Study: Improving Security in Corporate (SMTP) E-Mail Delivery

Brian Sommers

Case Study: Improving Security in Corporate (SMTP) E-Mail Delivery

GSEC Practical Assignment, Version 1.4b Option 2
Brian Sommers
December 23, 2003

1. Abstract

When I started working at my current job as information security analyst, I found that there were several problems with the configuration of Internet services (e-mail, web, DNS, etc.) in regard to security. The Internet servers were not hardened either at the operating system or the application level, the systems were all located on the internal network (not in a DMZ), and there were implementation details that allowed certain Internet traffic to bypass security controls that were in place. For this case study, I will examine one of these Internet services, e-mail over SMTP (Simple Mail Transfer Protocol), and what was done to improve the security of that system.

2. Before: Several security problems identified

The previous setup for SMTP mail is illustrated in Figure 1. To provide redundancy in case of a server failure, two systems were set up to accept Internet mail. This was implemented by using MX records in DNS, with one server having a higher priority than the other. SMTP mail would first attempt to deliver to the system with a higher priority MX value and fail over to the lower valued server if necessary (if the primary system was busy or unavailable). This is a good practice, but unfortunately in our case it allowed mail to bypass specific security controls in place. One of the two servers is a Windows system running the MMS mail content scanner from Tumbleweed Communications. This software acts as an SMTP gateway and has several policies you can apply to messages to enforce security, such as anti-virus checking, anti-spam rules, corporate confidentiality, mail archiving, etc. If the primary server (the MMS system) was unable to handle an SMTP message, it would fail over to the secondary system, which was our primary Microsoft Exchange server. This was not the desired behavior because mail was being delivered directly to the Exchange server without being scanned by the anti-virus engine and other policies located on the MMS server.

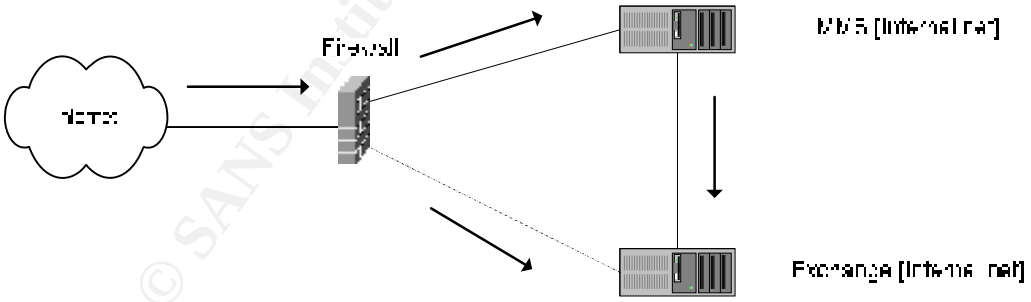

Figure 1. Previous SMTP flow

Neither of the two mail servers had any special configuration done with security in mind. The systems did not have their operating systems hardened to reduce installed software, change default permissions, or eliminate standard company login accounts. This meant that the systems would be an easier target for a potential attacker. Also, there were no intrusion detection system (IDS) components installed on the company's network or servers, making it less likely that an attack or compromise would even be detected.

Another security problem with this arrangement is that if an attacker from the Internet were able to compromise one of the two mail servers, he would be on the internal network and have the ability to attack any other system on the network, including our database servers containing confidential customer data.

We had several incidents in which employee's PCs were infected with a virus that would send copies of infected Internet mail through a self-contained SMTP server in the virus code, circumventing our mail servers and their anti-virus checks. As part of our SMTP security clean-up, we identified this as a problem that needed to be corrected.

3. Planning and Requirements

To correct these security problems, we decided to keep the internal e-mail systems but augment them with 2 new SMTP servers to be installed in a new DMZ network. This design would still provide redundancy for Internet messages and would allow us to funnel all mail through the MMS anti-virus and content-scanning system as shown in Figure 2.

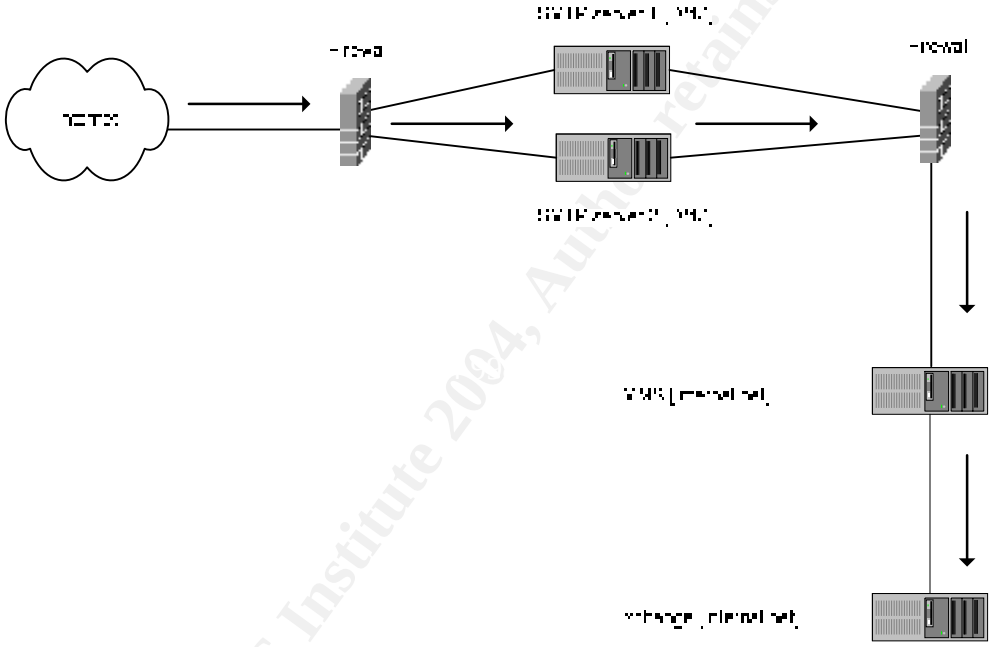

Figure 2. Planned SMTP flow

The goals for our new SMTP architecture would attempt to satisfy the following security principles that were stressed in the SANS Security Essentials[1] course:

3.1 Know Thyself: Know all about your systems, including what accounts, services, and ports they are using.

To satisfy this requirement, we would only install a minimum number of accounts necessary for the operating system and applications, any unnecessary software or services would be removed or disabled, and port scans would be used to identify all listening ports. In addition, each DMZ server would be required to have information recorded in a binder that documented all of the installed software (with version numbers), IP and MAC addresses, port and protocol requirements (including the direction of traffic), and the names of administrators responsible for the systems.

3.2 Least Privilege: Use the least amount of privileges and services required to do the job.

In this case, the SMTP software would be required to run as a non-privileged account and the TCP/IP communications to and from the servers would be limited to the minimum necessary by using the DMZ firewall.

3.3 Defense in Depth: Use layered security to repel, or at least slow down, attacks.

To satisfy this requirement, the SMTP software running on the DMZ mail servers must be different than the SMTP software running on the next hop in the mail routing scheme. This would prevent an attacker from using the same exploit to jump into the DMZ and then right into the internal network. The firewall also presents one additional layer of defense that must be breached as compared to the previous configuration.

3.4 Prevention is Ideal but Detection is a Must: The earlier you detect malicious activity the better.

In the area of detection, it was decided to add network-based IDS sensors in the new DMZ network and host-based IDS components to the SMTP servers in the DMZ (as well as all other DMZ servers). This logging would be in addition to whatever OS and application logging would be provided by the server.

4. During: Implementing the new solution

4.1 DMZ

The first task to be performed was the creation of a third-leg DMZ for our perimeter network. In the previous configuration, we had a pair of firewall appliances set up in a failover configuration, such that if the primary firewall had a serious error the secondary would automatically take its place. Each firewall had 2 network interface cards (NICs) which were used to define the external network (our border router to our ISP) and the internal network (everything else, including Internet servers, internal servers, and client desktop systems). For this effort, we added 2 NICs per firewall that allowed us to create a DMZ network and an additional network we ended up using for IDS communications. The firewalls are still configured for failover. The DMZ network would be a tightly-controlled resource, where any communication allowed in or out of the network would have to be explicitly defined and all servers would be secured at the operating system and application levels.

4.2 Operating System and Application Selection

After the DMZ was set up, the next task was to build the new SMTP servers. I decided to use Solaris Unix systems for several reasons. First, I have an extensive background in administering Unix systems (and Solaris in particular), so I felt I could do a better job in knowing the ins and outs of such a system than if we used a Windows system (which is the other predominant OS at our company). Secondly, I have had positive experiences using the Solaris Security Toolkit [2] (known as JASS) to harden a system. JASS is a set of scripts that Sun provides as an installable package. When run, it applies a series of changes to a Solaris system that follow best practices for minimizing services and increasing the security of default operating system settings.

The next decision I made concerning the new servers was to deploy Postfix as the SMTP server software. From the Postfix web site, the software is described as such: “Postfix attempts to be fast, easy to administer, and secure, while at the same time being sendmail compatible enough to not upset existing users.” [3] I had previous experience with running Sendmail, both as the built-in MTA (message transfer agent) that ships with Solaris systems and as an add-on compiled from the open source code available at sendmail.org. I was attracted to Postfix, however, because it has had fewer security vulnerabilities than Sendmail (a quick search at Securityfocus.com shows

Postfix has had 7 related vulnerabilities since 2000, compared to 25 for Sendmail) [4] and also because it is, in my opinion, easier to configure. The Postfix design incorporates several security features, including separation of duties by different processes, avoidance of set-uid programs, dynamic memory allocation to avoid buffer overflows, and filtering of user-supplied data. [5] Since the internal mail servers do not use Postfix, we have added an additional layer of security. An attacker who could exploit the postfix servers would need a different method to attack the next hop, which would be the MMS system.

To save on hardware resources, it was decided we would use these same 2 systems to also act as DNS (domain name system) servers and NTP (network time protocol) servers. Based on observations of similar servers, it was determined that this would not stretch the system resources too thin. I will not go into more detail on installing these additional services since this discussion is about SMTP, but I will touch on some of the implications of this decision later.

4.3 Building the System

With these decisions made, I began installing the operating systems on my Sun boxes. Following the principle of least privilege, I decided to install the smallest possible number of software packages by choosing the Core System Support group of packages. When the base operating system had been installed, I added some additional packages and removed some of the ones installed with the Core System Support option. I used suggestions made in an article [6] from Sun's Blueprints OnLine site to initially pick the Core option and then determine what packages to add to the initial installation. I used Spitzner's "Armoring Solaris: II" [7] paper as a starting point to identify packages that could safely be removed from my Solaris system to further limit my exposure to vulnerabilities. I also identified other packages that were not necessary for our application or hardware (your mileage may vary). The packages added and removed are listed in Appendix A. The idea behind this is that if your box doesn't have software X installed, then you are not vulnerable to an exploit against X. When finished, I compared a default Solaris installation to my reduced system and found that the former contained 541 software packages, while the latter only contained 108. One set of package removals I will mention here is `SUNWsndmr` and `SUNWsndmu`; they comprise the default Sendmail installation with Solaris; since we are not using Sendmail they were removed to avoid any confusion and to further reduce the number of programs on the system.

After the operating system was installed, the next step was to load the latest set of Sun recommended and security patches and run the JASS security toolkit. JASS works best when it is used as a wrapper around your patch process -- you put the patches where JASS expects to find them and it will install them as part of the hardening process. This is good practice because if you ran JASS and then later installed patches separately, the patch installer may re-enable something that JASS had previously disabled and thus you may lose a security setting you wanted to keep. I always install patches via JASS. There are several white papers available from Sun Blueprints Online [8,9] that detail steps that can be taken to improve the security of a default Solaris installation. Many of the guidelines described in these papers have been implemented in JASS. The Blueprints Online site also hosts documents that describe the JASS toolkit in great detail and show how you can customize it if necessary. Some of the security measures performed by JASS include:

- disabling some accounts and removal of others
- creating banner messages for logins, ftp, and telnet
- disabling almost all services (RPC, printing, NFS, inetd.conf programs, etc.)
- enabling of extra logging
- tuning kernel and networking settings for security

Four accounts were added to the Sun machines: one account for my own interactive logins, one account for the DNS server, one account for Postfix, and one account used by the secure shell (SSH) service. Each of these accounts was a non-privileged (that is, not root or super-user) login, and only the first one was given a password and the ability to log in to the system. The other

three were locked so that logins were not possible. Any default accounts (except root) installed with Solaris are automatically locked by JASS.

4.4 Adding Software

The first software added was to support remote logins and secure file transfers using SSH and SFTP (secure FTP). We chose the OpenSSH open source version of SSH because it is freely available and has a history of having bugs fixed quickly. To install this software we would first have to install the supporting software Zlib and OpenSSL [10]. Zlib is a library that provides data-compression, and OpenSSL is an open source implementation of the Secure Sockets Layer protocol [11,12]. Sun patch 112438-01 was also added to the systems to provide device files /dev/random and /dev/urandom as a source of entropy.

Instead of compiling these packages directly on the new systems, I use a different machine in our internal network to build packages for similar Sun systems and then move the compiled code to the DMZ systems in a tar file. This means we do not have to have a compiler and header files (also known as include files) on the minimized systems. The thought is that one of the first things an attacker might do once he compromises a box is download exploit code and then compile it on the target system. By eliminating the tools to compile code, we can make the attacker's job more complicated and time-consuming. For most open source packages, you have the following steps to build and install the software: *configure*, *make*, and then *make install*. For the DMZ servers, I put the code on my internal box of similar architecture (in my case, sun4u), perform the *configure* and *make* steps, but skip the *make install* step. Then I create a tar file of the build directory, move this to the DMZ system using SFTP, untar the directory, and then perform the *make install*. If you come across a package that will not install on the target system without the header files and/or a compiler, you can always add them to the system for the installation process and then immediately remove them when done.

Now I was ready to install the Postfix software. I downloaded the latest version from one of the mirrors listed at the Postfix web site, uncompressed and untared the distribution and then followed the instructions in the INSTALL text file. The first step is to perform the *make* command. On my particular setup, it complained because I had the shell environment variable LD_LIBRARY_PATH set – to continue I had to unset this. As mentioned previously, you do need to add a user account for Postfix, named "postfix," and a group, named "postdrop." The postdrop group should not be used by any user account, not even the "postfix" account. To install Postfix, either do a *make install* (for a new installation) or a *make upgrade* (to update an existing postfix server). Since this was a new installation I did a *make install* after I had moved the compiled code to the mail server. The installer will ask you several questions about your desired configuration; you can either accept the default or type in a different value for each item. At the end of the installation, you'll receive this message:

> Warning: you still need to edit myorigin/mydestination/mynetworks
> parameter settings in /etc/postfix/main.cf.

To set these values, cd to the Postfix config directory, /etc/postfix (the default, unless you changed this during the installation). Make a backup copy of the primary configuration file, main.cf, and then edit the original file. I set the following values to define my environment:

```
mydomain = mycompany.com
myhostname = host.mycompany.com
mydestination = $myhostname, localhost.$mydomain
relay_domains = mycompany.com myotherdomain.com
```

and left mynetworks and myorigin set to their defaults (Postfix will pick these values based on your system). The value of relay_domains determines what domain names the mail server will relay mail from; set this to all of the domains you need to handle mail for. Once this is done you

should be able to start postfix using the command `/usr/local/sbin/postfix start`. On my mail servers I've written a startup script for the /etc/init.d directory that does some checks on the postfix environment and then starts the daemon; this is listed in Appendix B. This script can also be used to stop or reload the daemon; I have linked this to the appropriate /etc/rc*.d directories so that Postfix is started when the system boots up and is stopped when the system is shut down. If there are any problems with the postfix configuration, the syslog file (/var/adm/messages for Solaris) would have such error messages listed.

The final set of software added to the systems was host-based intrusion detection system sensors. As a parallel task to the mail server improvements, I had gathered requirements, researched different IDS products, and installed several network-based IDS sensors, including one in the new DMZ network. As servers were moved to the new DMZ network, they were configured with the host portion of the IDS package. This gives us security event logging in near real-time from the IDS central console. I set up criteria for the IDS console to send out pages and e-mail messages for the more serious events in the hopes that an attack would be quickly noticed.

4.5 Testing and Firewall Changes

Before adding rules to the firewall to allow the new systems to send mail, I created an information sheet (example in Appendix C) that described all of the important facts about this system. This allows us to know at a quick glance what software a system is supposed to have, what its addresses are, and who is responsible if any questions come up about it. The form also has a field to enter the change control number from our configuration management process, to ensure any new DMZ systems have been properly approved.

In addition to this form, I put any other relevant security results or other pertinent data in a DMZ notebook. One of the pieces of data I include are the results of a scan with Nessus, the open source port and vulnerability scanner [13]. This tool will remotely scan a system and try to determine if any of its listening services or parts of the TCP/IP implementation are vulnerable to an attack. Any warnings or security holes must be corrected or explained before it is allowed to be accessed from the Internet.

The Nessus scan did not pick up any vulnerabilities, so the next step was to open up ports on the firewall for the servers to pass SMTP traffic. These are the specific rules I added:

- allow any Internet host to send data to the DMZ mail servers using TCP port 25
- allow the DMZ mail servers to send TCP port 25 traffic to any Internet host
- allow the DMZ mail servers to send TCP port 25 traffic to the internal mail content gateway (MMS)
- allow the internal mail content server (MMS) to send port 25 TCP to the DMZ
- block all other TCP port 25 traffic from leaving the internal network

The last 2 rules were a defense we added to limit virus-infected PCs on our network from being able to spread a virus using SMTP. This is found in many viruses, such as Mimail, Swen, and Bugbear. In addition to stopping the virus activity, it also alerts us to an infected machine because we will see numerous blocked attempts to send SMTP past the firewall.

After opening the firewall ports, I performed some tests to and from the DMZ mail servers to verify they could send and receive e-mail. As a final security check, I used the Mail Relay Testing web site from the Network Abuse Clearinghouse [14] to determine if the new servers could be tricked into acting as an open relay. An open relay is an SMTP server that will pass mail on to a third party (e.g., one not related to its own domain names). These can and will be used by spammers to send junk mail through servers so that they can hide their tracks and distribute the work of delivery. I had a bad experience at my former job where our mail server was used in this way -- it was not a pleasant experience. The mail server suffered due to the load of thousands of

unwanted messages traversing the system and I had to handle the complaints of users and administrators who contacted us about the problem.

The new servers passed each of the relay tests (results shown in Appendix D) and I placed the output in the DMZ notebook.

4.6 Configuring DNS

At this point I was ready to start passing regular company mail using the servers. To do this I modified the MX records in DNS for each of our domains that use e-mail accounts. To make sure I didn't disrupt our e-mail, I brought the new servers up with lower priority MX values at first, so they would only receive a small amount of mail. That way if there was a problem it would only impact a few messages.

Here is a short description of how the MX records are used. When a server wants to send mail to mycompany.com, it will query DNS for the MX records. If there is more than one MX record, the server listed with the highest priority (lowest numeric value) will be contacted first; if that system does not respond, the server will attempt to deliver to the second highest priority MX record, and so on. You may also have MX records with equal values to have a round-robin effect. Before bringing up our new servers, which I'll call new1 and new2, we had 2 MX records for our mail, which I'll refer to as old1 and old2. Here is the DNS data:

```
mycompany.com      MX    10 old1.mycompany.com.
                   MX    20 old2.mycompany.com.
```

where old1 was our MMS mail content server and old2 was the Exchange server. To start receiving Internet mail on the new servers, I put them in DNS as follows:

```
mycompany.com      MX    10 old1.mycompany.com.
                   MX    20 old2.mycompany.com.
                   MX    30 new1.mycompany.com.
                   MX    40 new2.mycompany.com.
```

Now a small amount of mail trickled in to the 2 new servers. After a few days of verifying everything worked as it should, I changed the MX records to eliminate the old servers altogether:

```
mycompany.com      MX    10 new1.mycompany.com.
                   MX    20 new2.mycompany.com.
```

Since DNS answers are cached (saved) by remote Internet servers, I left the firewall rules in place that allowed mail to be sent to the old servers for a few days and removed the rules when no more mail was being sent to them. The DNS examples above are actually from one of two different DNS systems we use, as we use a "split DNS" to have different DNS data used by internal systems and the external Internet. The external MX records above will get mail from the Internet to our DMZ, but we have separate MX records used internally, which would direct how mail should flow from the DMZ back into the internal network. Split DNS is explained thoroughly in DNS and BIND by Albitz and Liu [15].

At this point the new mail setup was working as planned.

5. After: Changes and problems

Initially we did not do any anti-spam functions on the Postfix mail servers. However as time went on and I learned more about postfix (and our spam numbers grew), I decided to implement some of the anti-spam features. Postfix uses several configuration files you can create to block mail based on a messages attributes, such as IP address of the connecting server, text in the body or

headers, domain name (or lack thereof) of the sender, etc. The main format for these lookup files is dbm; you create the ASCII text configuration file and then run the command `postmap [filename]` to create the binary DBM index files. The dbm files do not support regular expressions, however, which can be very useful when matching patterns in spam.

To add support for regular expressions in my configuration files, I added PCRE (Perl Compatible Regular Expressions) software to the mail servers and compiled Postfix to recognize this new format. The PCRE home page describes the library as: "a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5." [16] Once this was added, I could use Perl regular expressions in the configuration files.

Several months after putting the SMTP servers in place, we encountered an incident that highlighted several problems in our mail system. I had implemented Postfix's ability to search the bodies of messages for strings to search for spam. This worked just fine until one of our marketing employees sent 739 individual messages out that were each 3.6 MB in size (that's about 2.7 GB of mail all hitting the system at once). The mail servers both suffered under the heavy burden of not only delivering these many, big e-mail messages but also trying to perform pattern matching on the large attachments. The fact that the 2 mail servers were also responsible for serving DNS and NTP only made matters worse. It's not that those services added considerably to the systems' loads, but when the servers became unresponsive, it was very noticeable. It's likely that if the servers were only serving mail, no users would have noticed because it would just mean mail would be delayed. But when external DNS lookups started failing, it was very obvious. Looking back, it was a mistake to put several services on each system. From a security standpoint it is not ideal, because someone attacking one service can at a minimum impact another service, or worse, attack the additional services to further compromise your network. From a reliability standpoint it is not good practice either, because as we learned, one service's failure can impact other unrelated services.

After this incident, we made several changes. We spoke to the marketing department about more efficient ways to deliver mass communications. Postfix came out with a new version that allows the body searches to stop after checking a specified number of Kbytes, so the same scenario would not have such negative effects. Lastly, we allocated more servers to the DMZ so that we could put DNS on its own pair of servers, SMTP servers on another pair, and NTP on yet a third set of servers. I feel confident that this has increased both our security and our reliability.

6. Conclusion

The new SMTP servers have been in place for several months now and I feel that all of the original goals for this solution have been met. The systems and services are well documented and understood, the systems are running with minimal privileges and software packages, the machines are located in a limited access part of the network, we are using different application implementations at different levels, and we have logging and alert notifications from the IDS.

All mail is sent through the gateway that scans each message for viruses and other content as directed by our internal policies. That gateway is also the only system allowed to send SMTP through the firewall (to the DMZ mail servers), so mass-mailing viruses on internal PCs cannot spread to the Internet.

References

[1] Cole, Eric, et. al. SANS Security Essentials. SANS Press, 2003.

[2] Noordergraaf, Alex, et. al. "Solaris Security Toolkit (JASS)." July 2003. URL: http://wwws.sun.com/software/security/jass/

[3] Venema, Wietse. “The Postfix Home Page.” URL:
http://www.postfix.org/

[4] SecurityFocus. “Search.” December 2003. URLs:
http://www.securityfocus.com/search?category=2&query=postfix&rank=&submit=Search
http://www.securityfocus.com/search?category=2&query=sendmail&rank=&submit=Search

[5] Venema, Wietse. “Postfix Overview – Security.” URL:
http://www.postfix.org/security.html

[6] Noordergraaf, Alex. “Solaris Operating Environment Minimization for Security: A Simple Reproducible and Secure Application Installation Methodology.” Sun Blueprints OnLine November 2000.

[7] Spitzner, Lance. “Armoring Solaris: II.” July 20, 2002. URL:
http://www.spitzner.net/armoring2.html

[8] Watson, Keith and Noordergraaf, Alex. “Solaris Operating Environment Network Settings for Security.” Sun Blueprints OnLine December 2000.

[9] Noordergraaf, Alex and Watson, Keith. “Solaris Operating Environment Security.” Sun Blueprints OnLine April 2001.

[10] Reid, Jason and Watson, Keith. “Building and Deploying OpenSSH for the Solaris Operating Environment.” Sun Blueprints OnLine July 2001.

[11] Zlib. “Zlib home site.” November 23, 2003. URL:
http://www.gzip.org/zlib/

[12] OpenSSL. “Welcome to the OpenSSL Project.” November 4, 2003. URL:
http://www.openssl.org/

[13] Nessus. “Nessus.” November 5, 2003. URL:
http://www.nessus.org/

[14] Network Abuse Clearinghouse. “Mail relay testing.” 1999. URL:
http://www.abuse.net/relay.html

[15] Albitz, Paul and Liu, Cricket. DNS and BIND, 4th Edition. Sebastopol: O’Reilly and Associates, Inc, 2001. 342 – 350.

[16] Hazel, Philip. “PCRE - Perl Compatible Regular Expressions.” October 30, 2003. URL:
http://www.pcre.org/

Appendix

A. Solaris package modifications after installing Core Support (Solaris 8)

Added

SUNWlibC	Sun Workshop Compilers Bundled libC
SUNWlibCx	Sun WorkShop Bundled 64-bit libC
SUNWadmc	System administration core libraries
SUNWadmfw	System & Network Administration Framework
SUNWdoc	Documentation Tools
SUNWscpu	Source Compatibility, (Usr)
SUNWtoo	Programming Tools

SUNWtoox	Programming Tools (64-bit)
SUNWlibms	Sun WorkShop Bundled shared libm
SUNWlmsx	Sun WorkShop Bundled 64-bit shared libm
SUNWntpr	NTP, (Root)
SUNWntpu	NTP, (Usr)
SUNWarc	Archive Libraries
SUNWarcx	Archive Libraries (64-bit)
SUNWbash	GNU Bourne-Again shell
SUNWgzip	The GNU Zip (gzip) compression utility
SUNWman	On-Line Manual Pages
SUNWsprot	Solaris Bundled tools
SUNWsprox	Sun WorkShop Bundled 64-bit make library
SUNWscpux	Source Compatibility (Usr) (64-bit)
SUNWter	Terminal Information
SUNWbtool	CCS tools bundled with SunOS
SUNWbtoox	CCS libraries bundled with SunOS (64-bit)
SUNWlibm	Sun WorkShop Bundled libm)
SUNWlmx	Sun WorkShop Bundled misc. 64-bit libm files

Removed

SUNWpcelx	3COM EtherLink III PCMCIA Ethernet Driver)
SUNWpcmci	PCMCIA Card Services, (Root))
SUNWpcmcu	PCMCIA Card Services, (Usr))
SUNWpcmcx	PCMCIA Card Services (64-bit))
SUNWpcmem	PCMCIA memory card driver)
SUNWpcser	PCMCIA serial card driver)
SUNWpsdpr	PCMCIA ATA card driver)
SUNWxwdv	X Windows System Window Drivers
SUNWxwdvx	X Windows System Window Drivers (64-bit)
SUNWxwkey	X Windows software, PC keytables
SUNWxwmod	OpenWindows kernel modules
SUNWxwmox	X Window System kernel modules (64-bit)
SUNWnisr	Network Information System, (Root)
SUNWnisu	Network Information System, (Usr)
SUNWdtcor	Solaris Desktop /usr/dt filesystem anchor
SUNWatfsr	AutoFS, (Root)
SUNWatfsu	AutoFS, (Usr)
SUNWfcip	Sun FCIP IP/ARP over FibreChannel Device Driver
SUNWfcipx	Sun FCIP IP/ARP over FibreChannel Device Driver (64 bit)
SUNWfctl	Sun Fibre Channel Transport layer
SUNWfctlx	Sun Fibre Channel Transport layer (64-bit)
SUNWfcp	Sun FCP SCSI Device Driver
SUNWfcpx	Sun FCP SCSI Device Driver (64-bit)
SUNWi15cs	X11 ISO8859-15 Codeset Support
SUNWtleux	Thai Language Environment user files (64-bit)
SUNWwsr2	Solaris Product Registry & Web Start runtime support
SUNWfris	French install software localization
SUNWsndmr	Sendmail root
SUNWsndmu	Sendmail user

B. Postfix startup script (/etc/init.d/postfix)

```
#!/sbin/sh

case "$1" in
'start')
```

```
        echo 'Postfix SMTP daemon starting...'
        /usr/local/sbin/postsuper -s
        /usr/local/sbin/postsuper -p
        /usr/local/sbin/postfix check
        /usr/local/sbin/postfix start
        ;;

'stop')
        /usr/local/sbin/postfix stop
        ;;

'reload')
        /usr/local/sbin/postfix reload
        ;;

*)
        echo "Usage: $0 { start | stop | reload }"
        exit 1
        ;;

esac
exit 0
```

C. DMZ Request Form

DMZ Service Request Form

1. Instructions

This form must be used to either request a new system be added to the Internet DMZ network or if additional services need to be added to an existing DMZ server. Complete sections 2 and 3 and send to the IT Security department.

2. System Data [completed by requestor]

1. Date	
2. Hostname	
3. IP Address	
4. Ethernet Address(es)	
5. Hardware Vendor, Make, and Model	
6. Operating System Software and Version	
7. Primary Contact	
8. Secondary Contact	
9. Change Control #	
10. Requested Start Date	
11. How Long is Access Needed?	

12. System already in DMZ?	
13. Will sensitive data be transmitted? [1]	
14. What encryption will be used? (SSL, PGP, etc.)	
FOR SERVICES FROM INTERNET TO HOST 15. Internet Communication(s) Requested List **Source**, **Protocol/Port** for each (ex: 216.94.61.0, TCP/80)	
FOR SERVICES FROM HOST TO INTERNET 16. Internet Communication(s) Requested List **Destination**, **Protocol/Port** for each (ex: 204.70.57.242, TCP/53)	
FOR SERVICES FROM HOST TO LAN 17. LAN Communication(s) Requested List **Destination**, **Protocol/Port** for each (ex: 10.27.226.1, UDP/514)	
18. List installed services and version numbers (ex: BIND 8.2.4)	

[1] Sensitive data includes social security numbers, medical records, confidential company information, and credit card numbers. This information must be encrypted.

3. Hardening and Additional Information

Please describe the purpose of the above request(s) and any special considerations.

Also describe what methods were used in the system and application hardening for this request. This may include output from a tool or a checklist. Attach separate pages as necessary.

4. Implementation [completed by IT Security]

1. Internal IP Address	
2. Internal DNS Name	
3. External IP Address	

4. External DNS Name	
5. Port Security enabled?	
6. Port in Switch	

D. Mail relay testing results

NETWORK ABUSE CLEARINGHOUSE

Mail relay testing

Connecting to new1.mycompany.com for registered user test to abuse.net forwarding address ...

```
<<< 220 ***************
>>> HELO www.abuse.net
<<< 250 new1.mycompany.com
```

Relay test 1

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@abuse.net>
<<< 250 Ok
>>> RCPT TO:<user-38957@nf.abuse.net>
<<< 554 <user-38957@nf.abuse.net>: Relay access denied
```

Relay test 2

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest>
<<< 250 Ok
>>> RCPT TO:<user-38957@nf.abuse.net>
<<< 504 <spamtest>: Sender address rejected: need fully-
qualified address
```

Relay test 3

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<>
<<< 250 Ok
>>> RCPT TO:<user-38957@nf.abuse.net>
<<< 554 <user-38957@nf.abuse.net>: Relay access denied
```

Relay test 4

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@mycompany.com>
<<< 250 Ok
>>> RCPT TO:<user-38957@nf.abuse.net>
<<< 554 <user-38957@nf.abuse.net>: Relay access denied
```

Relay test 5

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@[192.168.97.102]>
<<< 250 Ok
>>> RCPT TO:<user-38957@nf.abuse.net>
<<< 554 <user-38957@nf.abuse.net>: Relay access denied
```

Relay test 6

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@mycompany.com>
<<< 250 Ok
>>> RCPT TO:<user-38957%nf.abuse.net@mycompany.com>
<<< 554 <user-38957%nf.abuse.net@mycompany.com>: Relay access denied
```

Relay test 7

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@mycompany.com>
<<< 250 Ok
>>> RCPT TO:<user-38957%nf.abuse.net@[192.168.97.102]>
<<< 554 <user-38957%nf.abuse.net@[192.168.97.102]>: Relay access denied
```

Relay test 8

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@mycompany.com>
<<< 250 Ok
>>> RCPT TO:<"user-38957@nf.abuse.net">
<<< 554 <user-38957@nf.abuse.net>: Relay access denied
```

Relay test 9

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@mycompany.com>
<<< 250 Ok
>>> RCPT TO:<"user-38957%nf.abuse.net">
<<< 504 <user-
38957%nf.abuse.net>: Recipient address rejected: need fully-
qualified address
```

Relay test 10

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@mycompany.com>
<<< 250 Ok
>>> RCPT TO:<user-38957@nf.abuse.net@mycompany.com>
<<< 554 <user-38957@nf.abuse.net@mycompany.com>: Relay access denied
```

Relay test 11

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@mycompany.com>
<<< 250 Ok
>>> RCPT TO:<"user-38957@nf.abuse.net"@mycompany.com>
<<< 554 <user-38957@nf.abuse.net@mycompany.com>: Relay access denied
```

Relay test 12

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@mycompany.com>
<<< 250 Ok
>>> RCPT TO:<user-38957@nf.abuse.net@[192.168.97.102]>
<<< 554 <user-38957@nf.abuse.net@[192.168.97.102]>: Relay access denied
```

Relay test 13

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@mycompany.com>
<<< 250 Ok
>>> RCPT TO:<@mycompany.com:user-38957@nf.abuse.net>
<<< 554 <user-38957@nf.abuse.net>: Relay access denied
```

Relay test 14

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@mycompany.com>
<<< 250 Ok
>>> RCPT TO:<@[192.168.97.102]:user-38957@nf.abuse.net>
<<< 554 <user-38957@nf.abuse.net>: Relay access denied
```

Relay test 15

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@mycompany.com>
<<< 250 Ok
>>> RCPT TO:<nf.abuse.net!user-38957>
<<< 504 <nf.abuse.net!user-
38957>: Recipient address rejected: need fully-qualified address
```

Relay test 16

```
>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@mycompany.com>
<<< 250 Ok
>>> RCPT TO:<nf.abuse.net!user-38957@mycompany.com>
<<< 554 <nf.abuse.net!user-38957@mycompany.com>: Relay access denied
```

Relay test 17

```
>>> RSET
<<< 250 Ok
```

```
>>> MAIL FROM:<spamtest@mycompany.com>
<<< 250 Ok
>>> RCPT TO:<nf.abuse.net!user-38957@[192.168.97.102]>
<<< 554 <nf.abuse.net!user-38957@[192.168.97.102]>: Relay access denied
```

Relay test result

All tests performed, no relays accepted.