Interested in learning
more about security?

SANS Institute
InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

Security Issues in Running an Email

This paper discusses security topics with respect to administering an email system. It starts discussing system hardening (CIS security benchmarks, disabling services, TCP wrappers, Tripwire, logging, etc.) from the perspective of an email system sysadmin. Then it discusses anti-virus software and why quarantining, cleansing, notifying are the wrong approach. Instead, messages containing viruses should be rejected during the SMTP protocol. It details how the SMTP protocol works and how a sendmail mail filter ("milter")...

Copyright SANS Institute

Author Retains Full Rights

AD

Security Issues in Running an Email Server

Jerry Berkman, May 30, 2003

GIAC Security Essentials Certification 4.1b, Option 1

Abstract

This paper discusses security topics with respect to administering an email system. It starts discussing system hardening (CIS security benchmarks, disabling services, TCP wrappers, Tripwire, logging, etc.) from the perspective of an email system sysadmin.

Then it discusses anti-virus software and why quarantining, cleansing, notifying are the wrong approach. Instead, messages containing viruses should be rejected during the SMTP protocol. It details how the SMTP protocol works and how a sendmail mail filter ("milter") can be used to reject messages containing viruses. The milter is included as an appendix. The last section discusses quotas on mail accounts and why blocking rather than queuing/retrying is the best policy, both for the user, the system, and for security.

Introduction

I've have been running a university email system for about 10 years. The system has over 40,000 faculty, staff, student, visitor, and departmental accounts, and processes about 400,000 messages per day. This paper covers security issues involved in running an email server.

Securing your Host and OS

This paper focuses mainly on security issues with respect to email servers; however, first it will discuss best practices for host security, especially as they relate to email servers. This section will discuss a few of the general tools and concepts, including:

- The CIS security benchmarks and scoring tools
- Enabling/disabling services
- Secure sysadmin access

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

- TCP wrappers
- Tripwire
- Logging
- Policy

Center for Internet Security benchmarks and tools

The Center for Internet Security (CIS) has developed security benchmarks and scoring tools for the Windows, HP UX, Linux, and Solaris operating systems. The benchmarks describe appropriate security settings to harden your system. The scoring tools check your system against the benchmarks and tell you what you might consider changing. The tools are read-only; they do not make any changes to the system. Even if there is not a tool for the platform you are using, it is useful to download and read a benchmark for a similar system. The benchmark documents and scoring tools are available for free download from the Center's web site, http://www.cisecurity.com.

Enabling/disabling services

It is very important to turn off all but essential services. Services may be run via *inetd*, it is a super daemon listening on a number of different ports for connections. When a connection request is received on a port *inetd* is listening to, it forks and execs the appropriate daemon process. *xinetd* is an open source version of *inetd* with many additional features; for example, *xinetd* allows rate throttling of connections by a remote host. *xinetd* is available from http://www.xinetd.org.

Services not run via *inetd* generally run with one process per service, with each process listening on a separate port. These are started on system startup by "rc" scripts, so called because they live in the file */etc/rc* or in directories */sbin/rc{1,2,3}.d*.

For an email server, the basic public services are POP, IMAP, and SMTP. How these run is determined by the specific software you are using. One POP daemon implementation may need to be run via *inetd* or *xinetd*, another may run as a separate daemon, and others may be able to be run either way. You may need to run a few other daemons, such as a daemon used by the backup system, but probably not many. From time to time, there have been vulnerabilities reported on just about every service run by *inetd* or *xinetd*. Make sure to remove the telnet, ftp, shell, login, and exec services that normally run via *inetd*. And turn off NFS, X servers, etc. Check the *inetd* or *xinetd* configuration files and the system startup scripts and eliminate anything else that is not required.

Secure sysadmin access

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

For system administration, use *ssh* and *scp* instead of *telnet* and *ftp*. These avoid problems with sniffers and man-in-the-middle attacks. Try to keep the number of people with root access small. If people want root access for only a few specific tasks, use *sudo*,[1] an open source tool, to give them privileged access for just those tasks.

TCP wrappers

The TCP wrappers allow you to control access by protocol. The wrappers were written by Wietse Venema, and are available for free download at http://www.porcupine.org/. They are often used with *inetd*. Then when a connection is made, *inetd* invokes the TCP wrappers module, which applies the access rules, logs the connection, and then, if everything is ok, invokes the appropriate application. The rules are contained in */etc/hosts.allow* and */etc/hosts.deny.* For example, the following rules in */etc/hosts.deny*

> telnetd:ALL
> pop3:santa.northpole.org

would deny access to *telnetd* for everyone and deny access to *pop3* from santa.northpole.org. The control files can also specify commands to be run. We once had a remote host fingering our system every few seconds. I tried blocking it by adding "fingerd:badguy.domain.edu" to */etc/hosts.deny*. This prevented that host from ever accessing *fingerd*, but now the remote host was trying to connect about five times as often. Evidently the remote host was in a loop fingering with no pause. So I modified the line to be "fingerd:badguy.domain.edu:sleep 3600". Then when the connection was made, a TCP wrapper process was forked and execed, and it slept for an hour before denying access to the finger application. The remote system was still looping, but at such a low rate it didn't matter.

You could use the same technique to slow down a user who is popping excessively. Since you can specify any command in the configuration file, you could write a program to monitor how often hosts are popping, and selectively delay excessive poppers.

Integrity Checkers

An integrity checker can be used to check important files for changes and report the changes to the system administrators. The best known integrity checker is Tripwire, developed by Dr. Eugene Spafford and Gene Kim at Purdue University. To use Tripwire, set up a configuration file specifying a list of files or directories to check. When Tripwire runs, it computes and save hashes of the files using one or more of the following hash algorithms: [2]

- CRC-32, POSIX 1003.2 compliant 32-bit Cyclic Redundancy Check
- MD5, the RSA Data Security, Inc.® Message Digest Algorithm

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

- SHA, part of the SHS/SHA algorithm
- HAVAL, a strong 128-bit signature algorithm

It is best to make an initial run of Tripwire before the host is put on the network and save the results, preferably offline. If your system is ever compromised, you can use Tripwire to identify which files have been compromised.

It can also help in other ways. For example, once when we were not running Tripwire, one of our sysadmins edited */etc/hosts* on one server when he thought he was editing it on a different server. This had subtle side effects and it took several days before we figured out what had happened. Another time, after a system crash, we noticed several files had been corrupted on the root and */usr* file systems. Tripwire helps identify which files have been changed or corrupted.

There are several different versions of Tripwire:

- the commercial version and a corresponding console for use on multiple systems[3]
- the original Purdue version, available to educational institutions[4]
- an open source version[5]

There is a comparison between the original and commercial version,[6] but not between the open source and commercial version. More information is available at http://www.tripwire.com.

Logging

Logging is an important tool to use for investigating intrusions, measuring usage, and capacity planning. There are several types of logging. Many applications and system processes log via the *syslog* facility. This is good because then the routing and handling of messages sent to *syslog* are controlled by the specification in the *syslog* config file, */etc/syslog.conf*. This can be changed without changing the program.

The entries in the *syslog* config file specify routing according to "facilities" and "priorities". Syslog "facilities" include: *user, kern, daemon, mail, auth, security, local1, local2*, etc. Syslog "priorities" are, in decreasing order: *emerg, alert, crit, error, warn, notice, info*, and *debug*.

Test how *syslog* works; make sure the log records are being written where you want them. Note the CIS benchmarks for Linux and Solaris[7] warn that those systems by default may not capture "auth" logging. It is easy to test and to fix. The *logger* command may be used to test how *syslog* is handling messages, e.g.:

% logger -i -p mail.warn -t test this is a test of *syslog*

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

This produces a *syslog* line of the form:

May 27 23:18:08 myhost test[308359]: this is a test of syslog

Be careful when testing this; many sysadmins have configured log watchers such as *swatch*[8] to beep them if unexpected lines appear in the log.

Process accounting, showing every command executed on the system, at what time, and what resources were used, does not use *syslog*. Instead, the kernel writes these records to the file */var/adm/pacct* or */var/adm/acct/pacct*. On some systems, accounting has to be specified when making a kernel. Then it needs to be started during system startup by the *accton* command. Some people worry about the resources taken by process accounting. However, the resources used are minimal, adding 80-100 bytes to the acct file for each command executed. The accounting data may be viewed by the *acctcom* command or the *lastcomm* command, depending on the platform:

```
% acctcom  | head

ACCOUNTING RECORDS FROM:  Mon May 26 23:55:00 2003 PDT
COMMAND                       START    END          REAL      CPU
MEAN
NAME       USER     TTYNAME  TIME     TIME       (SECS)   (SECS)
SIZE(K)
#accton    root     ?        23:55:00 23:55:00     0.03     0.01
0.00
pop3d      sms      ?        23:55:00 23:55:00     0.17     0.03
224.00
#sendmail  root     ?        23:55:00 23:55:00     0.11     0.09
424.00
#sendmail  root     ?        23:55:00 23:55:00     0.05     0.02
1288.00
#mv        root     ?        23:55:00 23:55:00     0.08     0.01
0.00
#sendmail  root     ?        23:55:00 23:55:00     0.09     0.09
416.00
%
```

acctcom has many options, e.g. "-t" to show system and user time separately, "-n string" to only display commands including the specified string, etc. Similarly, */var/adm/wtmp* or */var/adm/wtmpx* are binary files containing information on login sessions, *ftp* sessions, etc. The information is displayed with the *last* command.

Some application do there own logging, to files built into the application or specified in the applications configuration file, for example, the Apache web server logs.

It is a good idea to collect logs both locally and on a separate logging host. Having a separate, central logging host helps out in several ways. First, hackers

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

may cover their tracks by modifying or deleting logs. This is much harder if a second copy is on another system. Second, you can analyze the logs without worrying about the performance impact on the host being logged. You can also merge logs from several hosts to look for patterns in security information and to make sitewide summaries. Note that *syslog* uses UDP between hosts; if network traffic is very heavy, some records may be dropped.

It is easy to save *syslog* records in two places, as you can specify this in the *syslog* configuration file. With logs which are human readable but not routed by *syslog*, you can use "tail -f log" and pipe it to a script which writes it to *syslog*. This doesn't work for the accounting records or wtmp, as these are binary files.

Policy

It is important to have policies in place, especially an Appropriate Use Policy (AUP) for users to agree to. One source of policies, especially for universities, is the Educause, Cornell, Institute for Computer Policy and Law web site[9]. This has AUPs plus many other types of policies.

I also recommend having a written agreement with staff who have special privilege access describing what they can and can not do. See for example, UC Berkeley's "Model Privileged Access Agreement", [10] The agreement need not be very detailed; sitting down with staff and discussing it for a few minutes before signing clarifies rights and responsibilities.

The Message store

There are three main types of message stores:

- Message stores which store all the mail for a folder or inbox in a single file; this is known as "mbox" format. An example is traditional UNIX mail boxes in */var/spool/mail*.
- Message stores which use a separate directory for each folder, storing the metadata for the folder in a few files and each message in a separate file; this is known as "maildir" format. An example is the Cyrus IMAP server[11].
- Message stores which store all the information, messages, folders, metadata, etc., in a large data base. An example is Oracle Email.[12]

The type of mailstore has implications for security. The mbox format requires a complete rewrite of the folder whenever there is a change, even if the change is just changing the status of whether a message has been read or not. By contrast in the maildir format, messages are written to a file and that file is never changed thereafter. The corresponding metadata is written to a file which is continually updated, but the messages themselves are write-once, read-only.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

We used to offer POP using the Qpopper server[13] which used the mbox format. When there was a system crash, many spools would be in the process of being rewritten. Inevitably, we would find a few that had been corrupted or contained someone else's messages. If the file system filled, a different type of corruption would occur; namely some spools would start with blocks of null bytes. We have not had corruption since we switched to a system using the maildir format, which writes messages only once and never rewrites them.

Filename Attacks

If you are working inside an IMAP store in which folder names are used as directory names, be careful of Trojan Horses. Suppose a user has created a folder named `"; rm -r * ;".` This is legal as an IMAP folder name and as a UNIX directory name. However, in some cases, UNIX commands which are normally safe may now try to erase your mailstore. For a test of whether your system is susceptible to this type of attack, see "Filename attacks", http://www.soldierx.com/books/networking/puis/ch11_05.htm.

Anti-Virus Defenses

There are many commercial A/V (anti-virus) packages on the market. They generally give you several options for handling messages with viruses, for example:

- Send the message and virus along with a warning to the recipient(s),
- Send the message after deleting and quarantining the virus
- Send the message after deleting the attachment containing the virus
- Send the message after cleaning the virus from the attachment

The A/V packages also allow you to send a notification to the sender, recipient(s), system administrator, a combination of the above, or none of the above.

However, most viruses in email now are sent by SMTP engines contained within the virus itself and have forged "from" addresses. On the day that WORM_SOBIG.B appeared, ten thousand accounts on our server received fifty thousand copies of the worm, supposedly from support@microsoft.com, in a few hours. The body of the message consists of the statement "All information is in the attached file." plus the worm in the attachment.

None of the standard A/V options seem appropriate. Sending anything to the envelope recipients just wastes system and network resources and the recipients' time. There is no point in quarantining 50,000 copies of the virus. Sending 50,000 notifications to support@microsoft.com would have contributed to a DOS (denial of service) attack on Microsoft.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

The only thing that makes sense is to block these messages. Before looking at how to block messages containing viruses, let's review how messages are transmitted between hosts and what obligations the sender and receiver host have.

The SMTP Protocol

Messages are sent from one host to another via the SMTP (Simple Mail Transfer Protocol) which is defined in RFC 2821[14]. Here is a sample SMTP protocol session, copied from RFC 2821, p. 72. Lines sent by the sending host to the receiving host are labeled with "S:"; lines sent by the receiving host to the sending host are labeled with "R:".

```
   This SMTP example shows mail sent by Smith at host bar.com, to
Jones,
   Green, and Brown at host foo.com.  Here we assume that host bar.com
   contacts host foo.com directly.  The mail is accepted for Jones and
   Brown.  Green does not have a mailbox at host foo.com.

      S: 220 foo.com Simple Mail Transfer Service Ready
      C: EHLO bar.com

      S: 250-foo.com greets bar.com
      S: 250-8BITMIME
      S: 250-SIZE
      S: 250-DSN
      S: 250 HELP

      C: MAIL FROM: <Smith@bar.com>
      S: 250 OK

      C: RCPT TO: <Jones@foo.com>
      S: 250 OK

      C: RCPT TO: <Green@foo.com>
      S: 550 No such user here

      C: RCPT TO: <Brown@foo.com>
      S: 250 OK

      C: DATA
      S: 354 Start mail input; end with  <CRLF>. <CRLF>

      C: Blah blah blah...
      C: ...etc. etc. etc.
      C: .
      S: 250 OK

      C: QUIT
      S: 221 foo.com Service closing transmission channel
```

According to RFC 1123, when the receiver sends a "250 OK" in response to "DATA", then the receiving host has accepted responsibility for receiving or bouncing the message: [15]

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

> 5.3.3 Reliable Mail Receipt
>
> When the receiver-SMTP accepts a piece of mail (by sending a "250 OK" message in response to DATA), it is accepting responsibility for delivering or relaying the message. It must take this responsibility seriously, i.e., it MUST NOT lose the message for frivolous reasons, e.g., because the host later crashes or because of a predictable resource shortage.
>
> If there is a delivery failure after acceptance of a message, the receiver-SMTP MUST formulate and mail a notification message.

Thus it is not appropriate to simple discard messages containing viruses. However, it is ok to not accept the message; instead of replying "250 OK", reply with a permanent error message.

SMTP reply messages

RFC 821, the original SMTP RFC, specifies SMTP return messages have two parts, a three digit reply code followed by a text message.[16] The reply code's first digit specifies:[17]

- 2yz Positive Completion reply
- 3yz Positive Intermediate reply
- 4yz Transient Negative Completion reply
- 5yz Permanent Negative Completion reply

The second and third digits provide more detail. However, as the internet and email evolved, it became clear the reply code categories were not sufficient. RFC 1893 defined "enhanced system status codes" of the form x.y.z, which follow the RFC 821 reply codes.[18] For a virus, an appropriate response would be "550 5.7.0 VIRUS FOUND", where "550" is the RFC 821 reply code defined as "Requested action not taken: mailbox unavailable (e.g., mailbox not found, no access, or command rejected for policy reasons)" and "5.7.0" is the RFC 1893 status code for a permanent error of type "Other or undefined security status".

Viruses - to notify and deliver or to reject

Upon recognizing a virus during the SMTP protocol, instead of accepting the message and sending it on with or without the virus and with or without notification messages, it makes more sense to refuse to accept the message.

Then, since the receiving SMTP server has not accepted the message, it has no responsibility with respect to delivery or notification. If the sending host is a server following RFC 1193, it will compose a Mailer-Daemon message to send to the sender address. If it is a desktop client, such as Eudora, it will hopefully notify

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

the sender via a pop up window or some other means. If the message is being sent by a virus containing an SMTP engine, then the response will probably be ignored.

The KLEZ worm propagates by creating an email message including both the worm and a file from the victim's hard drive and the worm.[19] It forges the from address by using a random address found on the infected computer. Thus sending notification to the supposed sender only serves to confuse an innocent third party. If the A/V application cleans the virus attachment or deletes it, and then sends the mail, then a file from the victim's hard drive is being sent without the victim's knowledge to another user. This is a violation of privacy and confidentiality. Clearly, the best practice is to block any instance of KLEZ during the SMTP protocol.

MX records

Several A/V vendors have told me MX records can be used to route all mail to a gateway running an A/V system. However, MX records are only used for routing by mail relays; they are not used when mail is sent from desktop clients such as Eudora or Outlook. Recent viruses include their own SMTP engines, and may rarely, if ever, use MX records. So don't rely on MX records as part of an A/V solution.

Sendmail Milters

Some A/V products do have options to block during the SMTP protocol. However, they often are not the best SMTP servers so it may not be a good idea to use them as the SMTP gateway. Another option is to use an API during the SMTP protocol to check for viruses. The open source Sendmail MTA, available from http://www.sendmail.org, has included such an API for several years.[20] This API is named "milters" (short for "mail filters"). There is a web site, http://www.milters.org/, which serves as a clearing house for information on milters.

The Sendmail Milter interface is defined as a C callable library, to be integrated into your filter program. Although the sendmail source includes just a C language binding, others have written open source implementations for other languages, such as the Sendmail::Milter module in the Comprehensive Perl Archive Network (CPAN)[21]

The milter API allows you to specify callouts after:

- the initial connection
- the HELO
- the MAIL FROM specifying the envelope sender
- the RCPT TO specifying an envelope recipient

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

- each header line
- the end of the header
- each 64 KB of the message body
- the end of the message

This allows milters to "exercise fine-grained control at the SMTP level",[22] These entry points are optional; only specify in the program those you will use. When it is done, each callout routine returns control to sendmail passing one of the following codes:

- SMFIS_CONTINUE
- SMFIS_ACCEPT
- SMFIS_REJECT
- SMFIS_TEMPFAIL
- SMFIS_DISCARD

SMFIS_CONTINUE means to continue processing the message including calling the milter callouts. The others have slightly different meanings depending on context. For example, SMFIS_REJECT after a RCPT TO causes sendmail to reject that recipient with a 5xx SMTP return code and continue. Subsequent recipients may be accepted. In the other callouts, SMFIS_REJECT means to reject and close the connection or reject the current message. See the documentation for full details.[23]

Milters run as threaded applications. This means there is one daemon process executing for each milter, and that that process processes all the messages received by sendmail, and may be processing more than one at a time. This saves resources by not forking and execing multiple processes. If your milter is very resource intensive, you may want to have it fork and exec independent processes to do the work, but that is not needed for simple milters. For example, here is the cpu time used by four milters currently running on our system. The cpu time used is for processing about 1,000,000 messages in the past four days:

Milter name	cpu used	purpose
virus blocker	165 min	block messages containing common viruses
check quota	126 min	block message to recipient with full inbox
message logger	15 min	save copies of messages (currently idling)
spam blocker	16 min	spam blocker, for selected users

The heaviest cpu user is the milter which identify messages with viruses. It does a simple string search for about a dozen frequently occurring viruses, and is described in a later section. It used about 2.8% of a cpu and identified over 10,000 messages containing viruses, which sendmail then rejected.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Second is a milter which blocks mail to accounts which are already at their quota. After each RCPT TO command, it checks if the recipient is local, is not forwarding or using vacation, if the user's is within 25 KB of their quota, etc. If so, that recipient is rejected with SMFIS_REJECT. Then sendmail sends a 5xx reply to the sending host, and that host may choose to continue with additional recipients or quit. The milter returns SMFIS_ACCEPT when it receives the first header line, and sendmail continues and the message is sent to any recipients not already rejected. The milter does not specify routines for and does not process the body of the message.

The third is a milter which can save copies of all or part of the incoming message stream. It checks for the existence of a certain file when it is invoked, and if the file exists, creates a copy of the current message, otherwise it returns SMFIS_ACCEPT right after the envelope from. It is currently idle and using little cpu. It is easier and probably safer to have a switch and let the milter run all the time, rather than changing sendmail's configuration file, and restarting it. One use for this milter is to gather messages for testing, e.g. a new anti-virus program. Another is to use it to collect an individual's mail in response to a search warrant.

The last milter blocks spam by checking the "From " line for a list of frequent spammers. This is being tested by a few users and uses little cpu. It only looks at the envelope recipient and the header "From " line.

While simple, low resource milters can be written in C and executed on the SMTP server, more resource intensive milters are likely to be written in a scripting language, such as Perl, and run on another host.

Coding C Milters

This section will discuss a few of the problems and techniques in coding milters. This is not a complete discussion; just highlighting some points that may not be immediately clear from the milter documentation. The source code for the milter which blocks messages containing viruses in included in Appendix 1. It is based on the sample milter distributed with sendmail.[24]

Milters are threaded and this introduces some complexity to coding. Instead of forking a new process for each message being processed, one milter process handles them all, many simultaneously. It allocates a separate, private data space for each message. This private data space is a structure. In Appendix 1, the structure is declared by:

```
struct mlfiPriv
{
        char        mlfi_env_from[100];
        char        mlfi_virus[100];
};
```

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
#define MLFIPRIV    ((struct mlfiPriv *) smfi_getpriv(ctx))
```

The private space in Appendix 1 is allocated in the env_from() callout, while processing the envelope sender. Then it is accessed in each routine by:

```
        struct mlfiPriv *priv = MLFIPRIV;
```

The milter library keeps track of the private space; you don't need to worry about it as long as you keep the basic structure shown in Appendix 1 or sample milter distributed with sendmail. You must put all the variables for a message in the private data space for that message.

You must also make sure your milter always frees the storage when it terminates, whether it accepts the message or rejects it. Remember that accepting or rejecting will end processing of a message, but the milter process lives on. Check with the *ps(1)* command to see if the milter memory usage is growing to see if there is a memory leak.

Another issue is how to create temporaries. Often the process id is used in creating temporaries. However, the milter process lives on without changing process id, so the process id can not be used for this purpose. On most systems, *mkstemp(3)* provides a safe way to create unique files.

The milter will process thousands or millions of messages, so you have to make sure you close all files; otherwise the milter will exceed the limit on open files.

Make sure any procedures used are thread safe. For example, you could try via a milter to slow reception of mail from an ill-behaved site by executing calls to *sleep* when mail is being received from that site. However, if *sleep* is not thread safe, the sleep will stop the milter and sendmail, and processing of all messages will be suspended, not just those from the offending site.

The Virus Detecting Milter

The virus blocking milter is fairly simple. When it is first started up, it reads in a set of virus signatures from the file */etc/mail/milters/signatures*:

```
WORM_YAHA.K  =
DUxhuml30nenGwkz6SxOb2GMamANTGfHdhe2WDEx83Qb1i0WLBgMtqM3pQG4Gi0AiSYCMuF
dYleq
WORM_SOBIG.B =
uqZpmpiQhHhwL2RpmqZpYFxYVFCmaZqmTEhEQDyapmmaODQsIBgQs2y6QQgDAPhI7E3TNE3
k2MzI
...
```

It stores the names in the *names[]* array and the patterns in the *char patterns[]* array. Then in the callout for the body, it looks for any of the signatures. In simplified form:

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
for( i = 0; i <= n_sig; i++ ) {
        if( ptr = strstr( (char *)bodyp, patterns[i] ) ) {
                /* found a virus signature */
                        ...
                cleanup ...
                return SMFIS_REJECT;
        }
}
```

For full details, see the virus blocking milter source, which is in Appendix 1. In the last two weeks, the milter has blocked the following viruses:

Number blocked	Virus Name
28056	WORM_KLEZ.H
9504	WORM_SOBIG.B
3159	WORM_YAHA.P
1739	WORM_YAHA.G
1465	WORM_YAHA.K
688	WORM_FIZZER.A
516	WORM_SIRCAM.A
207	other

I configured this milter into the configuration for the sendmail listening on port 25. This allows us to reject messages containing viruses during the initial SMTP protocol. After messages are accepted, they are then passed via SMTP to our commercial A/V package. The milter averages blocking about 3000 messages per day. The A/V package then deletes viruses from another 30 or so messages per day, and then sends the messages to their destinations. Our volume is about 400,000 messages per day; so a little less than 1% contain viruses, about 99% of those are blocked by the milter, and about 1% have the virus removed by the A/V package before delivery.

Just like any other application, a milter may fail. It may fail for a particular message and continue processing other messages, or it may abort, for example, if it were started with a cpu time limit in effect. Milters may be configured in the sendmail configuration file so that if the milter is absent or fails to complete:

- sendmail continues with normally processing of the message (the default)
- sendmail send a 4xx transient failure to the sending host
- sendmail send a 5xx permanent failure to the sending host

The virus identification milter it is configured to send a 4xx failure if the milter does not complete. Our other milters are configured with the default: continue processing even if the milter doesn't exist or doesn't complete. Milters are

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

configured either as UNIX sockets on the local system or as TCP sockets on remote systems.

Gettting Signatures

It is best to block messages containing viruses via the milter, rather than having them go through the A/V software to the recipient, albeit without the virus. So when the A/V software starts catching a significant number of any particular virus, I try to get a signature for it and add it to the signature file. First I activate the message logging milter. It captures the entire message stream. Then I monitor the A/V software log waiting until it has recorded several instances of the virus, and then I isolate the messages containing the virus using the information from the A/V log. Then I search for common lines via:

```
% cat msg* | sort | uniq -c | more
```

Then I pick out a line which appears in all the copies and add it to our signature file. I try to use a line which looks completely random; the odds of an accidental match with a non-virus carrying message are infinitesimal. Our signature file currently has only 23 lines. This is amazing since there are tens of thousands of viruses, but we get just a few distinct viruses. Sometimes there are more than one versions of the virus and I add any line which looks like it will help. Note I am adding lines from the original message in MIME form, not the de-mimed virus. I am trying to figure out a way to use our A/V software directly in a milter, but haven't had time to do it yet. Our A/V software will quarantine viruses, but that doesn't help in developing signatures for the milter because it quarantines the de-mimed version.

Quotas

Quotas on mail folders are a necessary evil. Without quotas, users will use as much storage as they can, filling your disks. I once did a survey and could not find any consistency in how sites apply quotas. Some sites have "quotas" but do not really enforce them. Others grant larger quotas to just about anyone who asks. One site imposes quotas only on users who have not recently checked their mail, because they have probably abandoned their accounts. We have users who have reached their quota, 25 MB, even though they have never checked their mail, not even once! Apparently they signed up for the account, subscribed to mailing lists, and forgot about the account.

The first quota issue is what does the system do with a message which will not fit within the account's quota. Many systems deliver it, putting the user over quota, and then bounce all subsequent mail to that account. We often see the case where user A sends a series of multi-megabyte messages (MP3s or spreadsheets or PowerPoint presentations) to user B. If these are delivered, user

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

B will be over quota, and will not receive mail, even 5 KB or 10 KB messages. I think it is much better service to bounce the large message and deliver the smaller messages up to the quota limit.

If the system doesn't deliver a message which would put the account over quota, the system can:

- Save the message in the mail queue and retry periodically until there is room to deliver it, even if it takes months or years
- Save the message in the mail queue and retry periodically to deliver it, but warn the sender and bounce after a set time
- Bounce the mail; do not queue and retry

I think queuing and retrying forever is ridiculous; I wouldn't mention it except there a vendor that does just that.

Queuing the message to retry is reasonable if you have day to day contact with your users, such as in a small department. However for an ISP or for a large university mail server, the problem is mostly spam to inactive accounts, and queuing and retrying just wastes resources.

That can also lead to problems. We had a faculty member who was in Washington to give a grant proposal to the National Science Foundation. He needed some data and had his assistant email it to him. But he couldn't find it in his inbox. It wouldn't fit within his quota and had been queued for later delivery attempts, but neither the professor nor his assistant were notified by the email system.

In another example, a user was subject to a denial of service attack. He was sent 20000 messages. The system delivered about 5000 before hitting the accounts quota, and then queued the rest. When the user deleted the first 5000 messages, the next 5000 were delivered, and so on.

Recently we added the quota checking milter. It checks during the SMTP protocol after each RCPT TO command:

- whether the recipient a local account
- whether the recipient account is within 25 KB of its quota
- whehter the sender is someone other than postmaster or mailer-daemon
- whether the recipient is saving mail locally and not forwarding it or sending an automatic response

If these are all true, the recipient address is rejected with a "552 5.2.2 Mailbox full" message. Most of these are spam, and this saves our postmaster account about 500-1000 messages/day.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Endnotes

1. Miller, Todd C. "Sudo Main Page". 2003. URL: http://www.courtesan.com/sudo/ (May 30, 2003)
2. URL: http://www.tripwire.com/products/product_content/tfs_functional.cfm (May 30, 2003)
3. Tripwire. "Tripwire for Servers". URL: http://www.tripwire.com/products/servers/ (May 30, 2003)
4. Tripwire. "Tripwire Academic Source Release". URL: http://www.tripwire.com/products/tripwire_asr/ (May 30, 2003)
5. Source Forge Tripwire Project. "Project: Tripwire: Summary". URL: http://sourceforge.net/projects/tripwire (May 30, 2003)
6. Tripwire. "Version Comparison". URL: http://www.tripwire.com/products/tripwire_asr/compare.cfm (May 30, 2003)
7. Go to URL: http://www.cisecurity.com to download the benchmarks (May 30, 2003) (note - URL sometimes refuses connections; keep trying)
8. Source Forge Swatch Project. "Project: Swatch: File List ". URL: http://sourceforge.net/project/showfiles.php?group_id=68627 (May 30, 2003)
9. Educause, "The Educause, Cornell, Institute for Computer Policy and Law" web site, http://www.educause.edu/icpl/policies.asp? (May 30, 2003)
10. UC Berkeley, "Model Privileged Access Agreement", http://itpolicy.Berkeley.EDU:7015/proceds/access.html (May 30, 2003)
11. Project Cyrus. "Cyrus IMAP Server". URL: http://asg.web.cmu.edu/cyrus/imapd/ (May 30, 2003)
12. Oracle. "Oracle Collaboration Suite: Oracle Email". URL: http://otn.oracle.com/products/oemail/email_fo.html (May 30, 2003)
13. Qualcomm. "Qpopper". URL: http://www.eudora.com/qpopper/ (May 30, 2003)
14. Klensin, J. "RFC 2821 - Simple Mail Transfer Protocol". April 2001. URL: http://ietf.org/rfc/rfc2821.txt (May 30, 2003)
15. Braden, R. "Requirements for Internet Hosts -- Application and Support". URL: http://ietf.org/rfc/rfc1123.txt October 1989 p. 63 (May 30, 2003)
16. Postel, Jonathan B. "RFC 821 - Simple Mail Transfer Protocol". August 1982. URL: http://ietf.org/rfc/rfc821.txt section 4.2.3, p. 45 (May 30, 2003)
17. ibid, Appendix E, p. 48. See also Klensin, J. "RFC 2821 - Simple Mail Transfer Protocol". April 2001. URL: http://ietf.org/rfc/rfc2821.txt (May 30, 2003)
18. Vaudreuil, G. "RFC 1893 - Enhanced Mail System Status Codes" January 1996. URL: http://ietf.org/rfc/rfc1893.txt (May 30, 2003)
19. Hindocha, Neal. "W32.Klez.H@mm". Symantec's Virus Encyclopedia. URL: http://securityresponse1.symantec.com/sarc/sarc.nsf/html/w32.klez.h@mm.html (May 30, 2003)
20. Milters were added to Sendmail in version 8.10.1, April 6, 2000 (May 30, 2003)
21. Ying, Charles. "Sendmail::Milter". URL: http://search.cpan.org/author/CYING/Sendmail-Milter-0.18/Milter.pm (May 30, 2003)

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

22. Sendmail Inc. "Architecture". Filtering Mail with Sendmail. http://www.sendmail.com/partner/resources/development/milter_api/design.html. (May 30, 2003)
23. Sendmail Inc. "Filtering Mail with Sendmail", http://www.sendmail.com/partner/resources/development/milter_api/ (May 30, 2003)
24. Sendmail. ftp://ftp.sendmail.org/pub/sendmail/sendmail-current.tar.Z (May 30, 2003)

References

Braden, R. "Requirements for Internet Hosts -- Application and Support". URL: http://ietf.org/rfc/rfc1123.txt October 1989 (May 30, 2003)

Center for Internet Security. "Center for Internet Security". URL: http://www.cisecurity.com (May 30, 2003) (note - URL sometimes refuses connections; keep trying)

"The Educause, Cornell, Institute for Computer Policy and Law web site, http://www.educause.edu/icpl/policies.asp?

Hindocha, Neal. "W32.Klez.H@mm". Symantec's Virus Encyclopedia. URL: http://securityresponse1.symantec.com/sarc/sarc.nsf/html/w32.klez.h@mm.html (May 30, 2003)

Klensin, J. "RFC 2821 - Simple Mail Transfer Protocol". April 2001. URL: http://ietf.org/rfc/rfc2821.txt (May 30, 2003)

Miller, Todd C. "Sudo Main Page". 2003. URL: http://www.courtesan.com/sudo/ (May 30, 2003)

Oracle. "Oracle Collaboration Suite: Oracle Email". URL: http://otn.oracle.com/products/oemail/email_fo.html (May 30, 2003)

Postel, Jonathan B. "RFC 821 - Simple Mail Transfer Protocol". August 1982. URL: http://ietf.org/rfc/rfc821.txt (May 30, 2003) Project Cyrus. "Cyrus IMAP Server". URL: http://asg.web.cmu.edu/cyrus/imapd/ (May 30, 2003)

Qualcomm. "Qpopper". URL: http://www.eudora.com/qpopper/ (May 30, 2003) UC Berkeley, "Model Privileged Access Agreement", http://itpolicy.Berkeley.EDU:7015/proceds/access.html (May 30, 2003) Venema, Wietse. "www.porcupine.org". http://www.porcupine.org (2003)

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Sendmail. "Architecture". Filtering Mail with Sendmail. http://www.sendmail.com/partner/resources/development/milter_api/design.html. (May 30, 2003)

Sendmail. "Filtering Mail with Sendmail", http://www.sendmail.com/partner/resources/development/milter_api/ (May 30, 2003)

Sendmail. ftp://ftp.sendmail.org/pub/sendmail/sendmail-current.tar.Z (May 30, 2003)

Source Forge Tripwire Project. "Project: Tripwire: Summary". URL: http://sourceforge.net/projects/tripwire (May 30, 2003) "xinetd". URL: http://www.xinetd.org (May 30, 2003)

Source Forge Swatch Project. "Project: Swatch: File List ". URL: http://sourceforge.net/project/showfiles.php?group_id=68627 (May 30, 2003)

Tripwire. URL: http://www.tripwire.com/products/product_content/tfs_functional.cfm (May 30, 2003)

Tripwire. "Tripwire for Servers". URL: http://www.tripwire.com/products/servers/ (May 30, 2003)

Tripwire. "Tripwire Academic Source Release". URL: http://www.tripwire.com/products/tripwire_asr/ (May 30, 2003)

Tripwire. "Version Comparison". URL: http://www.tripwire.com/products/tripwire_asr/compare.cfm (May 30, 2003)

Vaudreuil, G. "RFC 1893 - Enhanced Mail System Status Codes" January 1996. URL: http://ietf.org/rfc/rfc1893.txt (May 30, 2003)

Ying, Charles. "Sendmail::Milter". URL: http://search.cpan.org/author/CYING/Sendmail-Milter-0.18/Milter.pm (May 30, 2003)

Books

Cheswick, William R. and Bellovin, Steven M. "Firewalls and Internet Security". Addison-Wesley Publishing Company, 1994. (Updated second edition is now available).

Costales, Bryan and Allman, Eric. "Sendmail". Third Edition. O'Reilly & Associates. 2002.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Garfinkel, Simson and Spafford, Gene. "Practical UNIX & Internet Security". 2nd Edition. O'Reilly & Associates. 1996.

Wall, Larry, Christiansen, Tom, and Schwartz, Randal L. "Programming Perl". 2nd Edition. O'Reilly & Associates, 1996. (Updated third edition is now available).

Appendix 1 - Milter to Detect Messages Containing Common Viruses

```
#include <errno.h>
#include <stdio.h>
#include <time.h>
#include <signal.h>

/*  A milter to reject messages containing selected worms/viruses */
/*  This is derived from the sample milter distributed with sendmail */

/*                    FOR DEMONSTRATION PURPOSES              */
/*  The author assumes no liability if this does not work    */
/*  Copyright 2003                                           */

#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sysexits.h>
#include <unistd.h>
#include <syslog.h>

#include "libmilter/mfapi.h"

typedef int bool;

#ifndef FALSE
# define FALSE     0
#endif /* ! FALSE*/
#ifndef TRUE
# define TRUE    1
#endif /* ! TRUE*/

struct mlfiPriv
{
        char        mlfi_env_from[100];
        char        mlfi_virus[100];
};

#define MLFIPRIV        ((struct mlfiPriv *) smfi_getpriv(ctx))

static int n_open;
```

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
static int n_messages;
static time_t last_summary;
static int n_worms_sir;
static int n_worms_bad;
static int n_worms_goner;
static int n_worms_myparty;
static int n_worms_kleza;
static int n_worms_klezg;
static int n_worms_klezgen;
static int n_worms_gibe;

/* following initialized by read_signatures() */
int n_sig = -1;
char *names[200];
char *patterns[200];
int occurrences[200];

timestamp()
{
        time_t ts;
        char tsascii[30];

        ts = time( NULL );
        strcpy( tsascii, ctime( &ts ) ) ;
        tsascii[19] = '\0';
        printf( "%s: %d ", tsascii+4, n_messages );
}

extern sfsistat         mlfi_cleanup(SMFICTX *, bool);

sfsistat
mlfi_envfrom(ctx, envfrom)
        SMFICTX *ctx;
        char **envfrom;
{
        struct mlfiPriv *priv;
        time_t cur;
        char time_temp[30];
        int iter = 0;
        int i;

        /* allocate some private memory */
        priv = malloc(sizeof *priv);
        if (priv == NULL)
        {
                /* can't accept this message right now */
                timestamp();
                printf( "*** malloc for priv failed\n" );
                return SMFIS_ACCEPT;
        }
        memset(priv, '\0', sizeof *priv);

        /* save the private data */
        smfi_setpriv(ctx, priv);
        strcpy( priv->mlfi_virus, "" );
        strncpy( priv->mlfi_env_from, *envfrom, 98 );
```

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
        n_open++;
        n_messages++;
        cur = time( NULL );
        if( cur - last_summary > 300 ) {
                strcpy( time_temp, ctime( &cur ) );
                time_temp[16] = '\0';
                printf( "%s: %5d messages ",
                        time_temp,
                        n_messages );
                for( i = 0; i <= n_sig; i++ ) {
                        if( occurrences[i] > 0 ) {
                                printf( " %s=%d", names[i],
occurrences[i] );
                                occurrences[i] = 0;
                        }
                }
                printf( "\n" );
                fflush( stdout );
                n_messages = 0;
                last_summary = cur;
        }

        /* continue processing */
        return SMFIS_CONTINUE;
}

mlfi_envto(ctx, envto)
        SMFICTX *ctx;
        char **envto;
{

        /* continue processing */
        return SMFIS_CONTINUE;
}

sfsistat
mlfi_body(ctx, bodyp, bodylen)
        SMFICTX *ctx;
        u_char *bodyp;
        size_t bodylen;
{
        struct mlfiPriv *priv = MLFIPRIV;
        char *ptr;
        u_char temp;
        int i;

        /* check for viruses */

        /* make sure bodyp contains a null character
           don't worry about last char as signature for Sircam
           always pretty early
         */
        temp = bodyp[bodylen];
        bodyp[bodylen] = '\0';
        for( i = 0; i <= n_sig; i++ ) {
                if( ptr = strstr( (char *)bodyp, patterns[i] ) ) {
```

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
                        strcpy( priv->mlfi_virus, names[i] );
                        occurrences[i]++;

                        /* log type of worm and envelope from */
                        openlog( "milter", LOG_PID | LOG_CONS, LOG_USER
);
                        syslog( LOG_WARNING, "virus: '%s' from: '%s'",
                                priv->mlfi_virus, priv->mlfi_env_from
);
                        /* reject ... */
                        return mlfi_cleanup( ctx, FALSE );
                }
        }
        bodyp[bodylen] = temp;

        /* continue processing */
        return SMFIS_CONTINUE;
}

sfsistat
mlfi_eom(ctx)
        SMFICTX *ctx;
{
        return mlfi_cleanup(ctx, TRUE);
}

sfsistat
mlfi_close(ctx)
        SMFICTX *ctx;
{
        return SMFIS_ACCEPT;
}

sfsistat
mlfi_abort(ctx)
        SMFICTX *ctx;
{
        return mlfi_cleanup(ctx, FALSE);
}

sfsistat
mlfi_cleanup(ctx, ok)
        SMFICTX *ctx;
        bool ok;
{
        sfsistat rstat = SMFIS_CONTINUE;
        struct mlfiPriv *priv = MLFIPRIV;
        char *p;

        n_open--;
        if( ok == FALSE ) {
                rstat = SMFIS_REJECT;
        }
        if (priv == NULL) {
                return rstat;
        }
        /* release private memory */
```

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
        free(priv);
        smfi_setpriv(ctx, NULL);

        /* return status */
        return rstat;
}

static int file_count;

struct smfiDesc smfilter =
{
        "SampleFilter",      /* filter name */
        SMFI_VERSION,        /* version code -- do not change */
        SMFIF_ADDHDRS,       /* flags */
        NULL,                /* connection info filter */
        NULL,                /* SMTP HELO command filter */
        mlfi_envfrom,        /* envelope sender filter */
        mlfi_envto,          /* envelope recipient filter */
        NULL,                /* header filter */
        NULL,                /* end of header */
        mlfi_body,           /* body block filter */
        mlfi_eom,            /* end of message */
        mlfi_abort,          /* message aborted */
        mlfi_close           /* connection cleanup */
};


int
main(argc, argv)
        int argc;
        char *argv[];
{
        int c;
        const char *args = "p:";


        /* Process command line options */
        while ((c = getopt(argc, argv, args)) != -1)
        {
                switch (c)
                {
                        case 'p':
                                if (optarg == NULL || *optarg == '\0')
                                {
                                        (void) fprintf(stderr, "Illegal conn:
%s\n",
                                                        optarg);
                                        exit(EX_USAGE);
                                }
                                (void) smfi_setconn(optarg);
                                break;

                }
        }
        if (smfi_register(smfilter) == MI_FAILURE)
        {
                fprintf(stderr, "smfi_register failed\n");
```

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
                exit(EX_UNAVAILABLE);
        }
        timestamp();
        printf( "Starting Virus milter...\n" );
        read_signatures();
        fflush( stdout );
        signal( SIGHUP, SIG_IGN );
        signal( SIGTERM, SIG_IGN );
        signal( SIGINT, SIG_IGN );
        n_open = 0;
        last_summary = time( NULL );
        n_messages = 0;
        return smfi_main();
}
```

© SANS Institute 2003, Author retains full rights

Last Updated: September 5th, 2015

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Crystal City 2015	Crystal City, VAUS	Sep 08, 2015 - Sep 13, 2015	Live Event
SANS Network Security 2015	Las Vegas, NVUS	Sep 12, 2015 - Sep 21, 2015	Live Event
SANS Seoul 2015	Seoul, KR	Sep 14, 2015 - Sep 19, 2015	Live Event
SANS Tallinn 2015	Tallinn, EE	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS Baltimore 2015	Baltimore, MDUS	Sep 21, 2015 - Sep 26, 2015	Live Event
Data Breach Investigation Summit & Training	Dallas, TXUS	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS Perth 2015	Perth, AU	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS ICS Amsterdam 2015	Amsterdam, NL	Sep 22, 2015 - Sep 28, 2015	Live Event
SANS Bangalore 2015	Bangalore, IN	Sep 28, 2015 - Oct 17, 2015	Live Event
SANS Seattle 2015	Seattle, WAUS	Oct 05, 2015 - Oct 10, 2015	Live Event
SANS DFIR Prague 2015	Prague, CZ	Oct 05, 2015 - Oct 17, 2015	Live Event
SOS: SANS October Singapore 2015	Singapore, SG	Oct 12, 2015 - Oct 24, 2015	Live Event
SANS Tysons Corner 2015	Tysons Corner, VAUS	Oct 12, 2015 - Oct 17, 2015	Live Event
SANS Gulf Region 2015	Dubai, AE	Oct 17, 2015 - Oct 29, 2015	Live Event
SANS Tokyo Autumn 2015	Tokyo, JP	Oct 19, 2015 - Oct 31, 2015	Live Event
SANS Cyber Defense San Diego 2015	San Diego, CAUS	Oct 19, 2015 - Oct 24, 2015	Live Event
SANS Sydney 2015	Sydney, AU	Nov 09, 2015 - Nov 21, 2015	Live Event
SANS South Florida 2015	Fort Lauderdale, FLUS	Nov 09, 2015 - Nov 14, 2015	Live Event
SANS London 2015	London, GB	Nov 14, 2015 - Nov 23, 2015	Live Event
Pen Test Hackfest Summit & Training	Alexandria, VAUS	Nov 16, 2015 - Nov 23, 2015	Live Event
SANS Hyderabad 2015	Hyderabad, IN	Nov 24, 2015 - Dec 04, 2015	Live Event
SANS Cape Town 2015	Cape Town, ZA	Nov 30, 2015 - Dec 05, 2015	Live Event
SANS San Francisco 2015	San Francisco, CAUS	Nov 30, 2015 - Dec 05, 2015	Live Event
Security Leadership Summit & Training	Dallas, TXUS	Dec 03, 2015 - Dec 10, 2015	Live Event
SANS Milan 2015	OnlineIT	Sep 07, 2015 - Sep 12, 2015	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced