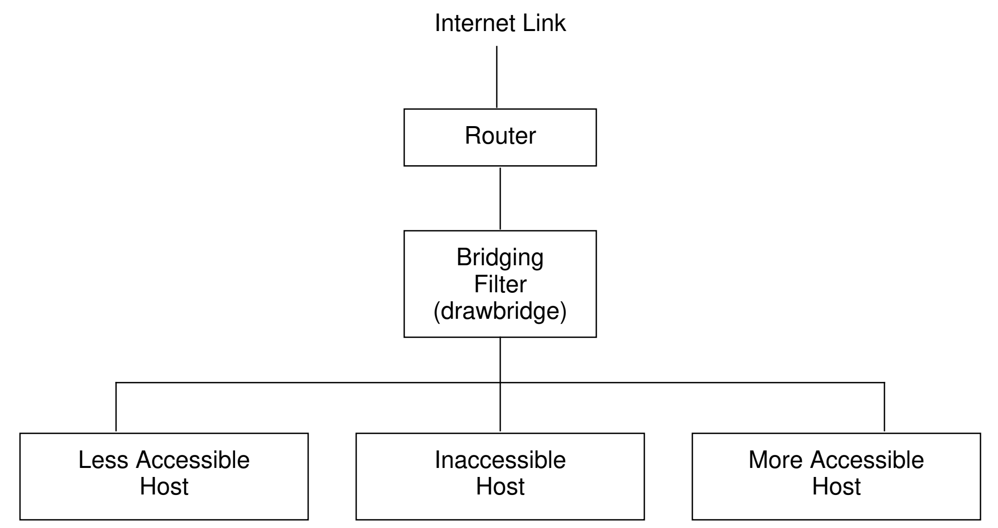

Figure 4. *Drawbridge* firewall.

Rather than repeat that material, it will be assumed that the reader is familiar with packet filtering and a discussion of how *drawbridge* tries to address some of these problems mentioned by Chapman [1] will be presented.

One of the first problems with current packet filtering implementations is that they are difficult to configure. They use a simple syntax that is designed for efficient implementation, not for effective configuration by an administrator. On a university campus, there is a need for many different filtering configurations to satisfy the diverse needs of the many users. Also, while the needs of administrators are usually defined in terms of connections, filters usually are defined in terms of packets only; the semantics of connections must be tediously mapped on to them.

These issues are addressed in *drawbridge* through the use of compiled tables. One table is defined for each (entire) IP network with each host address in that network being a single entry in the table. This allows a powerful source language to be designed that administrators can easily use and that is flexible enough to define complex sets of filters. In addition, *drawbridge*, under TCP, is not restricted to filtering in terms of packets but also filters in terms of connections. This makes configuration easier for the administrator and *drawbridge* more efficient.

This table design allows arbitrarily complex filters to be defined with little penalty. In conventional filtering routers, as filters are added, the performance begins to quickly drop due to how they implement the filtering rules. In *drawbridge*, arbitrary numbers of complex filters can be set up and the performance remains almost constant since simple look ups are performed and only connection establishment packets are filtered for TCP.

A second problem with most filtering implementations is that testing filter configurations is difficult. *Drawbridge* remedies this by allowing the administrator to check the results of a compiled configuration file to see if the correct filtering rules have been applied. Since *drawbridge* is less algorithmic than current filtering implementations, it is sufficient to investigate the compiler output. The administrator can look at the class that a host has been assigned and at the filtering lists defined for each subtable in that class.

A last problem that *drawbridge* addresses is the need for support for source port filtering. *Drawbridge* specifically defines an entire subtable to support TCP source port filtering (UDP source port is not currently supported). Since source port filtering does allow the possibility of tunneling, *draw-*

bridge does add the restriction that the destination port must be greater than 900; 900 was chosen due to certain FTP implementations that happen to use FTP data ports beginning at around TCP port 900 rather than following the BSD convention of starting at 1,024.

Physical Structure

Drawbridge is physically structured as shown in figure 5. The PC running *filter* is placed between the external network (Internet link) and the internal network (campus) that will be protected. Optionally, a Sun workstation can be used to communicate with and manage *filter* on the PC. *Filter* acts as a filtering bridge between the external and internal networks. *Filter* performs bridging but does not conform to bridging standards, e.g., it has no support for Spanning Tree Protocol.

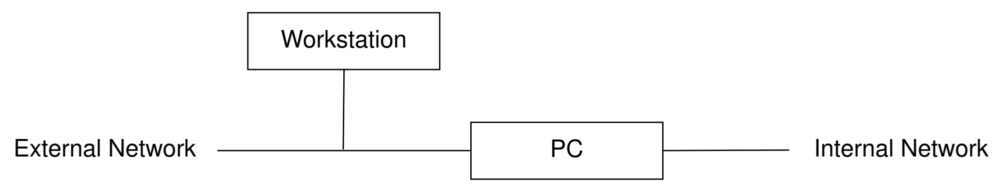

Figure 5. Ideal *drawbridge* Configuration.

In the ideal configuration, the workstation would be placed outside of *filter* so that monitoring of connection attempts from the external network can be performed. This is a good way to look for attacks and probes that are attempted against your internal network (and are hopefully blocked by *filter*). Since this also restricts the workstation's access to the internal network, a workstation will have to be committed specifically for this purpose. If a spare workstation isn't available, a machine on the internal network can be used to perform the management.

As mentioned above, *filter* is a table based filtering bridge. This approach was taken to improve the performance of filtering. The tables are generated by the following process (see figure 6). First, a source file containing filtering specifications in a special language is generated and maintained by an administrator. This file is then passed through *fc*, which generates the tables used by *filter*. These tables can be loaded via *fm* or by floppy disk.

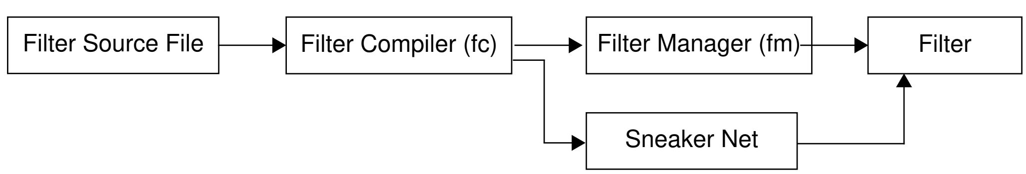

Figure 6. *Drawbridge* Compilation and Loading Process.

Filter Compiler Language

The language used by the compiler contains constructs for creating the various tables used by the filter. Constructs exist for specifying the network access on a per host basis, on a network or on a subnetwork basis. Groups of services can be created. These groups can be used in cases of related services or to group related machines. Access to particular external sites can also be granted, and access from certain sites can be denied. These constructs are shown in figure 7.

```
host (<hostname>|<ip_address>) <list of service_entries>
network (<ip_address>|<ip_address> - <ip_address>) <netmask> <list of service_entries>
define <group_name> <list of service_entries>
allow <ip_address> <netmask> <list of service_entries>
reject <ip_address> <netmask>
```

Figure 7. Constructs contained in the *filter* compiler language.

Hosts and networks can be granted network access using service specifications or group names. As hosts and networks are processed, the classes used by the filter are created. Hosts with equivalent network access (real access, not syntactic) will belong to the same class.

A group is a list of comma separated service specifications or other previously defined groups. Groups can be used to relate services or to categorize machines, allowing quick global changes to a category of machines. The special group "default" specifies the default access for any machine that does not match any of the networks loaded into *filter*.

Constructs also exist for building the allow and reject tables used by the filter. The allow table allows internal machines access to a restricted external service. The reject table is used to block all incoming packets from a host or network.

The basic element of the language is a service specification. The service specification contains four pieces of information: the service, protocol, source or destination, and traffic direction. The service can be either an entry from /etc/services or a numeric port. Service ranges can also be used. The protocol specifies the protocol the service uses. The source or destination indicates whether the filter should use the source port or the destination port. Finally, the traffic direction indicates whether this is for outbound packets, inbound packets, or both. The grammar defining a service specification is shown in figure 8.

An example configuration file is shown in figure 9.

```
<service_entry>  ::= <  (src=|dst=) <service_desc> (in|out|inout) > |
                     <! (src=|dst=) <service_desc> (in|out|inout) > |
                     <group_name>
<service_desc>   ::= <service> | <service_range>
<service>        ::= <port_number>                |
                     <port_number>  / <protocol> |
                     <service_name>               |
                     <service_name> / <protocol>
<service_range>  ::= <port_number> - <port_number>               |
                     <port_number> - <port_number> / <protocol>
<protocol>       ::= <protocol_number> | <protocol_name>
```

Figure 8. Service entry grammar.

```
# Defaults for any machine not listed in this file.
define default  <1-65535/udp in>, <!tftp/udp in>, <!sunrpc/udp in>,
                   <!2049/udp in>, <1-65535 out>, <src=ftp-data in>,
                   <smtp in>, <auth in>, <gopher in>;

# Admin requested no access in/out for this subnet
network 123.45.58.0 255.255.255.0 <!1-65535 in-out>;

# NNTP host and CSO phonebook server
host mailnews.tamu.edu          default,
                                <nntp in>, <time in>,
                                <csnet-ns in>, <domain in>,
                                <finger in>;

# Machine (PC) in library which uses tftp to do document transfers
host sender.tamu.edu            <1-65535/udp in>;

# Has to have X
host arrow.tamu.edu             default, <ftp in>, <6000 in>;

# No TCP access in/out
host bee.tamu.edu               <!1-65535 in-out>;
```

Figure 9. Example configuration file.

How filter Works

fc generates four different kinds of tables (see figure 10):

- Multiple "network" tables, with one entry per host address,
- A "class" table with one permission list per distinct service class,
- A global "allow" table, and
- A global "reject" table.

The network tables have an entry for each host in the network. The host portion of an address (ignoring any subnetting) determines the index into the table. The value in the table defines the “class” that will be applied to a host when a packet is to be filtered. Only class B and C networks are currently supported as *filter* does not have the capability to use any memory above 1 MB.

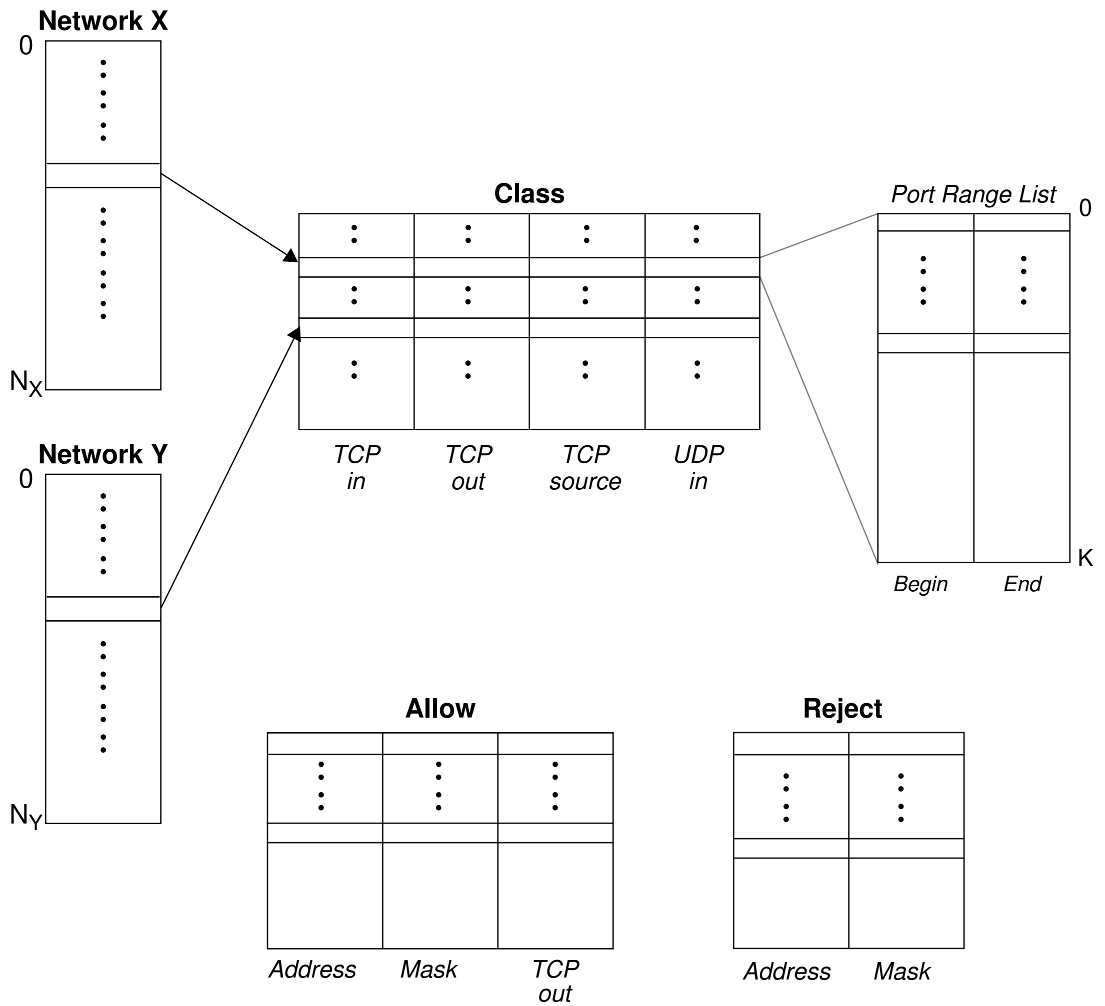

Figure 10. The four tables generated by *fc*.

Note that the network and class tables are defined in terms of a host on the internal network. No filtering is done based on the address of a host outside of the filter except on a global basis for reject and allow. It is assumed that an inside host will control which outside hosts are allowed to access its services, e.g., using TCPWrapper. *Filter* only controls which internal host's services are open, not which external hosts may access an internal host's services.

The host's class is used as an index into the class table. This second table is composed of four subtables: TCP in, TCP out, TCP source and UDP in. The subtables are composed of lists that contain port number ranges. A class specifies a list out of each subtable that defines a host's filtering. It is important to note that the TCP filtering only occurs when ACKless SYNs (connection initiation) are detected in a TCP header. All other packets of a TCP session are not filtered. Also, all UDP packets are filtered on an incoming basis only.

The last two tables are the allow and reject tables. The allow table globally allows packets out from any machine on the inside of the filter to the list of addresses in the allow table using the supplied list of port number ranges. The reject table globally rejects packets coming in from any machine on the outside of the filter with an address corresponding to an address in the reject table.

Figures 11 and 12 are flowcharts describing how *filter* uses these tables to check connections for incoming and outgoing requests, respectively.

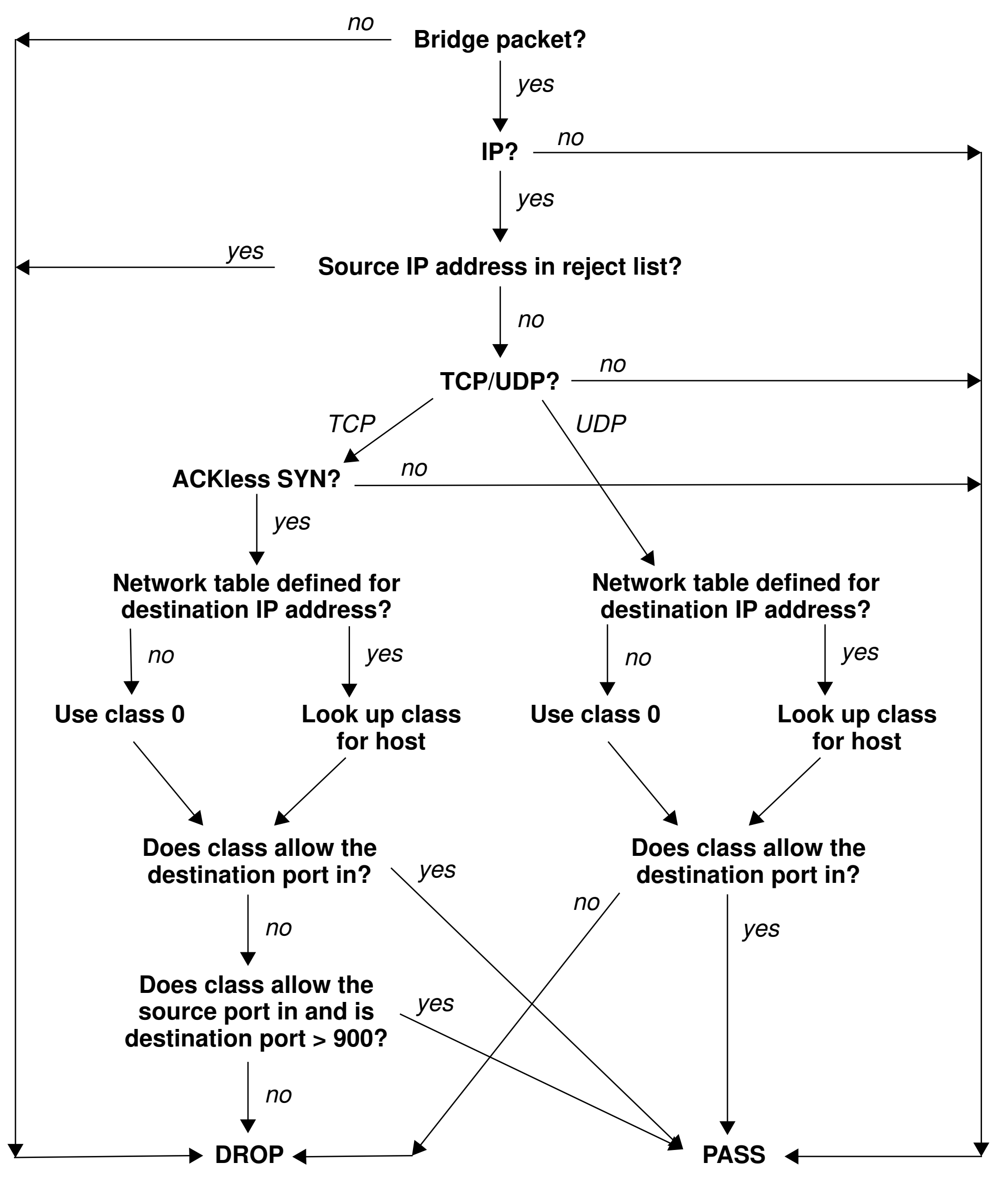

Figure 11. Incoming Packet Filtering Algorithm.

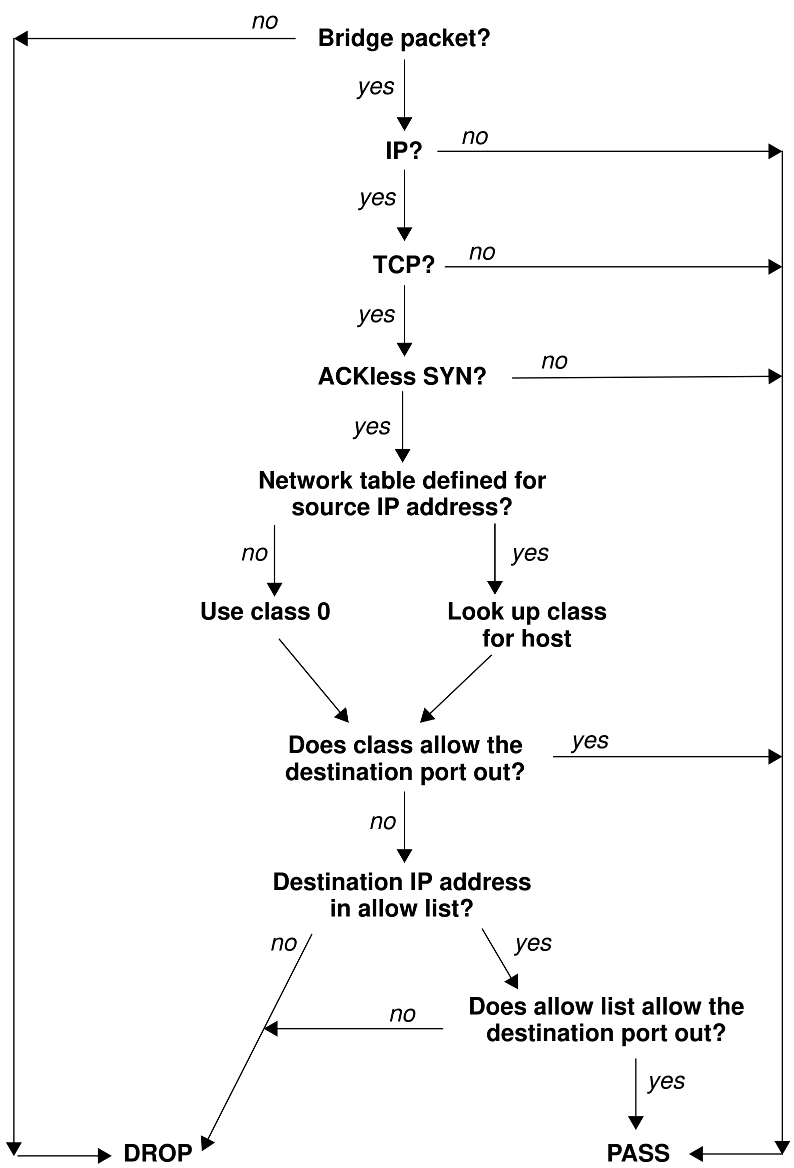

Figure 12. Outgoing Packet Filtering Algorithm.

Implementation

Filter started out as a simple modification to PCBridge, a public domain program written in assembly. It is now over 3,000 lines of C code. However, the efficiency of PCBridge has been retained and even improved upon. *Filter* double buffers outgoing packets and has code to perform adaptive bridging using a hash table.

Figure 13 shows the performance of *filter* under different loads. The input packet rates were generated and monitored with a combination of two workstations and an ethernet analyzer with the packets generated on one side of the filter and measured on the other. These packets were built such that all of them would pass through the filter (not be bridged) and would not experience any higher level protocol backoffs. A set of fixed sizes and a sample distribution of sizes were chosen with the sample distribution being based on the average distribution of packet size on our current campus ethernet backbone. While the filter peaked at about 2.5 Mb/s for the sample distribution, our campus backbone averages around 1.3 Mb/s. Our conclusions from these tests are that the filter is limited by the PC hardware and not the software. The configuration tested was a 33 MHz 486 with two 16 bit SMC ethernet cards on an 8 MHz ISA bus. By simply increasing the speed of the ISA bus to 11 MHz, we saw the performance threshold of the filter on 1514 byte packets jump by 1.0 Mb/s. We feel that 32 bit ethernet cards on an EISA bus would increase the performance greatly.

Memory usage depends greatly on the number of network tables that are loaded into *filter*. At TAMU we currently have one class B IP network that is filtered. *Filter* uses approximately 300 KB of memory in this configuration.

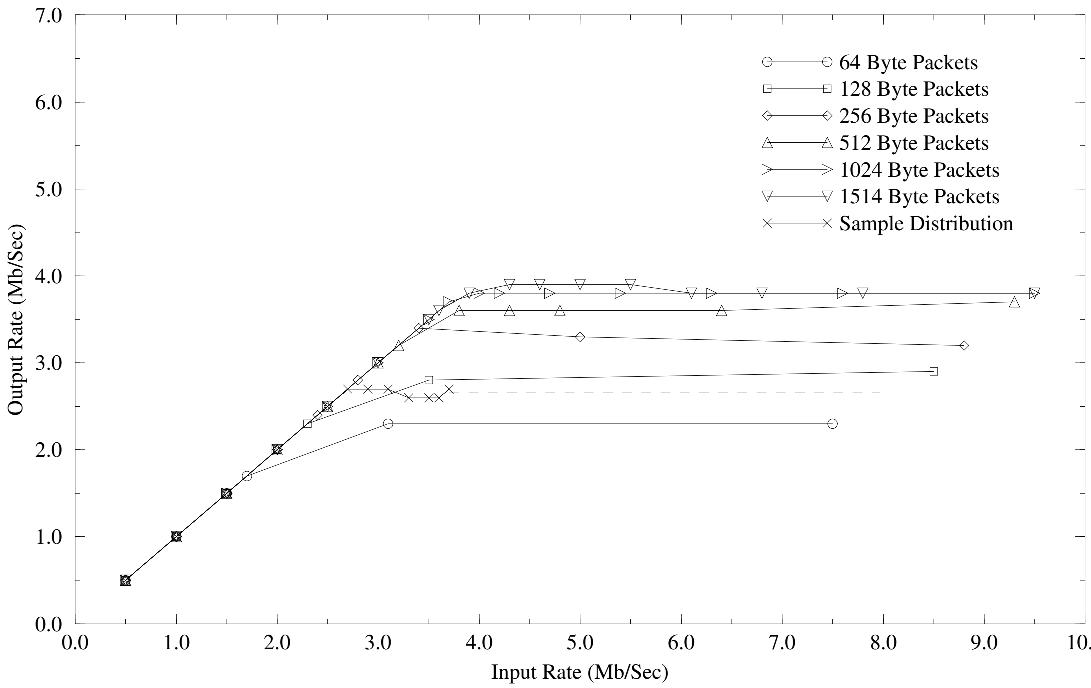

Figure 13. Performance of *filter* under different loads.

5. Monitor

Service Initiation Logging

The first tool, which is actually two tools (*tcplogger* and *udplogger*), records the initiation of a TCP session or UDP session. The start of a TCP session is indicated when the FLAGS field of the packet has the SYN flag set with no other flags set. A record is written to a log file, ASCII or binary, for each session. For a binary log file, the format of the record is:

```
struct sessinit {
      struct timeval start_time; /* Initiation time */
      unsigned long  ip_src;     /* IP source address */
      unsigned long  ip_dst;     /* IP destination address */
      unsigned long  tcp_seq;    /* TCP sequence number */
      unsigned short srcport;    /* Source port */
      unsigned short dstport;    /* Destination port */
};
```

The TCP sequence number is recorded so that duplicate packets can be removed in a post-processing phase. This simplifies the recording tool.

Detecting the initiation of a UDP session is not as simple. The UDP logging system uses a heuristic to determine the start of a session. Whenever a UDP packet is received, a table of "active" sessions is searched. If the packet belongs to an active session, that session's active time stamp is updated. If no active session is found, then a new "active" session is created and the packet is logged, using the same record structure as used by the TCP logging system. Since UDP packets do not have sequence numbers, the sequence number field is set to zero. After a packet has not been received for a session for an (user specifiable) amount of time, that session is deleted.

Both tools use the SunOS 4.1 Network Interface Tap (nit) with packet filters (nitpf). The use of the packet filter significantly enhances the performance of these two tools.

Service Initiation Log Processing

The binary log files created by *tcplogger* and *udplogger* contain records of all the TCP and UDP sessions that have occurred. This is, of course, a large number. On a normal day, there are over 100,000 TCP sessions and around 50,000 UDP sessions occurring on the TAMU Internet link. A tool was developed for extracting only those records of interest. The tool, *extract*, uses an "awk"-like language for selecting records and printing them. Records can be selected based on source or destination port, host, and network, date, and time. Selectors can be grouped using the boolean "and" and "or" operators. An example *extract* script is shown in figure 14. *Extract* can generate ASCII or binary log files. The binary log file uses the same format as the TCP and UDP logging tools, thus allowing further processing to occur. Since there is no information in the binary log file to indicate whether it is a TCP or UDP entry, the two can not be mixed, and *extract* must be informed of the type of file it is processing.

In practice this restriction is not a problem. There is generally a lot of noise with UDP traffic (traceroutes, FSP, etc.). If mixed together with the TCP log data, a TCP connection might be lost in the noise. Separating them into two log files eliminates this problem. As we normally maintain two logs for this reason, tagging each record by type so that the logs can be combined, is not of significant benefit.

```
#!/usr/local/etc/extract -f
#
#  Print out interesting TCP events coming from the Internet
#
srcnet = 128.194.0.0 {next}  # Skip sessions originating from A&M
dstport = shell ||
  dstport = exec ||
  dstport = 6000 {print; next}
srchost = terminus.lcs.mit.edu ||
  srchost = nyx.cs.du.edu {print; next}
dstport = smtp || dstport = telnet || dstport = finger ||
  dstport = 113 || dstport = ftp || srcport = ftp-data {next}
dstport = nntp && dsthost = news.tamu.edu {next}    ||
dstport = 2000 && (
    dsthost = mud1.tamu.edu ||
    dsthost = mud2.tamu.edu) {next}
dstport > 1023 {next}
{print}  # Print anything that makes it to here
```

Figure 14. Example *extract* script.

Protocol Signature Analysis

While the TCP/UDP logging tools allow us to detect when someone is probing the campus machines for tftp, or some related activity, they don't tell us what happens when someone connects to a system via telnet or some other TCP/IP service. The *etherscan* tool provides this capability. *Etherscan* monitors certain protocols for unusual activities. These protocols are the ones normally allowed through the filtering bridge, i.e., telnet, ftp, smtp. The specifics of what is watched for will not be discussed here, as we do not want potential intruders to know exactly at what we are looking. One example though is attempts to login using system account names, e.g., “root.” Another feature of *etherscan* is the ability to detect and report FSP servers. FSP is a UDP based file transfer system which has found favor among those wishing to keep their activities hidden from network and system administrators. A sample listing of the output of *etherscan* is shown in figure 15. At TAMU, there are approximately 20 records recorded per hour. Most of these are due to people attempting to login as the user ‘guest' or ‘anonymous'.

As with *tcplogger* and *udplogger*, *etherscan* uses the SunOS 4.1 Network Interface Tap and the packet filtering mechanism for performing packet captures.

```
05/26/93 08:35:54.19 [smtp] some.host.edu.2992 HOSTAT.TAMU.EDU cmd help
05/26/93 08:36:00.08 [smtp] some.host.edu.2992 HOSTAT.TAMU.EDU cmd help data
05/26/93 08:36:04.19 [smtp] some.host.edu.2992 HOSTAT.TAMU.EDU cmd help mail
05/26/93 08:36:19.84 [smtp] some.host.edu.2992 HOSTAT.TAMU.EDU cmd help rcpt
05/27/93 11:16:49.44 [udp] AHOST.UDE.EDU.21 possible FSP server.
05/27/93 13:23:30.42 [ftp] 128.194.180.66.27935 out.there.edu attempted login
                           ‘USER'
05/27/93 13:23:31.87 [ftp] 128.194.180.66.27935 out.there.edu 530 User USER
                           access denied..
05/27/93 13:55:37.29 [telnet] 128.194.225.211.33633 over.there.edu attempted
                              login as ‘system'
```

Figure 15. Example output from *etherscan*.

Traffic Analyzer

The final tool, *nstat*, is used to locate changes in network traffic patterns. This tool was originally written in order to gather statistics on the usage of various protocols on the Internet link. By recording these levels on an hourly basis, changes in the usage of a protocol can indicate someone attempting to bypass system security. Using this, an unauthorized FSP server was discovered on the campus when the UDP port that was being used suddenly increased in utilization (this was before support had been added to *etherscan* for detecting FSP servers).

Nstat has a raw output in ASCII format, although it is not really intended for direct viewing. Two programs, *nsum* and *nload*, are provided to analyze the raw *nstat* output. *Nsum* is a PERL program that summarizes the utilizations statistics, giving an ASCII histogram of the top ten usages at the ethernet, IP, TCP, and UPD levels. *Nsum* has support for analyzing certain time periods, such as morning, afternoon, etc., and for selecting weekday versus weekend times. The second analysis program, *nload*, is a simple awk program which produces data suitable for graphing with a tool such as xvgr. A parameter file for xvgr is included. Figure 16 shows a section of the raw *nstat* output. Figure 17 shows a histogram produced by *nsum*.

Ethics

Many may question the ethics and legality of such monitoring. We feel that our current system is not a privacy intrusion. The TCPLOGGER and UDPLOGGER are simply the network equivalent of process accounting, as they log routine network events, but none of the associated user level data associated with the event. Etherscan similarly reports unusual network events, which is the network equivalent of logging failed login attempts.

```
#Start Wed May 19 18:48:01 1993
#Stop  Wed May 19 19:22:52 1993
#550641 packets, 78169154 bytes, 9178 802.3, 2 runt, 3320 missed.
e 600       # 175        b 15882
e 800       # 457927     b 67569221
e 806       # 235        b 14104
e 889       # 184        b 11452
i 1         # 819        b 63686
i 6         # 446119     b 66005034
i 9         # 87         b 104942
i 17        # 10642      b 1375485
t 20        # 40416      b 16027884
t 21        # 1263       b 96426
t 23        # 81314      b 6522485
t 25        # 18540      b 3092486
t 37        # 8          b 480
t 53        # 58         b 4104
t 70        # 11347      b 4266897
t 79        # 262        b 22230
t 113       # 26         b 1596
t 119       # 108098     b 16296687
u 42        # 6          b 360
u 53        # 6634       b 628649
u 111       # 4          b 536
u 123       # 3622       b 325980
u 125       # 67         b 7102
u 127       # 67         b 7102
u 161       # 156        b 13950
u 213       # 24         b 1776
u 513       # 92         b 9960
u 514       # 3          b 348
u 517       # 6          b 552
u 518       # 137        b 17142
u 520       # 4864       b 954544
```

Figure 16. Raw *nstat* output.

```
Utilization: 68.44%

ETH IP        (50%/57%):#####################################################
ETH DECIVDNA  ( 3%/ 4%):###
ETH DECLAVC   ( 3%/ 4%):###
ETH oldIPX    ( 2%/ 3%):##
ETH DECLAT    ( 1%/ 1%):#
ETH Banyan    ( 0%/ 0%):
ETH DECRCONS  ( 0%/ 0%):
ETH DECLBM    ( 0%/ 0%):

IP  TCP      (56%/47%):########################################################
IP  UDP       ( 3%/ 3%):###
IP  ICMP      ( 0%/ 0%):

TCP ftp-data (18%/14%):##################
TCP nntp     (17%/13%):#################
TCP 2000     ( 7%/ 6%):#######
TCP telnet   ( 5%/ 4%):#####
TCP 175      ( 3%/ 2%):###
TCP smtp     ( 3%/ 2%):###
TCP 6667     ( 2%/ 1%):##
TCP 1023     ( 1%/ 1%):#

UDP route    (17%/ 1%):#################
UDP domain   (14%/ 1%):##############
UDP ntp      ( 6%/ 0%):######
UDP 2074     ( 3%/ 0%):###
UDP 1081     ( 2%/ 0%):##
UDP 1025     ( 1%/ 0%):#
```

Figure 17. Example *nsum* output.

6. Machine Cleanup (the tiger scripts)

Structure

The *tiger scripts* consist of one primary “driver” script named *tiger*, a set of scripts which check various components of the system, support scripts, and support files. The driver script, *tiger*, executes each of the component scripts. Its usage is:

```
tiger [-B tigerhomedir] [-d loggingdir] [-w scratchworkdir]
```

The configuration file, *tigerrc*, can be used to enable or disable the different component checks. This allows certain fast executing components to be executed frequently, while other, longer executing components can be executed less frequently.

The driver and component scripts make use of many support scripts and data files. These are used to make the scripts portable. The support scripts are used to set internal variables, such as the pathnames to the UNIX commands used by the scripts, and to convert system configuration files into the formats the main scripts can parse. For example, one of these scripts is used to generate ‘/etc/passwd'-like input from the various locations that this information is obtained on a particular system. They are also used to provide functionality that a particular system may be lacking. The data files contain information that is compared against information found on the system. For example, one of the files contains the permissions expected on system files and directories.

Component Scripts

The checks performed by the *tiger scripts* are broken out into several component scripts. Ordinarily, these are all executed by the driver script *tiger*, but they can all be executed directly.

Many of the scripts check the ownership and access permissions for files. There are two different types of checks. They will be distinguished in this paper by referring to them in the following manners.

A check of a ***pathname*** means that all components of the pathname are checked. If the support module ‘realpath' is available (i.e., it compiled), any symbolic links are handled as well. As an example, on SunOS 4.x systems, ‘/usr/spool' is a symbolic link to ‘/var/spool'. Therefore, the pathname ‘/usr/spool/cron' contains the components ‘/usr', ‘/var', ‘/var/spool', and ‘/var/spool/cron' which will be checked. All ‘‘incorrect'' ownerships or access permissions along the pathname are reported in detail.

A check of a ***filename***, ***directory*** or ***file*** means that only the ownership and access permissions of the file referred to by the filename are checked.

The ownership and permissions requirements are user configurable. The default configuration requires that pathnames to system executables and files be owned by root and writable only by root.

check_accounts

The ‘check_accounts' script examines user accounts for the following:

- passwordless accounts
- checks home directory ownership and access permissions
- checks ownership and access permissions of configuration files
- disabled account with .rhosts file, cron entries or .forward file that executes a program.

```
--WARN-- [acc004w] Login ID nag is disabled, but has a .rhosts file
--WARN-- [acc006w] Login ID dave's home directory has group
         `apcis' write access.
```

check_aliases

The ‘check_aliases' script examines the system mail aliases file. It reports any program aliases, and also checks the pathnames to the executable for proper ownership and access permissions. It also verifies that pathnames to any included files have proper ownership and access permissions (the entries in the included files are also processed).

```
--INFO-- [ali005w] Alias `drawbridge-server' contains a program
         entry: |"/xpub/secure/mailserver"
```

check_anonftp

This script checks for an anonymous FTP setup, and if one is found checks the integrity of it. Ownership and permissions of critical files and directories are checked. It also reports on any writable directories that are found.

```
--WARN-- [ftp010w] ~ftp/xpub/ftp/bin is writable by 'ftp'.
```

check_cron

The ‘check_cron' script checks user ‘cron' files. It examines the ownership and permissions of the pathnames used by each cron entry. Executables without absolute pathnames are also reported. The script attempts to be smart and interpret complex cron entries.

```
--FAIL-- [cron003] cron entry for root uses `/var/netlog/bin/logit'
         which contains `/var/netlog' which is group `wheel' and
         world writable.
         /var/netlog/bin/logit stop
```

```
--WARN-- [cron002] cron entry for root uses '/home/accts/doug/
         tiger-2.1.1/tigercron' which contains '/home/accts/doug/
         shop' which is not owned by root (owned by doug).
         /home/accts/doug/tiger-2.1.1/tigercron -B
         /home/accts/doug/tiger-2.1.1 -l
         /var/spool/tiger -w /var/spool/tiger/work -b
         /var/spool/tiger/bin > /dev/null 2>&1
```

check_exports

This script checks the server side of an NFS machine. Filesystems exported with "root" access, are reported, although the script does understand diskless clients and is (usually) quiet about these. It also reports exporting of the root directory (/), unrestricted exports, etc.

```
--WARN-- [nfs011w] Unprotected directory /fs is exported with root
         access to host(s) HOSTX.TAMU.EDU.
--WARN-- [nfs011w] Unprotected directory /export/export1/sunos5 is
         exported with root access to host(s) thea, and
         poshost.
--INFO-- [nfs010i] Directory /export/export1/root/Xkernel.sun3x is
         exported with root access to host(s) xhost.
```

check_group

The 'check_group' script cross references all of the sources of 'group' information for consistency. Any conflicting group ids or group names are reported. The group files are also checked for correct format.

```
--WARN-- [grp004w] GID conflict for group 'system' between
         /etc/passwd (gid = 2) and NIS (gid = 128).
--WARN-- [grp005w] Groupname conflict for gid 8 between
         /etc/passwd (group staff) and NIS (group grads).
```

check_inetd

The 'check_inetd' script examines the configuration files '/etc/inetd.conf' and '/etc/services.' It looks for things such as mismatched entries (i.e., telnetd on a port other than 23). It also reports any services which have been added from the standard distribution. The pathnames of executables are checked for ownership and access permissions.

```
--WARN-- [inet005w] Service login is using /xpub/etc/in.rlogind
         instead of /usr/etc/in.rlogind.
--FAIL-- [inet009] inetd entry for login service uses '/xpub/etc/
         in.rlogind' which contains '/xpub/etc' which is group
         'daemon' and world writable.
--WARN-- [inet005w] Service shell is using /xpub/etc/in.rshd
         instead of /usr/etc/in.rshd.
--FAIL-- [inet009] inetd entry for shell service uses '/xpub/etc/
         in.rshd' which contains '/xpub/etc' which is group 'dae-
         mon' and world writable.
```

check_known

This script tests for known signs of an intruder. Directories known to be used by intruders are checked for unexpected files. These directories include ‘lost+found' directories, system mail-spool directories and window server directories. The setuid(2) system call is also checked to verify that it is working properly.

```
--ALERT-- [kis001a] /usr/uucp/.sys is a directory."

--ALERT-- [kis002a] /usr/spool/uucppublic/.hushlogin is not zero-
         length."

--WARN-- [kis008w] File ".stuff" in the mail spool, owned by ‘bin'.
```

check_netrc

The ‘check_netrc' script examines the .netrc files in user home directories. It reports if the permissions on the file are incorrect (i.e. world readable). It also reports any entries in the .netrc file which contain passwords which are not anonymous ftp entries.

```
--WARN-- [nrc002w] User imauser's .netrc file contains passwords
         for non-anonymous ftp accounts.

--FAIL-- [nrc001f] User urauser's .netrc file is readable and
         contains passwords for non-anonymous ftp accounts.
```

check_passwd

The ‘check_passwd' script performs the same functionality as the ‘check_group' script, except it works with "passwd" sources. Sources of password information are cross referenced for conflicts and also check for correct format.

```
--WARN-- [pass004w] UID conflict for login ID ‘smith' between
         /etc/passwd (uid = 125) and NIS (uid = 1388).

--WARN-- [pass005w] Username conflict for uid 6 between /etc/passwd
         (login ID sys) and NIS (login ID joeuser).
```

check_path

The ‘check_path' script checks the PATH variable as set in the various shell startup files. It checks for ‘.' (dot) in the PATH, correct ownership and access permissions of pathnames in the PATH and correct ownership and access permissions of executables in the PATH. By default, the script only checks the PATH for the root account. It can be configured to check all users, though the author is not comfortable with this.

```
--INFO-- [path008i] Setuid program /usr/bin/uux in root's PATH from
         .cshrc is not owned by root (owned by uucp).

--WARN-- [path002w] /usr/bin/ls in root's PATH from .profile is not
         owned by root (owned by bin).

--INFO-- [path006] The PATH set in root's .profile contains
         ‘/usr/bin' which contains ‘/usr' which is not owned by
         root (owned by bin).
```

check_printcap

The script ‘check_printcap' checks the pathnames to filters used by printers for proper ownership and access permissions. Since not all systems use the BSD print system, this script is invoked only for systems on which it is used (there currently isn't an equivalent System V print system checking script).

```
--WARN-- [pcap001w] Print control ‘if' for printer ‘lw2' uses
```

```
        '/usr/local/lib/topsif' which contains '/usr/local/lib'
        which is not owned by root (owned by bin).
```

check_perms

The 'check_perms' script checks the ownership and permissions of system files. A database (specific to the platform) describes which files to check and what the expected permissions are. Files and directories such as '/', '/etc', '/etc/passwd', '/etc/group' and '/etc/aliases' are a few examples of the many files which are checked.

```
--WARN-- [perm019w] The owner of /etc should be root.

--WARN-- [perm019w] /etc should not have group write.

--WARN-- [perm003w] /sbin should not have group write.

--WARN-- [perm003w] /usr should not have group write.
```

check_rhosts

The 'check_rhosts' script examines the .rhosts files in user home directories. It checks the permissions on the files and examines the files, reporting entries with a '+', attempted comment entries, entries that are only partially specified. This is a remote hostname with no remote username. There is no direct security problem here, but it can lead to one as people carry a .rhosts file from site to site.

In addition, it is possible to configure the script to report hostnames that do not match a set of regular expressions. A version of this script written in PERL is provided as well which attempts to verify that the remote user is the same person as the local user (using 'finger').

For large sites, with user home directories distributed across multiple NFS servers, it is possible to configure this script to only check accounts with home directories on local filesytems. This can greatly increase performance. This can also be done for 'check_netrc' and 'check_accounts'.

The '/etc/hosts.equiv' file is also examined. All trusted hosts are reported. Any included netgroups are expanded and reported as well.

```
# Checking accounts from /etc/passwd...

--WARN-- [rcmd004w] User snag's .rhosts file has a '+' for user
         (host ourhost.tamu.edu).

# Checking accounts from NIS...

--WARN-- [rcmd006w] User imauser's .rhosts file has group 'tamug'
         and world read access.

--WARN-- [rcmd006w] User urauser's .rhosts file has group 'onet'
         read access.
```

check_signatures

This script is used to validate system binaries. It does this through the use of a data file which contains digital signatures, generated from distribution media, for important system binaries. There is currently support for two different signature methods: the XEROX Secure Hash Function signatures, commonly referred to as Snefru, and the RSA Data Security Inc., MD5 Message Digest Algorithm. An output block size of 8 is used (256 bits) for the SNEFRU hash. The script auto-detects the type of signature and generates the appropriate one for comparison.

The script reports any system binaries which do not match the stored signatures. It also reports system binaries which are out of date in regards to security patches (using signatures generated from the replacement binaries in the patches). This provides a means of determining quickly whether critical security patches have been installed on a system. A system is being planned which will allow up to date signature databases to be retrieved from a central site(s).

```
--WARN-- [sig004w] None of the following versions of /usr/bin/login
         (-rwsr-xr-x) matched the /usr/bin/login on this machine.
         >>>>>> Sun Patch ID 100630-01
         >>>>>> Sun Patch ID 100631-01
         >>>>>> Sun Patch ID 100633-01
         >>>>>> SunOS 4.1.2 (security patch is 100630)

--WARN-- [sig015w] /usr/bin/mail is from Sun Patch ID 100224-03
         (current is 100224-06)

--WARN-- [sig004w] None of the following versions of /usr/etc/
         rpc.yppasswdd (-rwxr-xr-x)matched the
         /usr/etc/rpc.yppasswdd on this machine.
         >>>>>> SunOS 4.1.2
```

find_files

The ‘find_files' script searches through the file systems and locates files that might present a security problem. These are

- setuid executables
- device files
- symbolic links to system files
- world writable directories
- files with an undefined owner or group
- files with unusual filenames

Setuid files are checked to see if they are scripts, or if they contain relative pathnames (an admittedly crude check). It also compares the list of setuid files against a list prepared from distribution media and reports any new setuid programs.

Any device files found in non-standard locations (i.e., not in /dev) are reported. The script understands diskless clients and will automatically ignore the device directories for these as well. Other directories can also be added to this list by setting a variable in the configuration file.

Any symbolic links to system files (such as /etc/passwd) are reported as well. While not directly a security problem, an ill-placed ‘chown -R' or ‘chmod -R' (or equivalent) could create one.

World writable directories are reported, as they are often used by intruders as a place to store log files. Unowned files are often an indication of a break-in. They also can lead to unexpected access for an account.

The unusual file names includes files with spaces in them, leading '.' or other characters. The list of file names is customizable via a variable which can be set in the configuration file.

check_embedded

The ‘check_embedded' script examines files and extracts any apparent pathnames which are embedded in the files. These pathnames are checked for “proper" ownership and access permissions. These files indicated by these pathnames are then in turn checked for embedded pathnames. This process is continued until no new pathnames are found, or a user specified search depth is reached.

The initial list of files searched comes from two sources. The first is a static list specific to a platform. For example, on SunOS 4.x systems, this would include the /etc/rc.* scripts. The second source is from the other 'tiger' scripts. When the scripts are run from 'tiger' or 'tigercron', they generate lists of files that should be checked. For example, ‘check_path' will request that any executable in root's path be checked, and ‘check_inetd' requests that all the servers defined in /etc/inetd.conf be checked.

```
--WARN-- [embed002w] Path '/usr/sbin/fsck' is not owned by root
          (owned by  bin).
          Embedded references in:
          /sbin/mountall->/etc/init.d/MOUNTFSYS
          /sbin/mountall->/etc/init.d/buildmnttab
          /sbin/mountall->/etc/init.d/nfs.client
```

```
--WARN-- [embed003w] Path '/usr/sbin/ypinit' contains '/usr/sbin'
         which is group 'bin' writable.
         Embedded references in:
         /usr/lib/netsvc/yp/ypbind->/usr/sbin/sysidnet->
         /etc/init.d/sysid.net
```

Miscellaneous checks

In addition to these standard checks, miscellaneous checks specific to a system are also performed. Items such as the use of the ‘securenets' file on SunOS NIS servers and the removal of the "_writers" property for printers on NeXTOS are examples of the checks performed.

```
--WARN-- [misc004w] The PROM monitor is not in secure mode.
```

```
--FAIL-- [misc003f] No /var/yp/securenets file.
```

crack_run

The script ‘crack_run' is used to perform password cracking. No password cracker is provided with the 'tiger' system. It is expected that a tool such as Alec Muffett's 'Crack' will be used. ‘crack_run' collects all password sources and runs the password cracking tool on them, and reports the results. Since this can take days to complete (or longer), 'tiger' by default does not wait for this to complete.

```
--WARN-- [crk001w] The following login id's have weak passwords:
          imauser urauser
```

Isn't this just COPS?

One common question is how does 'tiger' compare to the COPS package by Dan Farmer There is a lot of overlap between the two packages. Much of this is intentional. Dan Farmer allowed us to borrow ideas and code from his COPS package. We are giving Dan (and anyone else) the same ability in regards to 'tiger'.

There are advantages and disadvantages between the two packages. Note that most of this is subjective and hence will no doubt be biased.

We feel that 'tiger' is easier to use than COPS for a person who is not familiar with systems administration. As stated earlier, once unpacked, all that is necessary is to run 'tiger' and a report will be generated. The explain facility allows the administrator to get a better idea about the entries in the report. One area that 'tiger' is lacking in is user documentation.

Another area we feel 'tiger' is at an advantage is based on part of the design structure, in which no system files are referenced directly. This feature is what allows the auto-checking of multiple information sources (for example /etc/passwd, NIS, NetInfo, etc) without having to alter the scripts which perform the checks. This makes running the scripts easier for the administrator.

There are other areas that we feel 'tiger' is better than COPS. These include the use of digital signatures for checking for security patches, the complete checking of pathnames containing symbolic links, and the more thorough examination of system configuration files.

We recognize that COPS does have advantages over 'tiger'. The primary one is that COPS is a "proven product". It has been used by thousands for two to three years now. Also, though 'tiger' will attempt to run on platforms for which no configuration files exist, COPS is more likely to succeed in running. The use of newer shell features, primarily shell functions, prevents the use of 'tiger' on any UNIX system which has an older Bourne shell. There are also miscellaneous checks performed by COPS that have not been integrated into 'tiger'. COPS also includes the 'kuang' expert system checker. There is no equivalent functionality in 'tiger'.

7. Observations

We have been using this combination of filtering, monitoring, and checking almost a year, with very positive results. While our network monitoring tools continue to show incoming intrusion attempts (unfortunately along with some outgoing attempts), we have had no major incidents of the type we experienced last summer. The combination of approaches seems to have struck an appropriate balance between security and availability for our academic environment.

Our monitoring tools have produced some interesting statistics. During the last four weeks, for example, we have observed the following number of incoming security events:

e-mail forgery	6
knob-turning	48
TCP attacks/events (X11, DNS zones, rshell ...)	30
UDP attacks/events (TFTP, SNMP ...)	7

Of these 91 incidents, 49 (or 54%) originated from .edu sites. Edu sites account, however, for only 30% of all internet hosts (according to the July 1993 Internet Domain Survey). This means that a minority of hosts are accounting for a disproportionate majority of intrusion activity. One would hope that other university networking groups would be actively trying to reduce these incidents, yet of the requests for the etherscan tool, less than 25% have come from university sites.

8. Conclusions

A set of policies and tools for filtering, monitoring and checking has been developed in response to a significant series of intrusions from internet. Each of these three areas has proved critical: the filtering for its ability to protect machines from attack, the monitoring because it augments the filter and has yielded significant information about the intruders and their methods, and the checking tools for their ability to automate the task of checking and cleaning a large number of machines. With these tools and associated policies, we have achieved an appropriate balance between security and availability in an academic environment.

9. Availability

Drawbridge, the *tiger scripts*, and all monitoring tools other than *etherscan* are now available via anonymous ftp in sc.tamu.edu:pub/security/TAMU. Due to export restrictions, the DES routines used in *drawbridge* have been put in a separate tar file and are readable only by U.S.A. sites. Other sites should have no problem either running the filter without encryption or dropping in their own favorite encryption package.

The distribution of *etherscan* has been hotly debated within the TAMUSC group. One argument is that *etherscan* should be freely released, as the crackers already have equivalent knowledge and tools (they do) and restrictions would only hurt valid administrators. The counter argument is that free availability of the intrusion signatures would enable the crackers to design better intrusions and the availability of sources would provide novice crackers a significant help. Our resultant compromise will be to provide copies to Network Information Center registered site contacts, given an official request on respective letterhead. Requests should be sent to:

Dr. Dave Safford, Director
Supercomputer Center
Texas A&M University
MS 3363
College Station, TX 77843-3363

10.References

[1] D.B. Chapman. Network (In)Security through IP Packet Filtering, *Proceedings of the Third UNIX Security Symposium*, September 1992.
(available from ftp.greatcircle.com as pub/pkt_filtering.ps.Z)

[2] Ranum. "Thinking about Firewalls", available on the Internet

[3] Violino, Bob. "Are Your Networks Secure?" Information Week, April 12, 1993, page 30.