Global Information Assurance Certification Paper

Case Study: Securing Microsoft Outlook Web Access with a Reverse HTTP Proxy Server

David J. Waldo
GSEC Practical Assignment v1.4b, option 2
February 25, 2004

Abstract

This paper will present a case study of how to secure an existing Microsoft Internet Information Server (IIS) and Outlook Web Access (OWA) interface using a reverse HTTP proxy server. Besides the obvious need to increase the security posture of IIS/OWA server, the final solution was also driven by other factors: minimal impact to existing infrastructure, cost of implementation, and level of available administrative talent to deploy and maintain the solution, which are all factors often overlooked when trying to increase network security and maintain secure practices.

After describing the initial configuration and requirements, this document will consider the pros and cons of a variety of technical solutions, and describe the reasoning that leads to the chosen solution. Details of the implementation will be presented, including: network topology and required open ports, configuration of host-based firewall, HTTP proxy server configuration and deployment in a chroot jail, and required DNS name resolution. This paper will not discuss the installation or hardening of an IIS server or OWA interface. Finally, this paper will evaluate the deployed solution and describe which risks have been addressed and which risks remain.

Problem Description

I work for company with fewer than 50 employees whose business is information delivery via the Internet. Like most businesses, email services are vital to our operations, and, like many businesses, my company provides internal corporate email services with a Microsoft Exchange 5.5 email server. However, our employees, either from home or while traveling, also need remote access to corporate email services. Therefore, the company provides external Internet access to corporate email with an Outlook Web Access 5.5 interface running on a Microsoft IIS 4.0 web server. A Cisco PIX 515 firewall is used to partition the corporate network into internal, DMZ, and external networks. Both the DMZ and internal networks use private IP address space as described in RFC1918[1]. All three components – MS Exchange, OWA, and IIS – are installed on a single machine running Microsoft NT 4.0 Service Pack 6, which is located on the internal corporate network. Prior to the deployment of the solution presented below, HTTP, not HTTPS, was used to access the OWA interface, which was directly accessible from the Internet; therefore, port 80 was open on the firewall from the external network to the Exchange/IIS/OWA machine. The IIS server used NTLM authentication exclusively, not HTTP basic authentication. Our company had an existing FTP server and an SMTP gateway server in the DMZ network, but no HTTP server. Incoming and outgoing SMTP traffic was routed through the existing SMTP gateway in the DMZ, so port 25 was not open from the external network to the Exchange/IIS/OWA machine in our internal network (See figure 1 for the existing IIS/OWA access diagram).

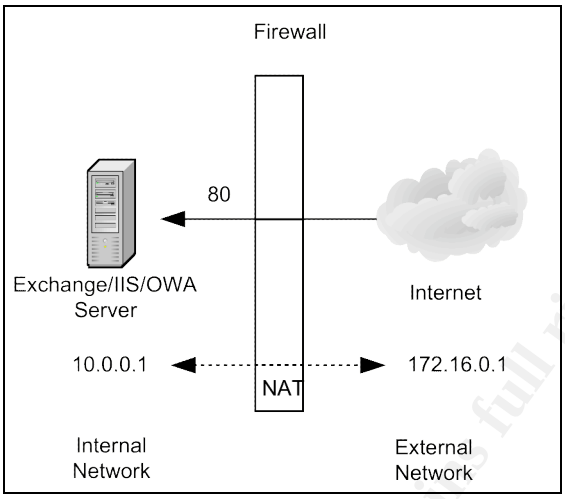

Figure 1

Our company's Information Technology (IT) personnel consisted of a Microsoft administrator, a Unix/Linux administrator, and myself, the IT manager and system architect. The Microsoft administrator maintained the company's MS Exchange server and a MS file server, but was primarily involved with the support of the company's desktop and laptop machines, which were running various Microsoft operating systems. He had no security training and little security awareness. The Unix/Linux administrator had far more network experience and security awareness. He maintained the company's co-located production infrastructure, which included a firewall, web load balancer, multiple web servers, and back-end database and text indexing servers. The production web server machines ran the Apache HTTP server on the Red Hat Linux 7.2 operating system. In addition, this administrator was responsible for the corporate network, which included a border router, firewall, and network switches, as well as 12 internal Linux machines, also all running Red Hat Linux 7.2. Given the nature of the company's business – content delivery via the Internet – security was a high priority in the production environment. Both the Unix/Linux administrator and myself were well versed in network security practices and were comfortable installing the Apache web server and other open-source software from source code.

The current IIS/OWA configuration presented many vulnerabilities and risks. Although the IIS web server had been previously hardened using the IIS lockdown tool[2] from Microsoft, IIS has a long history of being susceptible to various types of exploits. IIS vulnerabilities rank first in the SANS Top Vulnerabilities to Windows Systems[3], which references over 40 CVE/CAN entries. With continual announcements of new vulnerabilities, and constant

automated attacks threatening our IIS server, it was likely that new vulnerabilities would be found, and that IIS would continue to be a target for compromise well into the future. In addition, the OWA interface was transported via HTTP, so all its traffic was available to network sniffers. Because the IIS server was configured to use NTLMv2, our authentication data were not susceptible to network password sniffers, but our client browser was restricted to MS Internet Explorer, the only web browser that supported NTLM authentication over HTTP. Since the OWA interface and IIS server reside on the same machine as our Exchange server, a successful attack would allow the intruder direct control of our Exchange data store, one of our greatest corporate assets. A compromise of this machine would be disastrous for our company. In addition, since we allowed direct HTTP access from the Internet to this machine located in our internal network, if it were compromised, it could easily be used to compromise other internal machines. Lastly, we depended on our junior Microsoft administrator to maintain the Exchange, OWA, and IIS servers, and he had little security awareness or training. With the company's tight budget, he was unlikely to receive training in the future. In short, I found that we needed to increase the security posture of the MS Exchange/OWA server because of the high probability of future unanticipated exploits, the high risk to the company if the server were compromised, and the inability of our limited staff to respond to future IIS vulnerabilities in a timely manner.

Management agreed with my assessment that we needed to increase our network security and further protect the Exchange server, but we had no budget to upgrade the MS Exchange/OWA 5.5 sever or the MS NT 4.0 operating system. The existing versions of the software and operating system served the needs of our small number of employees adequately. Thus, I had to propose a solution that would not disrupt the existing email infrastructure, cost a minimal amount of money to implement, and not place a significant additional burden on the existing IT staff.

Analysis

From my SANS training, I knew I had to apply a defense in depth strategy to insulate the OWA/IIS server from direct access from the Internet. I would need to deploy an intermediate server in the DMZ, between the Internet and the internal IIS server, to handle HTTP requests and responses. Given this architecture, I had two options: (A) deploy an IIS server in the DMZ, migrate the OWA interface to this server, and use Microsoft's mechanisms for communication between a front-end OWA/IIS server and a back-end Exchange server, or (B) deploy a reverse HTTP/SSL proxy server in the DMZ and use the HTTP/SSL protocol for communication between the front-end proxy server and the back-end OWA/IIS/Exchange server.

Regardless of which option I would choose, I decided to deploy this new service on a single, dedicated machine in the DMZ network. A one-machine, one-external-service posture is vital from an incident response and business continuation perspective. A single machine with a single running service is quicker and easier to rebuild. I require that all our machines – especially those that are Internet facing in the DMZ – have well documented build procedures. If the machine is compromised, it can rapidly be replaced, with minimal impact to other machines delivering other services. A rapid response also means minimal disruption to the service that the compromised machine was delivering.

After researching option A in a Microsoft whitepaper[4], I found that deploying the front-end IIS/OWA server on a separate machine in the DMZ would require opening up multiple ports in the firewall, not just HTTP. We would also need to edit the registry of the Exchange server machine to statically bind ports for the Exchange directory and information store, which are normally dynamically assigned.

Option B only required that the HTTP port be open between the front-end and back-end servers. The two servers would be loosely coupled, and I was free to deploy any proxy server on any platform. I liked the simplicity and freedom that this solution offered. I realized that we could deploy an Apache HTTP/SSL server on a Linux machine in the DMZ, which could be maintained by our Unix/Linux administrator and myself, who both had considerable security experience maintaining our company's production environment. If I choose option A, we would just be moving the placement of the IIS server from the internal network to the DMZ. My concerns about our ability to maintain a secure IIS server would remain. Option B also deployed a defense in depth strategy not only in network zones, but also in server software and operating systems. It would be unlikely that an exploit would work against both an Apache/Linux server and an IIS/OWA/NT server. This solution would not only be the simplest and cheapest to implement, it would also minimally disturb our existing infrastructure, utilize the strengths of our existing administrative staff, and avoid platform monoculture[5].

Implementation

While deploying this additional layer of security, I also wanted to remove the restriction of using only the MS Internet Explorer as a client browser with OWA. If we used the Secure Sockets Layer (SSL) to tunnel all our HTTP traffic, we were no longer required to use NTLM authentication, and could instead safely use the less secure HTTP basic authentication protocol, which is supported by all web browsers. Therefore, we would only open port 443, not port 80, between the external and DMZ networks, and between the DMZ and internal networks (see figure 2).

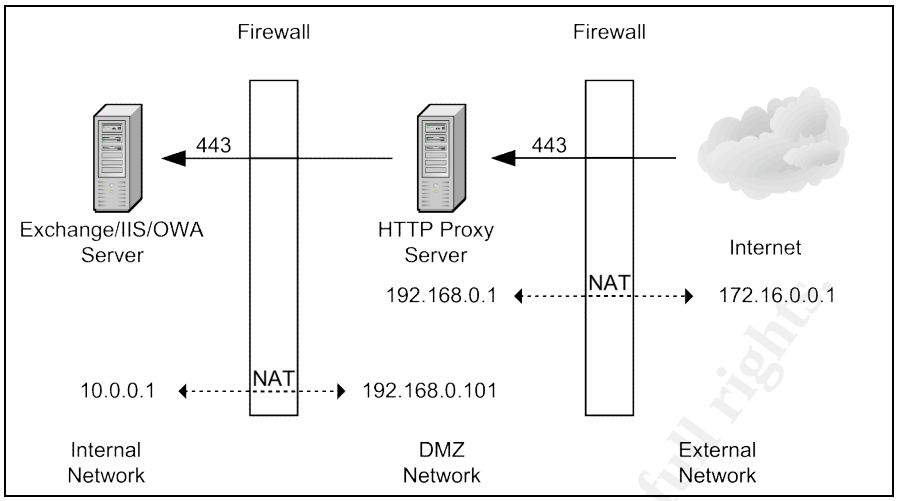

Figure 2

Linux Server Configuration

I had our Linux/Unix administrator configure a new machine, proxy.mycompany.com (192.168.0.1) that would be placed in the DMZ network. We had long ago standardized on Red Hat Linux 7.2 as the common platform for all of our Linux servers. Red Hat 7.2 is a bit dated at this point in time, but we had previously decided that – as long as the security and performance was adequate on a RH7.2 – it was best to keep all our Linux platforms consistent. I've been in other shops that installed the latest version of an operating system with each new machine rollout, and when a security patch was issued, the system administrator would have to scramble to find and install the correct patch for each machine. We planned to upgrade to a later release on all of our Linux servers on own schedule in the future. The proxy machine would only run two services: SSL and SSH. All other daemons would be turned off. The Linux administrator would perform the RH7.2 installation and install and configure openssh (http://www.openssh.org) from source code.

Since we build most of our open-source software from source code, we have standardized scripts to make the build process as simple as possible. The script we have developed for openssh installations is:

```
#! /bin/sh

if [ $# != 1 ]
then
    echo "usage: $0 version"
    exit
fi
VERSION=$1

ADMIN=/usr/local/admin
SOURCES=$ADMIN/build/sources
BUILD=$ADMIN/build/build
```

```
cd $SOURCES
if [ ! -f openssh-$VERSION.tar.gz ]
then
        echo "openssh-$VERSION.tar.gz not found"
        exit
fi
cat openssh-$VERSION.tar.gz | ( cd $BUILD; tar xvzf - )

cd $BUILD
chown -R 0.0 openssh-$VERSION
cd openssh-$VERSION
./configure --with-ssl-dir=/usr/local/ssl \
        --with-md5-passwords \
        --with-default-path=\
                /bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
make
make install

cp contrib/redhat/sshd.pam /etc/pam.d/sshd

if [ -f /usr/bin/ssh ]
then
        /bin/rm /usr/bin/ssh*
        /bin/rm /usr/bin/scp
        /bin/rm /usr/bin/sftp
fi
if [ -f /usr/sbin/sshd ]
then
        /bin/rm /usr/sbin/sshd
fi
if [ -d /usr/etc/ssh ]
then
        /bin/rm -rf /usr/etc/ssh
fi
```

Automating the build process and making it reproducible is especially important for a bastion host server located in the DMZ. All the build scripts, configuration files, and needed software on proxy.mycompany.com will be archived on a remote machine. If proxy is compromised, we replace it with a new server as quickly as possible using the rote build processes.

Before handing the server over to me, the Linux administrator would verify what services were running at what run-level with:

```
$ chkconfig -list
```

After a reboot of the machine, he would then verify that, at this point, only SSH was running on the machine. First, he would check this locally by running:

```
$ netstat -atup
```

Then he would double-check this from a remote machine using:

```
$ nmap -p1- 192.168.0.1
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on  (192.168.0.1):
(The 65532 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh

Nmap run completed -- 1 IP address (1 host up) scanned in 5 seconds
```

Note, the Apache/SSL server is not yet installed, so port 443 is not yet open. When working in teams, I find that it is important to have a written statement about who will perform what activities, and what commands or procedures they will use, so no mistakes or miscommunication take place during a handoff to another person.

Name Resolution and Network Address Translation

The existing OWA server, webmail.mycompany.com, had a DNS A record in both our internal and external DNS servers:

Server Name	Internal DNS A Record	External DNS A Record
webmail.mycompany.com	10.0.01	172.16.0.1

The Cisco PIX firewall performed network address translation (NAT) to bind the external address to the internal address:

```
static (inside,outside) 172.16.0.1 10.0.0.1 netmask 255.255.255.255 0 0
```

I didn't need to alter the internal or external DNS records, but I did need to establish the following relationships between the IP addresses of IIS/OWA server and the new proxy server in each of the three network zones:

Server Name	Internal IP Address	DMZ IP Address	External IP Address
IIS/OWA Server	10.0.0.1	192.168.0.101	-
Proxy Server	-	192.168.0.1	172.16.0.1

The external IP address of webmail.mycompany.com (172.16.0.1) would now be bound by the firewall to the proxy server's DMZ IP address (192.168.0.1). The IIS/OWA server's DMZ IP address (192.168.0.101) would be bound by the firewall to the internal IP address of IIS/OWA server's internal IP address (10.10.1.1). The PIX firewall mappings to bind these addresses would be:

```
static (dmz,outside) 172.16.0.1 192.168.0.1 netmask 255.255.255.255 0 0
static (inside,dmz) 192.168.0.101 10.10.1.1 netmask 255.255.255.255 0 0
```

The `/etc/hosts` file on the proxy server would resolve webmail.mycompany.com to 192.168.0.101. The entry in the /etc/hosts file on proxy.mycompany.com would read:

```
192.168.0.101    webmail webmail.mycompany.com
```

Network and Host Firewall Configuration

With the network address translation configured with the directives above, I needed to establish two conduits in the PIX firewall to allow SSL traffic from the Internet to the proxy server in the DMZ and to allow SSL traffic from the proxy server in the DMZ to the IIS/OWA server in the internal network:

```
conduit permit tcp host 172.16.0.1 eq 443 any
conduit permit tcp host 192.168.0.101 eq 443 host 192.168.0.1
```

Second, I would need to configure ipchains on the proxy.mycompany.com machine to allow only SSL and SSH traffic:

```
ipchains -A input -s 0.0.0.0/0.0.0.0 -d 198.168.0.1 443 -p tcp -j ACCEPT
ipchains -A input -s 0.0.0.0/0.0.0.0 -d 198.168.0.1 22 -p tcp -j ACCEPT
ipchains -A input -s 0.0.0.0/0.0.0.0 -d 0.0.0.0/0.0.0.0 -p tcp -j REJECT
ipchains -A input -s 0.0.0.0/0.0.0.0 -d 0.0.0.0/0.0.0.0 -p udp -j REJECT
service ipchains save
```

Apache Installation and Configuration

I wanted to keep the Apache installation as secure as possible, so I would install a minimal number of add-on modules, and I would configure the server to run in a chroot jail. I have used the Apache mod_rewrite module in a number of other web servers, and was comfortable with its syntax and power, so I choose to use it for the HTTPS proxy capability. Therefore, I would need the following source code software distributions:

openssl: http://www.openssl.org/source/
apache: http://httpd.apache.org/download.cgi
mod_ssl: http://www.modssl.org/source/

Once downloaded, I used the following automated scripts to build each of these modules. Each is similar to the SSH build script discussed above:

openssl

```
#! /bin/sh

if [ $# != 1 ]
then
        echo "usage: $0 version"
        exit
fi
VERSION=$1

ADMIN=/usr/local/admin
SOURCES=$ADMIN/build/sources
BUILD=$ADMIN/build/build

cd $SOURCES
if [ ! -f openssl-$VERSION.tar.gz ]
then
        echo "openssl-$VERSION.tar.gz not found"
```

```
        exit
fi
cat openssl-$VERSION.tar.gz | ( cd $BUILD; tar xvzf - )

cd $BUILD
chown -R 0.0 openssl-$VERSION
cd openssl-$VERSION
./config
make
make test
make install
```

apache

```
#! /bin/sh

if [ $# != 1 ]
then
        echo "usage: $0 version"
        exit
fi
VERSION=$1

ADMIN=/usr/local/admin
SOURCES=$ADMIN/build/sources
BUILD=$ADMIN/build/build

cd $SOURCES
if [ ! -f apache_$VERSION.tar.gz ]
then
        echo "apache_$VERSION.tar.gz not found"
        exit
fi
cat apache_$VERSION.tar.gz | ( cd $BUILD; tar xvzf - )

cd $BUILD
chown -R 0.0 apache_$VERSION
cd apache_$VERSION
export LD_LIBRARY_PATH=/usr/local/lib
cp $SOURCES/mod_proxy_add_forward.c src/modules/extra/.
./configure \
        --enable-module=rewrite \
        --enable-module=proxy
make
make install
```

modssl

```
#! /bin/sh

if [ $# != 1 ]
then
        echo "usage: $0 version"
        exit
fi
VERSION=$1

ADMIN=/usr/local/admin
SOURCES=$ADMIN/build/sources
BUILD=$ADMIN/build/build

cd $SOURCES
if [ ! -f mod_ssl-$VERSION.tar.gz ]
then
        echo "mod_ssl-$VERSION.tar.gz not found"
        exit
fi
cat mod_ssl-$VERSION.tar.gz | ( cd $BUILD; tar xvzf - )

cd $BUILD
```

```
APACHE=apache_`echo $VERSION | cut -d'-' -f2`
chown -R 0.0 mod_ssl-$VERSION
cd mod_ssl-$VERSION
./configure --with-apache=../$APACHE
cd $BUILD/$APACHE
./config.status \
        --enable-module=ssl
make
make install
```

Next, for maximum security, I wanted to run Apache in a chroot jail. I found directions on the "How to chroot an Apache tree with Linux and Solaris"[6] page. The directions are straightforward but lengthy, so I won't reproduce them here. Following the directions, I created a new chroot tree in `/chroot/apache` and populated it with the needed libraries and system files, then installed the Apache executable and configuration files into this tree. The new Apache tree is located in `/chroot/apache/usr/local/apache`. I then edited the `apachectl` startup script in the bin directory of this tree and inserted the commands for chroot execution:

```
PIDFILE=/chroot/apache/usr/local/apache/logs/httpd.pid
HTTPD="/usr/sbin/chroot /chroot/apache /usr/local/apache/bin/httpd -DSSL"
```

I then altered the standard Apache startup script, located in `/etc/init.d/httpd` to execute the apachectl located under the `/chroot/apache` directory:

```
#!/bin/sh
#
# Startup script for the Apache Web Server
#
# chkconfig: 345 85 15
# description: Apache is a World Wide Web server.  It is used to serve \
#              HTML files and CGI.

# Source function library.
. /etc/rc.d/init.d/functions

# See how we were called.
case "$1" in
start)
        echo -n "Starting httpd: "
        /chroot/apache/usr/local/apache/bin/apachectl start
        echo
        ;;
stop)
        echo -n "Shutting down http: "
        /chroot/apache/usr/local/apache/bin/apachectl stop
        echo
        ;;
status)
        /chroot/apache/usr/local/apache/bin/apachectl status
        ;;
restart)
        /chroot/apache/usr/local/apache/bin/apachectl restart
        ;;
reload)
        echo -n "Reloading httpd: "
        /chroot/apache/usr/local/apache/bin/apachectl restart
        echo
        ;;
*)
        echo "Usage: $0 {start|stop|restart|reload|status}"
        exit 1
esac
```

```
exit 0
```

I then set the run-level of this initialization script with the command:

```
checkconfig –add httpd
```

My company had an SSL certificate from Verisign for the existing webmail.mycompany.com domain name currently running on the IIS/OWA platform. Searching the web, I found directions on the "IIS SSL Certificate FAQ for IIS SSL"[7] page under the heading**, "**How to move a certificate from IIS 4.0 to Apache". Unfortunately, we were never able to successfully complete this conversion process. Instead, we eventually purchased a new SSL Certificate from Verisign and installed it according to the directions in the mod_ssl F.A.Q.[8]. We produced a RSA `webmail.mycompany.com.key` private key file, received the `webmain.mycompany.com.crt` SSL certificate from Verisign, and placed them in the chroot tree.

The last step was to edit the Apache httpd.conf file to configure the server and enable the reverse proxy behavior. First, the following directives are placed in the main configuration file so they apply to the entire server:

```
Port 443
Listen 443
```

I then created a virtual server with the directives below. Note that the name of the server is 'webmail.mycompany.com', since this virtual server will be accepting requests for that hostname. The IP address we are binding to is 192.168.0.1, the DMZ address of the proxy server, which is also the IP address that is bound – via the PIX firewall – to external IP address of 172.16.0.1. The DNS A record in our external DNS server also resolves to this same external address (172.16.0.1). In addition, I now reference the SSL certificate and key files that I created in the step above. Note, that we installed our certificate and key files in the following files:

```
/chroot/apache/usr/local/apache/conf/ssl.crt/webmail.mycompany.com.crt
/chroot/apache/usr/local/apache/conf/ssl.key/webmain.mycompany.com.key
```

But when these directives in the http.conf are read and parsed, the httpd process is already executing within a chroot jail, so the `/chroot/apache` directory is not appended to the front of the pathname. It is the root (or /) directory of the process, and all references to files use this directory as a base.

```
NameVirtualHost 192.168.0.1

<VirtualHost 192.168.0.1:443>
ServerName webmail.mycompany.com
DocumentRoot /usr/local/apache/htdocs
CustomLog /usr/local/apache/logs/webmail.access_log combined
ErrorLog /usr/local/apache/logs/webmail.error_log
SSLEngine on
SSLCertificateFile /usr/local/apache/conf/ssl.crt/webmail.mycompany.com.crt
```

```
SSLCertificateKeyFile /usr/local/apache/conf/ssl.key/webmail.mycompany.com.key
</VirtualHost>
```

Finally, I added the mod_rewrite directives to proxy HTTPS requests back to the IIS/OWA server within this <Virtual Host> scope:

```
<VirtualHost 192.168.0.1:443>
ServerName webmail.mycompany.com
DocumentRoot /usr/local/apache/htdocs
CustomLog /usr/local/apache/logs/webmail.access_log combined
ErrorLog /usr/local/apache/logs/webmail.error_log
SSLEngine on
SSLCertificateFile /usr/local/apache/conf/ssl.crt/webmail.mycompany.com.crt
SSLCertificateKeyFile /usr/local/apache/conf/ssl.key/webmail.mycompany.com.key

RewriteEngine on
RewriteRule ^/(exchange|public|exchweb)/(.*) \
                      https://198.168.0.101/$1/$2 [NC,P]
RewriteRule     ^/$  \
                      https://198.168.0.101/ [P]
</VirtualHost>
```

Mod_rewrite is a powerful tool used to perform URL manipulations. The documentation[9] and Guide[10] for this complex tool deserve careful examination. Please note that in the explanation of these directives, the backslash character (\) is a line-continuation symbol (in the Unix tradition) and is used for ease of presentation. The first directive:

```
RewriteEngine on
```

turns the rewrite engine on. The next directive is more complex:

```
RewriteRule ^/(exchange|public|exchweb)/(.*) \
                      https://198.168.0.101/$1/$2 [NC,P]
```

This rule consists of three parts: a pattern, a substitution, and flags. The pattern in this directive is a Regular Expression (more information about regular expression can be found in the regex(3) manpage) matching URLs that begin with a slash, followed by the string “exchange” or “public” or “exchweb” (which, if matched, is captured in the $1 variable for later use), followed by another slash, followed by the rest of the URL string (which, if matched, is captured in the $2 variable for later use). If this URL pattern is matched, the rewrite engine will perform the substitution in the second part of the line. If the incoming URL was,

```
http://webmail.mycompany.com/exchange/document.html
```

the “exchange” string would match and be placed in the $1 variable. The rest of the URL, “document.html”, would match and be placed in the $2 variable. The resulting substitution would be:

```
http://198.168.0.101/exchange/document.html
```

The IP address 198.168.0.101 is the address of the back-end IIS/OWA server, as it is known in the DMZ network. It will be bound to the 10.10.0.1 IP address as

packets flow through the PIX firewall to the internal network (see the Name Resolution and Network Address Translation section above). Finally, the two flags have the following meaning:

```
NC - Makes the Pattern case-insensitive.
P - Makes a proxy request of the resulting substitution string.
```

The last rewrite rule is much simpler. All page requests to the root (/) of the virtual server will be forwarded to the root of the back-end IIS/OWA server at 198.168.0.101.

It is important to note that page requests that do not conform to one of the specified URL patterns will not be proxied back to the IIS/OWA server. Potentially dangerous page requests, such as:

```
http://webmail.mycompany.com/_vti_bin/owssvr.dll?UL=1&ACT=4&BUILD=2614…
http://webmail.mycompany.com/msadc/Samples/SELECTOR/showcode.asp?…
```

will be rejected by the Apache server with a “404 – document not found” HTTP error. The Apache reverse proxy server will only forward legitimate OWA page requests, as defined in the Regular Expressions patterns of the rewrite rule directives.

To insure that the proxy server is ready to deploy and is only running the desired services, I run a final nmap port scan. Only port 22/tcp and 443/tcp are open.

```
$ nmap -p1- 192.168.0.1
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on  (192.168.0.1):
(The 65532 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh
443/tcp    open        https

Nmap run completed -- 1 IP address (1 host up) scanned in 5 seconds
```

Microsoft IIS configuration

With all the IP bindings, firewall configuration, and Apache configuration now complete, the only item remaining was reconfiguration of the IIS/OWA sever to accept SSL page requests, and to authenticate clients using the HTTP basic authentication protocol. Our Microsoft administrator configured the IIS server for SSL only traffic with the following procedure:

```
1.  Open the Internet Service Manager
        Click on Start -> Programs -> Windows NT 4.0 Option Pack ->
        Microsoft Internet Information Server -> Internet Service Manager.
2.  Type "443" into the "SSL Port" text box which is on the "Web Site" tab.
3.  Click on the "Directory Security" tab.
4.  Click the "Edit" button under "Secure Communications".
5.  Select the "Require Secure Channel when accessing this resource" Checkbox to
    disable all but HTTPS.
```

He then configured HTTP basic authentication with this procedure:

```
1. Open the Internet Service Manager
       Click on Start -> Programs -> Windows NT 4.0 Option Pack ->
       Microsoft Internet Information Server -> Internet Service Manager.
2. Click on the "Directory Security" tab.
3. Click the "Edit" button under "Anonymous Access and Authentication Control.
4. Ensure that "Basic Authentication" is the only checkbox selected.
```

Alternate Configuration

During the implementation described above, we debated whether to use the HTTP or SSL protocol when proxying page requests from the front-end proxy server to the back-end IIS/OWA server. We decided to use SSL over this path to maintain a secure channel all the way to the IIS/OWA server in order to prevent the sniffing of insecure HTTP basic-auth authentication data in the DMZ network. Other companies may use NTLM authentication, and so may be less concerned about network sniffers. Or corporate-wide IT policy may forbid the tunneling (via as SSL or SSH) of network traffic within an internal network, because tunneling undermines network intrusion detection efforts. Such a policy would require the use of HTTP, rather than SSL, over this channel, so I'll briefly describe the additional configuration needed to deliver this solution (see figure 3). I will not present each detailed configuration step as I did in the previous implementation. Instead, I'll just note how any configurations differ form those already presented, and discuss the additional steps required.

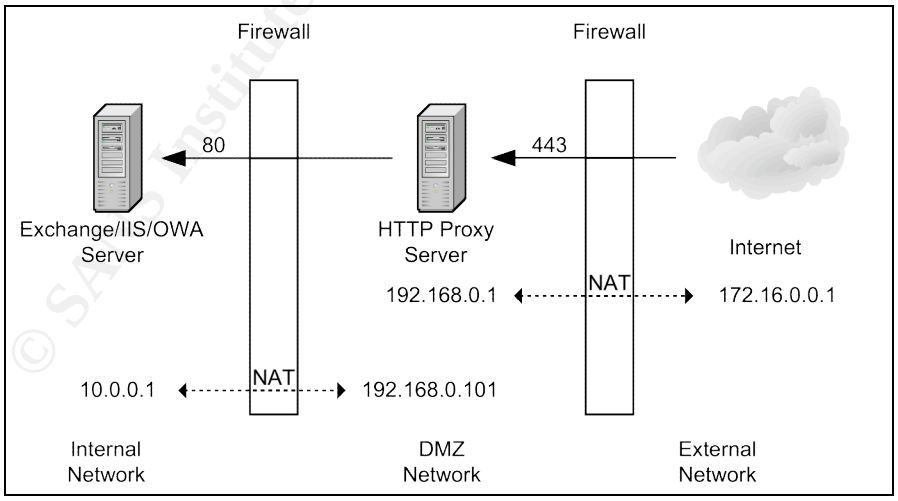

Figure 3

The basic problem with having an SSL front-end server proxy to a back-end HTTP IIS/OWA server is the OWA embeds full URLs in the HTML pages it delivers to web browsers. For example, a client browser issues a page request for `https://webmail.mycompany.com/`, which the proxy server would forward to the back-end IIS/OWA as a page request for `http://webmail.mycompany.com/`. The OWA interface would return an HTML referencing other pages with full URLs such as `http://webmail.mycompany.com/exchange/page.html`. When the client browser attempts to follow this hyperlink, it will issue an HTTP (not SSL) page request to the front-end proxy server. However, the front-end proxy server isn't configured to respond to HTTP requests on port 80. It only responds to SSL requests on port 443. We need to direct the back-end IIS/OWA server to embed 'https' URLs in the pages it returns, not 'http' URLs. The solution to this problem is presented in a Microsoft Exchange 2000 Server Front-End and Back-End Topology White Paper[11], which states, "If the SSL decryption is done on the front-end server, the front-end server knows SSL was used, and it notifies the back-end server of this by passing an HTTP header that says, "Front-End-Https: on" in all requests to the back-end server." In our front-end Apache proxy server, additional HTTP headers can be added to forwarded traffic with the optional Apache mod_proxy_add_forward[12] module.

Network and Host Firewall Configuration

In order to allow HTTP rather than SSL traffic from the front-end proxy server in the DMZ to the back-end IIS/OWA server in the internal network, we will replace the original PIX firewall directive of

```
conduit permit tcp host 192.168.0.101 eq 443 host 192.168.0.1
```

with

```
conduit permit tcp host 192.168.0.101 eq 80 host 192.168.0.1
```

Apache Installation and Configuration

We will still download and install openssl, mod_ssl, and Apache. In addition, we will download the mod_proxy_add_forward module, install it in the proper Apache source code directory, edit it to include only the 'front-end-https: on' header, and add the compilation of this module to our Apache build file. After downloading the source file of `mod_proxy_add_forward.c` from the URL listed in the references, I placed it in the `/src/modules/extra` directory of the Apache source code tree. I then edited the add_forward_header() function in the C source file, removed all the original C code that inserted the "X-Forwarded-For", "X-Host", and "X-Server-Hostname" HTTP headers into proxied requests, and replaced them with C code to add the "front-end-https" HTTP header set to "on". The resulting code is much simpler than the original:

```
static int add_forward_header(request_rec *r)
{
        if (r->proxyreq) {
                ap_table_set(r->headers_in, "front-end-https", "on");
                return OK;
        }
        return DECLINED;
}
```

I saved the file and added this new module to the Apache build process with an additional line in our Apache build script. Note the addition of the line: '`-- activate-module`':

```
#! /bin/sh

if [ $# != 1 ]
then
        echo "usage: $0 version"
        exit
fi
VERSION=$1

ADMIN=/usr/local/admin
SOURCES=$ADMIN/build/sources
BUILD=$ADMIN/build/build

cd $SOURCES
if [ ! -f apache_$VERSION.tar.gz ]
then
        echo "apache_$VERSION.tar.gz not found"
        exit
fi
cat apache_$VERSION.tar.gz | ( cd $BUILD; tar xvzf - )

cd $BUILD
chown -R 0.0 apache_$VERSION
cd apache_$VERSION
export LD_LIBRARY_PATH=/usr/local/lib
cp $SOURCES/mod_proxy_add_forward.c src/modules/extra/.
./configure \
        --enable-module=rewrite \
        --activate-module=src/modules/extra/mod_proxy_add_forward.c \
                --enable-module=proxy_add_forward \
        --enable-module=proxy
make
make install
```

Next, I altered the mod_rewrite rules in the httpd.conf file to proxy the OWA page requests to an 'http' URL, rather than the original 'https' URL:

```
<VirtualHost 192.168.0.1:443>
ServerName webmail.mycompany.com
DocumentRoot /usr/local/apache/htdocs
CustomLog /usr/local/apache/logs/webmail.access_log combined
ErrorLog /usr/local/apache/logs/webmail.error_log
SSLEngine on
SSLCertificateFile /usr/local/apache/conf/ssl.crt/webmail.mycompany.com.crt
SSLCertificateKeyFile /usr/local/apache/conf/ssl.key/webmail.mycompany.com.key

RewriteEngine on
RewriteRule ^/(exchange|public|exchweb)/(.*) \
                        http://198.168.0.101/$1/$2 [NC,P]
RewriteRule     ^/$  \
```

```
                        http://198.168.0.101/ [P]
</VirtualHost>
```

Microsoft IIS configuration

With the alternate firewall and apache configurations complete, we now only needed to reconfigure the IIS/OWA sever to accept HTTP page requests:

1. Open the Internet Service Manager
   Click on Start -> Programs -> Windows NT 4.0 Option Pack ->
   Microsoft Internet Information Server -> Internet Service Manager.

2. Click on the "Directory Security" tab.
3. Click the "Edit" button under "Secure Communications".
4. Unselect the "Require Secure Channel when accessing this resource" Checkbox to disable HTTPS and click OK.
5. Click OK in the "Directory Security" tab window.
6. Enter "80" in the "TCP Port" text box and click OK.

Conclusion

The implemented solution has significantly reduced the risks inherent with running an IIS/OWA server. The remaining vulnerabilities can be partitioned into three sets: (a) attacks against the Apache reverse proxy server, (b) attacks against the IIS server outside of the OWA namespace, and (c) attacks against the IIS server within the OWA interface namespace.

Though infrequent, new vulnerabilities occasionally are found in the components of our proxy server: Apache, mod_ssl (CVE-2002-0082[13]), and openssl. However, our implementation and build process have kept the risk of an attack against the proxy server low. Even if an intrusion were to occur, the httpd process is running in a chroot jail, so access to the system would be limited and damage would be minimal. In addition, because we only offer one service on this machine, and the machine build process is well documented, the service can quickly be restored on another machine.

The vulnerability and risk of attacks against the IIS server outside the OWA namespace has been eliminated. The Apache proxy server catches all HTTPS page requests outside the OWA namespace and never forwards them on to the IIS server.

The vulnerability to attack against the IIS server within the OWA namespace remains, but the number of these type of vulnerabilities is small in comparison to the total number of IIS vulnerabilities. Our only response to this risk is to continue to monitor vulnerability announcements for the OWA interface and the IIS server, and respond to them in a timely manner.

I judged the overall solution a success because – in addition to greatly enhancing the security posture of our webmail service – it did not disrupt the existing email infrastructure. It cost very little to implement, and it didn't add a significant

additional burden on our IT staff. The only disruption to our webmail service was requiring users to use a ‘https’ URL prefix instead of the original ‘http’ URL prefix. Because we used a surplus machine for the Linux proxy server and all the software was open-source and free, the only hard cost involved with the project was a few hundred dollars for the purchase of an additional SSL certificate. Finally, from an IT staffing perspective, we utilized our strengths. The production IT staff is best qualified to maintain an Internet facing server. They are already sensitive to security issues, comfortable with open-source software, and can easily maintain one more Linux machine without additional cost to our company.

References

[1] Rekhter, et al. "Address Allocation for Private Internets." RFC 1918. February 1996. URL: http://rfc.net/rfc1918.html (18 February 2004).

[2] Microsoft Corporation. "IIS Lockdown Tool." Microsoft TechNet. 2004. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/locktool.asp (18 February 2004).

[3] SysAdmin, Audit, Network, Security Institute. "Top Vulnerabilities to Windows Systems - W1 Internet Information Services." SANS Top 20 Vulnerabilities. Version 4.0. 8 October 2003. URL: http://www.sans.org/top20/#w1 (18 February 2004).

[4] Microsoft Corporation. "Planning and Deploying Outlook Web Access." 1999: 8-9. URL: http://support.microsoft.com/support/exchange/content/whitepapers/owaguide.doc (10 February 2004).

[5] Greer, Dan et al. "CyberInsecurity: The Cost of Monopoly." 24 September 2003. URL: http://www.ccianet.org/papers/cyberinsecurity.pdf (9 February 2004).

[6] Deatrich, Denice. "How to 'chroot' an Apache tree with Linux and Solaris." 26 February 2001. URL: http://penguin.epfl.ch/chroot.html (16 February 2004).

[7] InstantSSL Corporation. "IIS SSL Certificate FAQ for IIS SSL." URL: http://www.instantssl.com/ssl-certificate-support/server_faq/ssl-server-certificate-iis4.html (17 February 2004).

[8] Engelschall, Ralf. "mod_ssl 2.8, User Manual, F.A.Q. List." The Apache Interface to OpenSSL. 2001. URL: http://www.modssl.org/docs/2.8/ssl_faq.html#cert-real (17 February 2004).

[9] Engelschall, Ralf. "Apache module mod_rewrite." Apache HTTP Server Version 1.3 Documentation. July 1977. URL: http://httpd.apache.org/docs/mod/mod_rewrite.html (18 February 2004).

[10] Engelschall, Ralf. "Apache 1.3 URL Rewriting Guide." Apache HTTP Server Version 1.3 Documentation. December 1977. URL: http://httpd.apache.org/docs/misc/rewriteguide.html (18 February 2004).

[11] Microsoft Corporation. "Microsoft Exchange 2000 Server Front-End and Back-End Topology." 2001: 17. URL: http://www.somorita.com/Exchange2000/E2KFrontendBackEndTopology.doc (19 February 2004).

[12] Hansen, Ask. "mod_proxy_add_forward." Apache Module Registry. 28 February 2001. URL: http://modules.apache.org/search?id=124 (25 February 2004).

[13] The MITRE Corporation. "CVE-2002-0082." Common Vulnerabilities and Exposures. 6 June 2002. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0082 (23 February 2004)