

SECURING AND MANAGING THE ORACLE HTTP SERVER – REAL WORLD EXAMPLES AND LESSONS LEARNED

Kevin Sheehan, Unisys
Brian Mulreany, Unisys

EXECUTIVE SUMMARY

This paper describes practical solutions for securing your Oracle HTTP Server (OHS). First, the concept of Defense in Depth is introduced along with the important role of OHS within the context of the overall security architecture. Next, we cover how to securely install OHS and configure httpd.conf. The focus here is on real life examples of Apache directives that attendees can make use of in their own OHS environments. After addressing the basic hardening issues with OHS, the more advanced topics of configuring OHS as a reverse proxy server as well as implementing mod_security are covered. Finally, the paper concludes with tips and tricks for managing OHS.

SECURITY ARCHITECTURE OVERVIEW

While securing your web server is critical, doing so without consideration of the security posture of the entire environment is foolhardy. This section introduces the concept of *defense in depth* to ensure all potential attack vectors are considered.

CONFIDENTIALITY, INTEGRITY & AVAILABILITY (CIA)

The goal of the security architecture is simply to protect the confidentiality, integrity and availability of the information system itself, the data within it and the security controls that protect it.

- Confidentiality – Information should only be disclosed to authorized systems or personnel. Common controls used to enforce confidentiality include encryption and *role based access controls* (RBAC). Examples of common breaches of confidentiality include: sharing passwords, shoulder-surfing, sniffing unprotected wireless networks and stolen laptops and/or removable media.
- Integrity – Protecting information from unauthorized changes. Common controls used to enforce integrity include enforcement of a *least privilege* access control model, regular job rotations and technical controls such as referential integrity, anti-virus software and *Virtual Private Database* (VPD). Examples of common breaches include: computer viruses and SQL Injection attacks.
- Availability – Ensuring the system, as well as the access, audit and other security controls that protect the system, are continuously available as per business requirements. Common controls used to enforce availability include redundant hardware and software, *Oracle Real Application Clusters* (RAC) and disaster recovery sites. Examples of common breaches of availability include *Denial of Service* (DoS) attacks and single points of hardware or software failure.

Since the October 2006 *Critical Patch Update* (CPU), Oracle has used the *Common Vulnerability Scoring System* (CVSS) to evaluate the impact of Oracle security vulnerabilities on CIA. It is strongly recommended that information technology organizations that use Oracle products analyze the quarterly Oracle CPUs, their associated CVSS scores and their own security controls in order to determine an appropriate patching strategy. For more information on CVSS see <http://www.first.org/cvss/cvss-guide.html> and the CVSS calculator at <http://nvd.nist.gov/cvss.cfm?calculator&version=2>.

DEFENSE IN DEPTH

Defense in depth has its origins in military strategy. Conventional military theory amassed troop concentrations along the front line of the battlefield but if the enemy broke through that line there were no other defenses available. A defense in depth strategy spreads forces out into successive lines of protection well before the front line of battle. The strategy also employs multiple technologies in anticipation of differing military attack strategies, for instance, interspersing anti-tank trenches to stymie armored attacks or lines of barbed wire to slow infantry attacks. The strategy allows defensive troops to fall back in a

controlled manner, thus slowing down the enemy and giving military commander's time to determine the location of the attack and respond accordingly with additional military support.

Defensive strategies have matured within information security in a similar fashion. Originally, information security concentrated on protecting the perimeter, or edge, of your environment with firewalls and possibly *Network Intrusion Detection Systems* (NIDS) which was tantamount to putting all your defenses on the front line. Such a strategy is often called "crunchy on the outside, soft on the inside." It has the same pitfalls as the similar military strategy in that once the perimeter is breached; there are no other defenses available to protect your systems. It also provides no protection from security breaches INSIDE the firewall, and current published estimates report that inside attacks account for 70-90% of all incidents.

Within information security, defense in depth represents a layered approach to security by addressing the security posture of not only the technology stack but the operational and management controls within the system as well. A comprehensive list of controls can be found in the *National Institute of Standards and Technology* (NIST) "Recommended Security Controls for Federal Information Systems", aka NIST 800-53: <http://csrc.nist.gov/publications/nistpubs/800-53-Rev2/sp800-53-rev2-final.pdf>.

Since the focus of this paper is on the technology stack, we recommend at minimum the following technical controls within your system(s):

- Use firewalls to carve out, at minimum, three zones of protection:
 - *Demilitarized Zone* or DMZ which will typically contain your proxy servers
 - Intranet or trusted zone that will contain your application servers
 - Database or restricted zone that will contain your database servers
- Use at least two different types of firewall technologies so that a breach of a single firewall does not compromise all the security zones
- Use separate network infrastructure (routers, switches, etc.) for internet-facing components in the DMZ from that used within your intranet
- Use a switched network to restrict traffic to point-to-point communication
- Deploy NIDS
- Deploy a *Host Intrusion Detection System* (HIDS) on all servers
- Centrally manage NIDS and HIDS, typically from a *Security Operations Center* or SOC which is responsible for security incident management
- Harden all components (network, servers, web servers, application servers and database servers) based on secure benchmarks such as those provided by the *Center for Internet Security* (CIS) at <http://www.cisecurity.org>
- Insure all inbound http traffic flows through a reverse proxy server and all outbound http traffic flows through a forward proxy server
- Use redundant servers at all tiers including RAC at the database tier
- Encrypt all end user web traffic, preferably with a hardware-based SSL accelerator, typically deployed at the load balancers in front of the web or reverse proxy servers.
- Use only *Federal Information Processing Standard* FIPS 140-2 certified cryptographic modules for encryption. A list of vendors using such modules can be found at <http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/1401vend.htm>. Note that this list is not all inclusive so contact your particular vendor if you do not find them on the list.
- Do not allow end-users to directly access the database. Such access should be available only through an application.
- Harden your database listener by turning on valid node checking and providing a list of invited nodes that is restricted to your application servers
- Avoid the use of DNS on servers where possible. DNS adds another attack vector when name resolution can typically be handled by `/etc/host` files.

ROLE OF THE WEB SERVER

The web server is a critical component in the security architecture for a number of reasons. First, it is the gateway to all of your web-based applications and thereby the de facto attack vector of choice for those who intend to do harm. Second, the web server, by default, is configured to serve content and not protect content. Unless you take action to properly lock it down, OHS is insecure by default. The general strategy for protecting content is first lock it all down and then open only what should be opened. And finally, OHS contains a number of capabilities such as reverse proxy and mod_security that can significantly enhance the security posture of your system(s).

FIRST THINGS FIRST – EXACTLY WHAT SHOULD I INSTALL ON MY WEB TIER?

It seems a trivial question but it turns out to be a little more difficult than you might think and yet the answer is critical to your success. Let's begin with our first OHS lesson learned by reviewing the evolution of the web tier in our own environment.

HOW OUR WEB TIER EVOLVED

Our environment is entirely Oracle-based, including the web, application and database tiers. We support custom J2EE applications and Oracle Portal, so our middle tier has a J2EE/Portal installation of the Oracle Application Server.

CHIMPS (OR IS THAT CHUMPS)

Our initial design called for Oracle Web Cache at the web tier. So what did we install? We did a full J2EE/Portal install and then configured and started only Web Cache. Seems silly now but we thought it would be easier to simply clone the middle tier. This made everything the same on both the web and application tiers and the thinking was that this would be easier to maintain.

In reality we just created more headaches for ourselves. With this heavy footprint, patching was a constant issue. In addition, we had violated a fundamental security policy by installing software that was not needed. To make matters worse, this software was installed in our DMZ.

NEANDERTHALS

Our next evolutionary step was to replace the full blown J2EE/Portal install with Standalone Web Cache. This improved our security posture by significantly reducing our footprint. Patching was much quicker since only a single component was installed. However, we started having misgivings about Web Cache itself.

First of all, Web Cache is a single-thread process. Oracle recommends a 2-CPU server, one for the *Operating System* (OS) and one for Web Cache. Additional CPUs will have no impact; that is, you cannot scale up. If you need additional throughput you must buy more servers.

Second, Web Cache cannot be configured as a reverse proxy nor as a software firewall such as `mod_security` and it cannot be used to rewrite URLs.

HOMO SAPIENS

We have now replaced Standalone Web Cache with Standalone OHS. In the initial rollout, OHS was configured as a reverse proxy server. This allowed us to block any URLs that were not valid on the platform and stopped a lot of internet probing of the system by bad guys trying to *fingerprint* the environment.

In a second phase we deployed `mod_security` to block common signatures for *Cross-Site Scripting* and *SQL Injection*.

Note also that we deployed the Apache 2.0 version of OHS that allowed us to use the threaded implementation called *MPM Worker*. This gave us much greater throughput without buying any new hardware.

This configuration gives us better security and better performance, a rare win-win.

THERE ARE 10+ VERSIONS OF OHS

Oracle distributes over 10 different variations of OHS. This includes distributions with the Application Server, E-Business Suite, and the Database. The different versions are based on two different versions of Apache, 1.3 and 2.0. Oracle is planning to release a new version of the Application server based on Apache 2.2. The 2.2 version of Apache modified the Apache architecture. Load modules used with earlier versions of Apache are not compatible. This means a third major version will be part of the picture. Within each of these main distributions there are different point releases that correspond to the products and product releases that include OHS. This means 10gAS has slightly different versions depending on the 10gAS release. The same is true for EBS and the Database. Even Oracle has a hard time keeping track of all these versions. One of the OHS Metalink notes has this statement about an OHS version: *"It is externally labeled as "10.1.3.3", but the component version is actually "10.1.3.1", and is a special build, different than the Oracle Application Server counterpart.* " This can complicate maintenance, and particularly patching of OHS to prevent security vulnerabilities. You may find that the architecture goal of using the best Oracle technology available competes with the security goal to minimize the number of versions of products in production. When planning an OHS deployment you need to balance these goals to produce the best solution. If you have an unlimited

maintenance budget, or you have a lifetime goal to use every OHS version that has ever been released, then just grab the first download and get started. If not, you need to carefully consider which versions you plan to use.

ALL OHS VERSIONS ARE NOT CREATED EQUAL

Did you see this statement from Metalink note # 400010.1 - Steps to Maintain Oracle Database 10.2 Companion CD Home (for Oracle HTTP Server)?

“Something to think about...”

The Oracle HTTP Server delivered with the Oracle Database 10.2 Companion CD is provided for demonstration purposes, primarily for HTMLDB. However, its an older version with limited functionality and support. It also installs a mix of 10.2.1 and 10.1 products which is more difficult to maintain. Consider installing a better package of the Oracle HTTP Server.“

It's certainly convenient if you need OHS to install the version that comes on the database companion CD. However, you should not do it. The main versions of OHS come packaged with the Application Server. Oracle does not recommend that you use this version, so don't! If you choose to use a standalone version of OHS take the extra step of downloading the version that comes with 10gAS. If you decide to go ahead and use the database version of OHS, please remember that OHS patches are not included in database patchsets. When you apply a patchset to your database home, you must separately apply any OHS patches.

BASIC ORACLE HTTP SERVER (OHS) HARDENING

So you've done your homework and have determined the appropriate version of OHS to run in your environment. This section of the paper covers the minimum you must do to secure that OHS implementation; however, before you crank up the *Oracle Universal Installer* (OUI), make sure you can answer the following question.

HOW MANY OS ACCOUNTS ARE REQUIRED TO RUN OHS?

Consider the *Two-Man Rule* or the *Four-Eye Principal* that requires two people to initiate a critical function such as launching a nuclear missile or approving a financial transaction over a certain dollar threshold. The same concept applies to operating system accounts. Wherever possible, limit the capabilities of an account by spreading responsibility across multiple accounts. In this way, compromise of a single account does not put your entire environment in jeopardy.

For OHS, then, this means we should use three accounts on the web server:

- One to own the software (ORACLE_HOME)

```
-rwx----- 1 oracle oinstall 428250 2006-09-18 16:16 ../bin/httpd
```

- One to run the software (httpd)

```
httpd.conf:
```

```
User webohs
Group webohs
```

```
ps -ef | grep httpd
```

```
root      25846  5810  27  23:19 ?          00:00:01 httpd -d
webohs    25847  25846   0  23:19 ?          00:00:00 httpd -d
```

- One to own the content (htdocs)

```
bjm-desktop:/ohs/htdocs#
```

```
drwxr-xr-x 2 webdoc webdoc    4096 2009-03-09 23:20 webcontent
```

```
drwx----- 2 oracle oinstall  4096 2009-01-25 18:22 win
```

Now that you've created those three OS accounts, go head and crank up OUI.

SECURE HTTPD.CONF CONFIGURATION

CIS APACHE BENCHMARK

The Center for Internet Security (CIS) (see reference section) produces a number of benchmarking tools to verify configurations against standard industry best practices. We use the Apache benchmarking tool to evaluate the setup we have and identify any variations. The tool works by running a check against the configuration files. It does not probe the actual web site. We end up using both kinds of tools. The configuration benchmarking tool is a first step to producing a hardened configuration. Once we have the basic setup in place we run scanning tools that perform actual tests against the site. Using both sets of tools provides better coverage to verify that we have addressed known vulnerabilities.

Let's start the process by running the CIS benchmark against the base OHS 2 install and see what we get.

```
#===== [ CIS Apache Benchmark Scoring Tool 2.10 ] =====#
[Section 1.14] Web Server Software Obfuscation General Directives
  [FAILED]      ServerSignature is "On"
[Section 1.18] Access Control Directives
  [PASSED]      Directory entry for "/" is properly configured. allowoverride None
  [FAILED]      Directory entry for "/" is not properly configured. options FollowSymLinks
  [FAILED]      Directive "deny" Directory entry for "/" is not defined.
[Section 1.20] Directory Functionality/Features Directives
  [FAILED]      Did not disable Option directive "Includes" for DocumentRoot
[Section 1.21] Limiting HTTP Request Methods
  [FAILED]      There is no LimitExcept directive for DocumentRoot
[Section 1.23] Remove Default/Unneeded Apache Files
  [VERIFY]      Verify DocumentRoot files are not default Apache files.
...
[Apache Benchmark Score]: 2.79 out of 10.00]
```

The overall score is not that good. There are a number of things that can be done to improve this score. The next several sections of this document will go through configuration changes that lock down OHS. We will rerun the benchmark after the changes have been made to see the final score.

FINGERPRINTING

Fingerprinting the web server is a technique to identify the details of a web server configuration. If the web server configuration is known, then an attacker can select the known vulnerabilities for that configuration and immediately probe for an opening with the most likely exploits. Hiding the details of the configuration means that attacks must use a wide range of probes to find vulnerabilities. This provides you with additional time to detect the attack and respond. It's a red flag when you start seeing requests come in for technology that you do not use. We have had "friendly" scans done against the environment and received false positive results that we were vulnerable to an IIS exploit. The scanner had incorrectly identified the web server, and thought we were running an old version of IIS with a known vulnerability.

There are a number of places where information is revealed about the server configuration. The obvious place is the HTTP Header listing the web server version and turning this off is one of a number of steps you should take. However, there are other places, such as error pages, where configuration information will be displayed unless you modify the default setup. The fingerprinting techniques have become much more sophisticated with scanners detecting subtle differences in the way a particular web server responds to a bad request, the order of HTTP headers, and the format of certain responses. Closing these information leaks is one of the first steps you can take.

To start this process we can use a tool, called HTTPPrint (<http://www.net-square.com/httpprint/>) to fingerprint the default installation. We will then modify the server configuration and obscure the information leaks to see if we can get this tool to mis-identify the web server.

We downloaded the tool and ran it against a fresh install of OHS 2.0 Standalone. The tool launches a number of tests against the site to see how it responds and compares the results to a set of signatures for a variety of web servers. Here is a list of some of the requests that the tool sends:

```
192.168.0.10 - - [01/Mar/2009:11:24:09 -0500] "PUT / HTTP/1.0" 403 160
192.168.0.10 - - [01/Mar/2009:11:24:09 -0500] "JUNKMETHOD / HTTP/1.0" 403 160
192.168.0.10 - - [01/Mar/2009:11:24:09 -0500] "GET / JUNK/1.0" 403 160
192.168.0.10 - - [01/Mar/2009:11:24:09 -0500] "get / http/1.0" 403 160
```

The tool produces a report with a confidence rating to indicate which web server is being run. While looking at the server header is one check, the tool takes into account that the server header could have been modified to another value or turned off. The header is only one of many checks being done. If you look at the types of requests, you will see invalid methods and protocols being used to detect the type of response to each case. The report that was generated from the baseline run shows Apache 2.0 with a high degree of confidence.



If we examine the HTTP headers and default error pages we see that OHS is displaying a number of configuration details.

Basic Header:

```
HEAD / HTTP/1.0
HTTP/1.1 200 OK
Date: Mon, 23 Feb 2009 02:19:58 GMT
Server: Oracle-Application-Server-10g/10.1.3.1.0 Oracle-HTTP-Server
```

Error Page:

```
<body>
<h1>Not Found</h1>
<p>The requested URL /notfound was not found on this server.</p>
<hr>
<address>Oracle-Application-Server-10g/10.1.3.1.0 Oracle-HTTP-Server Server at bjm-desktop
Port 80</address>
</body>
```

STOP LEAKING CONFIGURATION INFORMATION

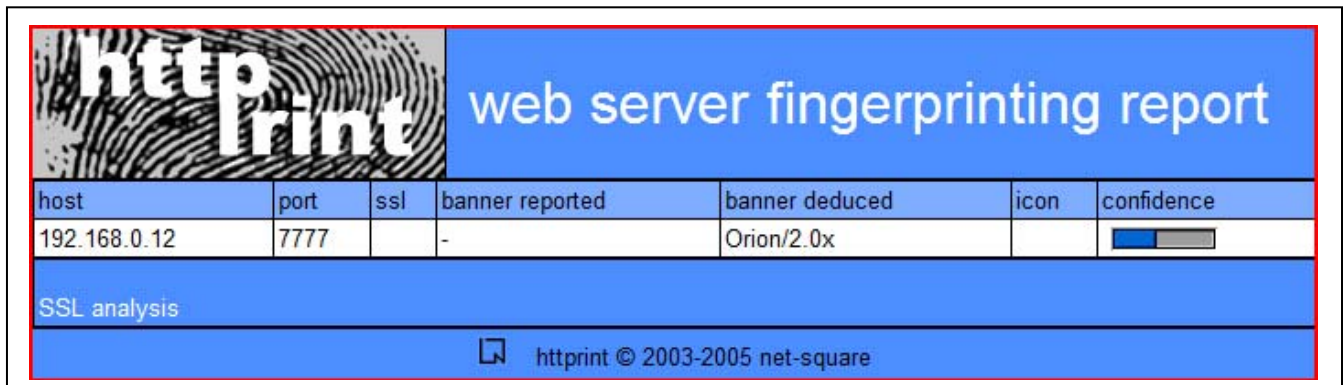
To stop this information from leaking we are going to make several configuration changes. First, we will turn off Server Headers and Tokens to eliminate obvious places where the information is being shown. We will eliminate methods that are not required to be used and that can reveal the type of web server. We will replace error pages with custom versions that provide only the basic response information. Finally, we will insert some fake headers that will modify the order of the headers that are sent and that may confuse a fingerprinter into thinking we are running some other version of software. The following table highlights the changes:

Table 1 - Configuration changes for server information

Original Configuration	Revised Configuration
ServerAdmin you@example.com	###ServerAdmin you@example.com
ServerName bjm-desktop	ServerName ohs.collaborate09.org
ServerTokens Minimal	ServerTokens Prod
	Note that <i>ServerTokens Prod</i> get you a lower CIS score but Oracle provides the option of <i>ServerTokens None</i> . You should use the Oracle-provided <i>None</i> for the highest security.

Original Configuration	Revised Configuration
No Limits on GET and POST or OPTIONS	<pre><LimitExcept GET POST> deny from all </LimitExcept> Options None</pre>
Using default error pages	<pre>ErrorDocument 403 /error_contactus.htm ErrorDocument 500 "There was an error processing your request, please retry."</pre>
No fake headers	<pre>Header onsuccess set X-Powered-By "ASP.NET"</pre>

After implementing these changes we reran the HTTPPrint tool. The tool now reports that the site is running Orion, but the confidence level is low. Based on this information an attack might be launched using a known vulnerability of the Orion 2 web server, such as the JSP source disclosure vulnerability (<http://secunia.com/advisories/18950/>). We are not running Orion, so we are not vulnerable to this attack, but we could insert a rule to look for someone sending a request that attempts to exploit this vulnerability and flag it because it is likely coming from someone who is using fingerprinting to probe the site.



LIMITING ACCESS TO DIRECTORIES AND FILE TYPES

The following directives were modified to restrict access to files and limit access to directories.

LIMITING ACCESS TO THE ROOT DIRECTORY

Here is an example of the default configuration:

```
# Default controls on root directory
<Directory />
    Options FollowSymLinks MultiViews
    AllowOverride None
</Directory>
```

Note that the root directory is completely unprotected by default and OHS will happily server up whatever content is there. You need to lock this down.

Here's what locked down configuration looks like:

```

#
# Each directory to which Apache has access can be configured with respect
# to which services and features are allowed and/or disabled in that
# directory (and its subdirectories).
#
# First, we configure the "default" to be a very restrictive set of
# features.
#
<Directory />
#
# The Options directive has been removed since specifying
# anything other than Options All causes 403 errors for
# all requests. Options All is the default.
#
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>

#
# Because of the issue above with Options in the Directory directive,
# we use a DirectoryMatch directive to accomplish the same thing.
#
<DirectoryMatch ^/>
    Options None
</DirectoryMatch>

#
# Note that from this point forward you must specifically allow
# particular features to be enabled - so if something's not working as
# you might expect, make sure that you have specifically enabled it
# below.
#

```

Note that we completely lock down root with *Deny from all* and later in `httpd.conf` we open up only what we need. Note also the odd issue with *Options*. For reasons unknown to us adding *Options None* to the root *Directory* directive (which is what we want to do) causes 403 errors. We got around this problem by instead specifying it with the *DirectoryMatch* directive. We have no explanation for this - just a heads up. Please contact the authors if you can explain this behavior.

LIMITING ACCESS TO THE DOCUMENT ROOT

Here is the default configuration for `htdocs`:

```

DocumentRoot "/u01/app/oracle/product/10.1.3.2/companionCDHome_1/ohs/htdocs"

<Directory "/u01/app/oracle/product/10.1.3.2/companionCDHome_1/ohs/htdocs">

    Options FollowSymLinks MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>

```

You should lock your content directory down as follows:

```

#
# This should be changed to whatever you set DocumentRoot to.
#

```

```

<Directory /ORACLE_HOME/ohs/htdocs>

#
# Allow only Get and POST methods
# within DocumentRoot
#
    <LimitExcept GET POST>
        deny from all
    </LimitExcept>

#
# The Options directive is both complicated and important. Please see
# http://httpd.apache.org/docs-2.0/mod/core.html#options
# for more information.
#
    Options None

#
# AllowOverride controls what directives may be placed in .htaccess files.
# It can be "All", "None", or any combination of the keywords:
#   Options FileInfo AuthConfig Limit
#
    AllowOverride None

#
# Controls who can get stuff from this server.
#
    Order allow,deny
    Allow from all
</Directory>

```

Note that we restrict all methods except *GET* and *POST*. In addition we set *Options* to *None*. Finally, we are allowing everyone access to content on this server. If it makes sense for your environment you can lock down access to particular subnets or IP addresses with the *Allow* directive. Remember that we have locked down the root directory so you must open DocumentRoot or all access attempts will result in a 403 error.

LIMITING ACCESS TO THE DOCUMENT ROOT

You should lock down access to CGI or perl scripts to only appropriate personnel. We do not even use CGI or perl so we don't load their corresponding load modules. Still, remembering Defense in Depth, we lock down access further since all it takes is one person to uncomment the `cgi_module` and then any such scripts are executable by anyone unless restricted as we do here. To avoid that single point of security failure we lock such scripts down as follows:

```

<LocationMatch /(cgi-bin|fcgi-bin|cgi|fastcgi/testperl\.pl|fcgi\.jar|libfcgi\.a)/>
    Order deny,allow
    deny from all
    allow from localhost
</LocationMatch>

```

Here we let the scripts execute only from localhost but you could eliminate them entirely by just removing the allow directive or saying allow from none.

LIMITING ACCESS TO BACKUP FILES

Remembering the credo of “first do no harm”, production support personnel should always backup configurations before changing them. Oracle will also make backup copies when applying patches. If these patches were security patches, that means the unpatched versions are sitting on your server and unscrupulous people will try to find and use them to exploit known vulnerabilities. Think of how should files are created in your environment and modify the directive below accordingly:

```
# Restrict access to backup files
<LocationMatch /*(\~|\.bak|\.sav|\.orig|\.old|\.20[0-9][0-9][0-1][0-9][0-3][0-9])$>
    deny from all
</LocationMatch>
```

CONTROL THOSE LOAD MODULES

One key change that you want to make is to comment out load modules that are not appropriate for your environment. The default install enables all the load modules that are shipped with OHS except for userdir and php5. We cannot tell you which load modules you should comment out because it depends on your intended use. If you have a Standalone OHS 2.0 installation being used as a reverse proxy server, then you would need mod_proxy enabled. If you are using OHS 1.3 in a 10gAS configuration with a set of J2EE containers, then you probably can disable mod_proxy. You need to evaluate which ones you really need and disable all the others. It may take a little trial and error to determine your minimum level of load modules. This is a one-time effort because the typical environment does not change that significantly once you establish your baseline. As an example, we do not use php in the environment. We disable all php load modules for all OHS configurations, since we don’t use this technology. We have filtering rules on the reverse proxy to filter out php requests. We still take the step to disable php on the application tier even though none of these requests should get through. Remember the defense in depth strategy. If someone was able to get past the php filter on the web tier, the restriction on the application tier OHS would still block execution of php.

One thing you should be aware of when doing subsequent patching of the OHS setup. A patch might re-enable a module that you disabled. You should always recheck the hardening steps to verify that they are still in place following the application of a patch. In some cases, you might have to restore the configuration prior to applying a patch because the patch may be surprised by some of the hardening steps that you have taken. You always want to keep a copy of the original configuration file in case you need to revert to the default setup. You should backup any custom configuration changes that you made to the OHS setup prior to applying a patch to prevent any of your changes from being overwritten.

The following table provides some samples of load modules that could be disabled. This list was established based on using OHS 2.0 as a reverse proxy server and using mod_security.

Table 2 - Sample configuration changes to limit load modules

Original Configuration	Revised Configuration
LoadModule file_cache_module	#####LoadModule file_cache_module
LoadModule vhost_alias_module	#####LoadModule vhost_alias_module
LoadModule env_module	LoadModule env_module
LoadModule log_config_module	LoadModule log_config_module
LoadModule mime_magic_module	#####LoadModule mime_magic_module
LoadModule mime_module	LoadModule mime_module
LoadModule negotiation_module	LoadModule negotiation_module
LoadModule status_module	LoadModule status_module
LoadModule info_module	#####LoadModule info_module
LoadModule include_module	#####LoadModule include_module
LoadModule autoindex_module	#####LoadModule autoindex_module
LoadModule dir module	#####LoadModule dir module

Original Configuration	Revised Configuration
<pre> LoadModule cgi_module LoadModule cgid_module LoadModule asis_module LoadModule imap_module LoadModule actions_module LoadModule speling_module #LoadModule userdir_module LoadModule alias_module LoadModule access_module LoadModule auth_module LoadModule auth_anon_module LoadModule auth_dbm_module LoadModule proxy_module LoadModule cern_meta_module LoadModule expires_module LoadModule headers_module LoadModule usertrack_module LoadModule unique_id_module LoadModule setenvif_module LoadModule fastcgi_module LoadModule perl_module LoadModule php4_module #LoadModule php5_module LoadModule ossl_module LoadModule rewrite_module </pre>	<pre> #####LoadModule cgi_module #####LoadModule cgid_module #####LoadModule asis_module #####LoadModule imap_module #####LoadModule actions_module #####LoadModule speling_module #####LoadModule userdir_module LoadModule alias_module LoadModule access_module #####LoadModule auth_module #####LoadModule auth_anon_module #####LoadModule auth_dbm_module LoadModule proxy_module #####LoadModule cern_meta_module LoadModule expires_module LoadModule headers_module LoadModule usertrack_module #####LoadModule unique_id_module LoadModule setenvif_module #####LoadModule fastcgi_module #####LoadModule perl_module #####LoadModule php4_module #####LoadModule php5_module LoadModule ossl_module LoadModule rewrite_module LoadModule security_module LoadModule proxy_http_module LoadModule proxy_connect_module LoadModule certheaders module </pre>

ADVANCED ORACLE HTTP (OHS) HARDENING

Now that you have performed basic OHS hardening, it's time to consider the more advanced options of *Reverse Proxy* and *mod_security*.

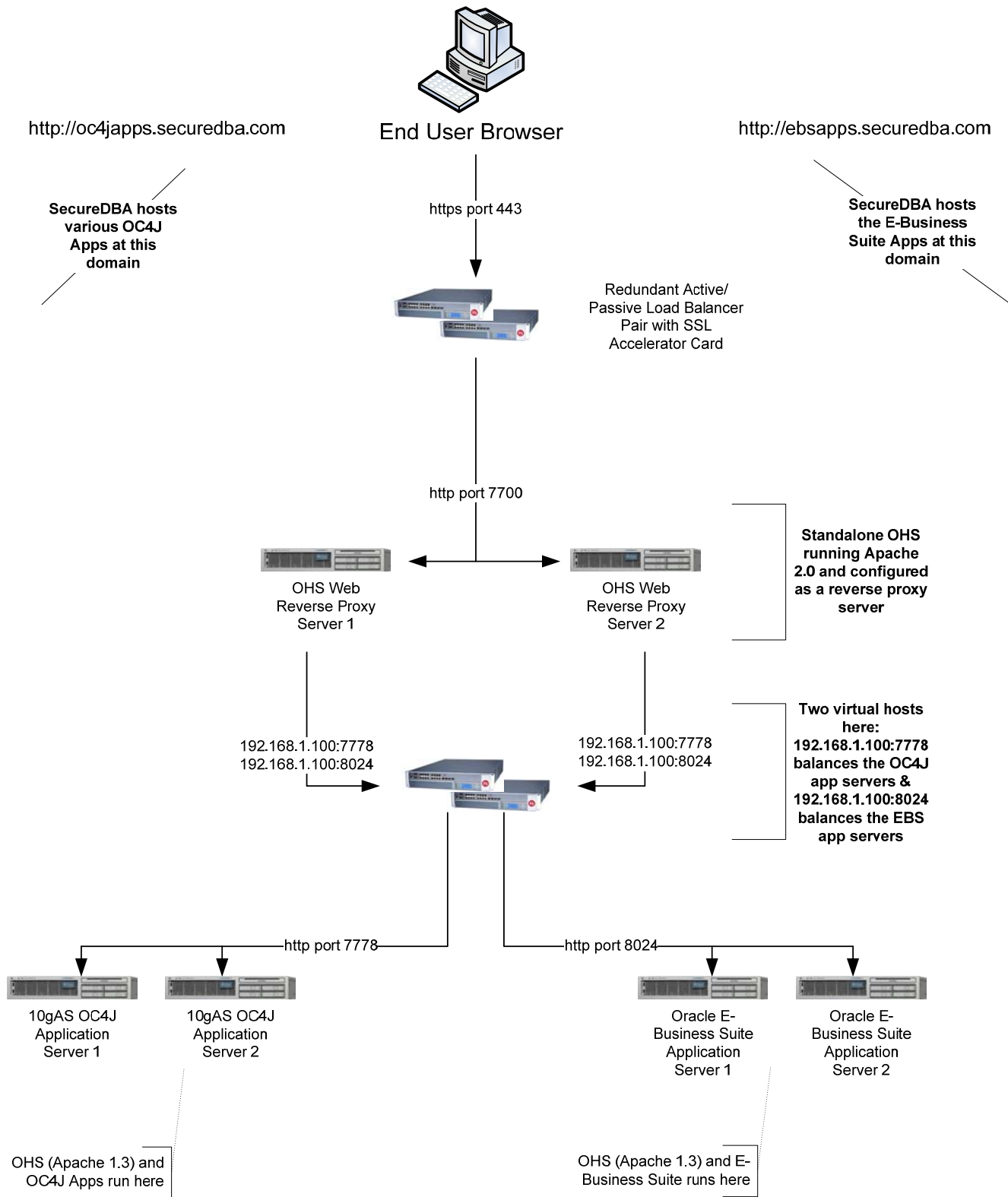
CONFIGURING OHS AS A REVERSE PROXY SERVER

First of all, just what is a reverse proxy server and why should you consider one. For our purposes, a reverse proxy server is an instance of OHS that:

- takes an inbound HTTP request and forwards it to your web servers thus providing a layer of obfuscation;
- based on rules you define, either passes (proxies) a request onward or denies it access and therefore you can configure if to limit probes by individuals trying to fingerprint your environment;
- can serve up static content to take some load off of your web/application servers;
- can act as a server-side cache; and
- can compress content

For the purposes of this paper, we are only going to cover the first two bullets.

EXAMPLE ENVIRONMENT



Assume that SecureDBA, LLC hosts both OC4J applications and E-Business Suite Applications on separate pools of application servers both fronted by the same pair of OHS Reverse Proxy Servers. Our goal is to proxy valid traffic forward to the correct pool of application servers and deny any traffic that does not represent a uri hosted by SecureDBA's applications. There are two *Virtual IPs* that live on the second pair (application tier) of load balancers:

- 192.168.1.100:7778 traffic will be forwarded to the OC4J application server pool by the load balancer
- 192.168.1.100:8024 traffic will be forwarded to the EBS application server pool by the load balancer

Therefore, our job is to setup the reverse proxy server to pass OC4J traffic to 192.168.1.100:7778 and EBS traffic to 192.168.1.100:8024 and deny all other traffic. The load balancer will take care of the rest.

VIRTUAL HOST SETUP IN HTTPD.CONF

It is not really required but it is a best practice to set up virtual hosts in OHS for every website you host. This allows you to have separate configurations within OHS that are specific to that host. We'll be using different reverse proxy directives for these two websites so we'll setup virtual hosts in the reverse proxy server. The pertinent snippets of the httpd.conf configuration would look something like this:

```
...
...
...
Listen 7700
...
...
...
NameVirtualHost *:7700

include "/ORACLE_HOME/ohs/conf/secureDBA_virtualhost.conf"
```

Note that we put very little detail in the httpd.conf file. Keep it simple and put your environment specific details in include files instead. Next we look at the secureDBA_virtualhost.conf file which defines the virtual host for the two site supported by SecureDBA:

```
### This is the VirtualHost file that includes for SecureDBA's sites ###
<VirtualHost *:7700>
    ServerName oc4japps.securedba.com:80
    <Proxy *>
        Order deny,allow
        Deny from all
        Allow from all
    </Proxy>
    include "/ORACLE_HOME/ohs/conf/oc4j_proxy.conf"
</VirtualHost>

<VirtualHost *:7700>
    ServerName ebsapps.securedba.com:80
    <Proxy *>
        Order deny,allow
        Deny from all
        Allow from all
    </Proxy>
    include "/ORACLE_HOME/ohs/conf/ebs_proxy.conf"
</VirtualHost>
```

So here we see our two virtual hosts. The *Proxy* * is simply a container for the directives applied to this particular proxy. This one is not particularly interesting as we are allowing everyone access to everything. You can add more granular access control by replacing the * with a location (such as /my_uri) and then allow only specific IPs or subnets to proxy to that location.

Finally, we'll look at the two *.proxy.conf files which are at the heart of the reverse proxy configuration.

REVERSE PROXY SETUP

Let's first take a look at oc4j_proxy.conf:

```
### This is for proxy config for OC4J ###
<IfModule mod_proxy.c>
    ProxyRequests Off
    ProxyPreserveHost Off
    ProxyTimeout 300

ProxyPass /oc4j_mountpoint_1 http://192.168.1.100:7778/oc4j\_mountpoint\_1
ProxyPassReverse /oc4j_mountpoint_1 http://192.168.1.100:7778/oc4j\_mountpoint\_1

ProxyPass /oc4j_mountpoint_2 http://192.168.1.100:7778/oc4j\_mountpoint\_2
ProxyPassReverse /oc4j_mountpoint_2 http://192.168.1.100:7778/oc4j\_mountpoint\_2

</IfModule>
```

The *ProxyRequests* directive is critical. If set to *On*, it allows your reverse proxy server to act as a forward proxy. Given that we have not blocked any traffic and assuming this is an internet-hosted site, turning *ProxyRequests On* means you are now an open forward proxy server on the internet. Needless to say this is very bad from both a performance impact on your server as well as a security issue for the internet at-large. The default is *Off*, but we like to explicitly set it anyway.

ProxyPreserveHost Off means to pass the hostname the *ProxyPass* directive (192.168.1.100) to the backend server and not the hostname from the original request (oc4japps.securedba.com). If you are using name-based virtual hosts on the application tier then you should turn this on.

ProxyTimeout 300 will gracefully end requests that take longer than 300 seconds to get a response from the application server. 300 is the default.

The *ProxyPass* directive is used to proxy requests for a specific location to a backend server. Here the locations are simply two OC4J containers and the backend server is the load balanced VIP for the OC4J application servers.

ProxyPassReverse is needed to ensure that self-referencing redirects on the application tier do not bypass the reverse proxy. That is, with this directive, a redirect on the application server to http://192.168.1.100/oc4j_mountpoint_2/image1.gif will get redirected to http://oc4japps.securedba.com/oc4j_mountpoint_2/image1.gif. Without this directive, the redirect will bypass the reverse proxy and go directly to http://192.168.1.100/oc4j_mountpoint_2/image1.gif.

Note that requests for anything other than the two OC4J mount points will fail with a 404 since they will not be proxied and they presumably have no matching content in htdocs. This is a good way to stop probing of your site with invalid URIs.

Finally let's look at ebs_proxy.conf:

```
### This is for EBS proxy
<IfModule mod_proxy.c>
    ProxyRequests Off
    ProxyPreserveHost Off
    ProxyTimeout 300

ProxyPass /OA_MEDIA http://192.168.1.100:8024/OA\_MEDIA
ProxyPassReverse /OA_MEDIA http://192.168.1.100:8024/OA\_MEDIA
```

```

ProxyPass /OA_TEMP http://192.168.1.100:8024/OA_TEMP
ProxyPassReverse /OA_TEMP http://192.168.1.100:8024/OA_TEMP

ProxyPass /OA_HTML http://192.168.1.100:8024/OA_HTML
ProxyPassReverse /OA_HTML http://192.168.1.100:8024/OA_HTML

ProxyPass /OA_SECURE http://192.168.1.100:8024/OA_SECURE
ProxyPassReverse /OA_SECURE http://192.168.1.100:8024/OA_SECURE

ProxyPass /media http://192.168.1.100:8024/media
ProxyPassReverse /media http://192.168.1.100:8024/media

ProxyPass /OA_JAVA http://192.168.1.100:8024/OA_JAVA
ProxyPassReverse /OA_JAVA http://192.168.1.100:8024/OA_JAVA

ProxyPass /oa_servlets http://192.168.1.100:8024/oa_servlets
ProxyPassReverse /oa_servlets http://192.168.1.100:8024/oa_servlets

ProxyPass /servlet http://192.168.1.100:8024/servlet
ProxyPassReverse /servlet http://192.168.1.100:8024/servlet

ProxyPass /servlets http://192.168.1.100:8024/servlets
ProxyPassReverse /servlets http://192.168.1.100:8024/servlets

ProxyPass /html http://192.168.1.100:8024/html
ProxyPassReverse /html http://192.168.1.100:8024/html

ProxyPass /OA_CGI http://192.168.1.100:8024/OA_CGI
ProxyPassReverse /OA_CGI http://192.168.1.100:8024/OA_CGI

ProxyPass /CACHE http://192.168.1.100:8024/CACHE
ProxyPassReverse /CACHE http://192.168.1.100:8024/CACHE

ProxyPass /pls http://192.168.1.100:8024/pls
ProxyPassReverse /pls http://192.168.1.100:8024/pls

ProxyPass /OA_JAVA http://192.168.1.100:8024/OA_JAVA
ProxyPassReverse /OA_JAVA http://192.168.1.100:8024/OA_JAVA

</IfModule>

```

Very similar to the OC4J proxy. The only difference is now we use the other VIP listening on 8024 to get to the EBS pool and of course, the URIs here are some of those found in the E-Business Suite.

MOD_REWRITE AND MOD_SECURITY

Two of the standard tools that we use to filter URLs are mod_rewrite and mod_security. Both of these modules are part of the standard OHS distribution, which means Oracle provides support. Oracle has a number of Metalink notes regarding use of these tools. Please see the reference section for links to both Oracle notes about these modules and general industry references.

Mod_rewrite was designed to be a tool for URL rewriting. It uses a set of rules based on regular expression matching to check the incoming request URL and perform an action that modifies the request. This provides the capability to block requests by redirecting the incoming request to an error page. Mod_security was designed to be a web application firewall tool. It uses a set of rules based on regular expression matching to check the incoming request URL and perform an action that modifies the request. There is certainly some overlap between the two tools. This summary provides a quick comparison between the two.

Table 3 - Comparing mod_security and mod_rewrite

mod_security	mod_rewrite
Pro	Pro
Availability of Rules Detailed logging Designed as a security tool	Typically already in use Good for simple blocking Performance
Con	Con
New module to maintain Parsing adds overhead OHS uses old 1.84 version	More work to code rules Logging more for debug Not designed for security

Using either tool for blocking bad requests will produce the same output to the end user. As an example, if we could setup a rule to block the PUT method then the rule would redirect the user to an error page. Below is an example of the format of the rule used for each tool to accomplish this.

Table 4 - mod_security vs mod_rewrite rule formats

mod_security	mod_rewrite
SecFilterSelective REQUEST_METHOD "PUT" "id:888000,deny,log,status:405"	RewriteCond %{REQUEST_METHOD} ^PUT RewriteRule .* - [F]

One of the primary differences between the two tools is logging. Mod_security includes more extensive logging that is useful to security personnel that are tracking and investigating incidents. Mod_rewrite uses logging more for debugging. Another significant difference is that mod_security supports a pass, log action. There are a number of gray issues when you are evaluating whether or not a request is valid. The pass, log action allows you to log the request when it matches the rule, but does not block or modify the request. This approach provides a method to evaluate a rule without impacting an application. The following table illustrates the difference in logging between the two tools.

Table 5 - mod_security vs mod_rewrite logging

mod_security	mod_rewrite
<pre>===== UNIQUE_ID: QBywrH8AAQEAAHrxXvwAAAAA Request: 127.0.0.1 - - [12/Mar/2009:20:33:32 --0400] "PUT /test HTTP/1.0" 405 339 Handler: (null) ----- PUT /test HTTP/1.0 mod_security-message: Access denied with code 405. Pattern match "PUT" at REQUEST_METHOD mod_security-action: 405 HTTP/1.0 405 Method Not Allowed Allow: TRACE Content-Length: 339 Connection: close Content-Type: text/html; charset=iso- 8859-1</pre>	<pre>127.0.0.1 - - [12/Mar/2009:20:38:12 -- 0400] [bjm- desktop/sid#8fe7e88] [rid#95d97c8/initial] (2) init rewrite engine with requested uri /test 127.0.0.1 - - [12/Mar/2009:20:38:12 -- 0400] [bjm- desktop/sid#8fe7e88] [rid#95d97c8/initial] (3) applying pattern '.*' to uri '/test' 127.0.0.1 - - [12/Mar/2009:20:38:12 -- 0400] [bjm- desktop/sid#8fe7e88] [rid#95d97c8/initial] (4) RewriteCond: input='PUT' pattern='^PUT' => matched 127.0.0.1 - - [12/Mar/2009:20:38:12 -- 0400] [bjm- desktop/sid#8fe7e88] [rid#95d97c8/initial] (2) forcing '/test' to be forbidden</pre>

We initially started using mod_rewrite rules to block obviously invalid request URLs. As we evolved and needed more rules and the ability to deal with these gray issues we implemented mod_security. We use both tools, with mod_rewrite handling the chore of rewriting request URLs and mod_security handling filtering. If all you need is basic blocking of invalid requests, you may find that mod_rewrite is sufficient. Anyone running the E-Business suite may have noticed that Oracle distributes configuration files that use both tools. The security.conf file contains mod_security rules, while the url_fw.conf file contains mod_rewrite rules.

If you decide to use mod_security you can download free rule sets or purchase more up-to-date rule sets from vendors that provide support. It makes sense to structure your rule sets into categories to better organize the rules and make them easier to maintain. We have one rule set file containing just the list of user agents. In this file we initially test for the user agents that are typical for the site and that we recommend be used. We allow requests with these user agents to pass through and skip the remaining user agent checks. Requests that do not come from supported user agents are filtered through a list of known bots, spiders, and scanners. We block those user agents that are not allowed. The request is then evaluated by additional rule sets. We have some application specific rule set files, and files that differ between Intranet-hosted and Internet-hosted sites.

One final point concerns the HTTP error code that is returned when a request is blocked. Some of the documentation mentions returning a code 500 – Internal Server Error when blocking requests. This can be confusing for the Operations team because they associate a 500 status with a system problem and set monitoring alerts when these get triggered. We use the status code 400 – Bad Request for a majority of the mod_security rules. This indicates that the request input was unacceptable. We use this status to indicate that the request input did not pass, however if the client modifies the request it might be acceptable. If the user enters a request with the text “alter user sys identified by”, maybe it was a typo. We will still flag it and log the request, but we want to distinguish this from obviously bad requests. We use another status code to indicate that requests are always prohibited. If a user sends a request using the Trace method, then a 405 – Invalid method is returned. Other obvious errors might return a 406 – Not Acceptable. Oracle uses the 410 – Gone code for the mod_rewrite rules implemented in EBS. Some of the security documentation suggests returning just the 404 – Not Found status for all situations, so you are not revealing any additional information. Returning this status can confuse Operations and generate more support calls. We avoid using the 403 – Forbidden status in mod_security rules because this status is generated from the main configuration rules. If we get a report of a problem where a valid request is being denied, the status code helps to identify where the rule comes from and which log to check. This approach is useful when looking at web site summary statistics by error code too.

LOGGING

The default access log format for OHS is the common log format. This provides a very basic level of information, which is suitable for hit counts and standard web site reporting. We used this format for a period of time, and produced end user reports from the information. A typical common log entry looks like this:

```
192.168.0.10 - - [23/Feb/2009:21:45:58 -0500] "GET /index.html HTTP/1.1" 200 14679
```

We also use the extended format which includes the referring URL and the browser type. These additional fields provide additional detail for end user reporting. They also provide detail that can be used for investigation. The user agent field frequently contains the name of a bot, spider, or scanner. Certainly hackers can modify the user agent or fake running as another agent, but many do not bother to do this. The referrer can assist in tracking the path through the site or references from another site. The extended format would look like this:

```
192.168.0.10 - - [10/Mar/2009:21:23:17 -0400] "GET /index.html HTTP/1.1" 200 14679
"http://192.168.0.12:7777/OHSDemos.htm" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT
6.0; GTB5; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.5.30729; .NET CLR
3.0.30618) "
```

Several years ago, Chris Josephes wrote an article called "*Profiling LAMP Applications with Apache's Blackbox Logs*". Chris recommended including several additional fields in the access log. This provided detail that could be used for performance tracking and assessment. We used this "blackbox" format as the basis for the logs that we use. We did make a couple of modifications to include a few extras. The original blackbox format had more of a focus on performance and we wanted to add some security related items. The applications we support use session cookies to maintain user sessions. One of the potential exploits is capturing session cookies from cross site scripting vulnerabilities. We log the user cookies in the access log, which provides access to the session keys and makes it easy to spot anomalies between IP addresses and sessions. Since we run an OHS reverse proxy, we want to record the original client IP address passed from the proxy in addition to the proxy entry point. The OHS configuration directive looks like this:

```
LogFormat "%h %l %u %t \"%r\" %>s %B \"%{Referer}i\" \"%{User-Agent}i\" \"%{X-FORWARDED-
FOR}i\" \"%{cookie}i\" %v %X %P %T" blackbox
CustomLog /u01/app/oracle/product/10.1.3.2/companionCDHome_1/ohs/logs/access_log blackbox
```

The fields used in this format maintain the same order as the common log format for the start of the file. This allows standard reporting tools to be used against the log because the tools just ignore the extra fields.

Table 6 - Blackbox Log Parameters

Log Parameter	Definition
"%h	Source IP address
%l	Identity of the client
%u	User ID from the remote_user variable
%t	Time request completed
\"%r\"	URL request line
%>s	HTTP status code
%B	Size of object in bytes sent in response
\"%{Referer}i\"	Referring URL
\"%{User-Agent}i\"	Client User Agent
\"%{X-FORWARDED-FOR}i\"	Client IP for reverse proxy
\"%{cookie}i\"	Cookies including session cookies
%v	Virtual host name
%X	Connection Status
%P	Process id and Thread id
%T"	Time to Serve Request

The access log entry for a request in this format looks like this:

```
192.168.0.10 - - [10/Mar/2009:21:23:17 -0400] "GET /index.html HTTP/1.1" 200 14679
"http://192.168.0.12:7777/OHSDemos.htm" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT
6.0; GTB5; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.5.30729; .NET CLR
3.0.30618)" "10.0.0.100" "JSESSIONID=8EEEE08C4DEFF1B72F9BCCEC72B58544" bjm-desktop + 27860
0
```

This format has helped resolve problems and provided the detail we need to investigate issues. We have been able to track user activity through the site that led to errors. We have used the cookie information to spot problems with sticky sessions when the user session was incorrectly routed to the wrong app tier. We have found cases where the sessions and IP addresses did not match up. We use inexpensive off the shelf tools to analyze the access logs for end user reporting, troubleshooting, and log analysis. For end user style reporting we use WebLog Expert (<http://www.weblogexpert.com/>).

This produces an attractive output in pdf format that is easy to distribute. You can download a free lite version or purchase a full featured version. We have used a simple perl script to perform a quick analysis of current activity to see which URLs are generating 500 errors. The find_status.pl script, available at: http://perl.apache.org/docs/tutorials/tips/mod_perl_tricks/mod_perl_tricks.html#Log_Parsing

This is one of a number of handy perl tools that can be used for analysis. Another quick analysis can be done with grep, such as finding transactions that took more than 5 seconds by checking the time field in the last column (grep -v [012345]\$ access_log). We have used Log Parser from Microsoft to perform summaries: (<http://www.microsoft.com/technet/scriptcenter/tools/logparser/default.msp>).

This is a useful tool that allows SQL queries against flat files. It does not require a database instance to execute.

A SHORT CASE STUDY

The best way to illustrate how we use the techniques in this paper is to walk through a short case study. We had an incident where the operations team reported some unusual activity in production and reported it to the security team. There are a number of clues that something suspicious is going on in the environment. These include:

- Big increase in 403 not authorized requests
- Big increase in 404 not found requests
- Big increase in 400 Bad Request or 406 Not Acceptable requests
- Unusual 404 pattern, not favicon.ico
- Hundreds of requests per minute off-peak
- Many requests from one IP in under a minute
- Requests for unused technology, PHP
- Non-standard user-agent

The security team reviewed the production logs and found a large number of requests from a user-agent that was identified as w3af.sourceforge.net. A Google search turned up this information:

w3af is a Web Application Attack and Audit framework. The project's goal is to create a framework to find and exploit web application vulnerabilities that is easy to use and extend.

The action that we took was to add a new mod_security rule:

```
SecFilterSelective HTTP_USER_AGENT "w3af\.sourceforge\.net"  
"id:888000,deny,log,status:406,msg:'User Agent invalid'"
```

The rule blocks access by the user agent w3af and returns a 406 Not Acceptable response. Blocked request information is logged in the mod_security log. This rule was added to the existing mod_security configuration file that contains the list of user agent blocking rules.

The original logs that showed information about the attack requests and the IP address where these requests came from was turned over to the corporate security team for further investigation.

The effort to secure the OHS configuration is an ongoing process. It needs to respond to changes in the environment as new software is deployed. It also must adapt to new types of attacks. Having a framework in place that includes a capability to block requests, log activity, and tailor rules that fit the environment provides the tools needed to minimize security risks.

THE FINAL SCORE

After making the above changes to the OHS configuration file the final CIS Apache benchmark score is:

```
[Apache Benchmark Score]: 8.14 out of 10.00]
```

We hope you were not expecting a perfect 10! This is the real world where perfect scores are not attainable. What we were looking to achieve was to improve the configuration by hardening it, and explain any differences between the benchmark checks and the configuration we implemented. Let's examine some of the categories to explain the variances.

The CIS benchmark checks for Apache load modules that are compiled into the binary. We use an Oracle provided version, so we cannot control the modules that are compiled into it. This means certain checks will not pass:

```
[Section 1.9] Configure the Apache Software
[FAILED]      Unless required, module "mod_status" should not be compiled into Apache.
```

For performance reasons we turn off hostname lookups. We can post process the access log when generating end user reports to retrieve the hostname.

```
[Section 1.11] Server Oriented General Directives
[FAILED]      HostnameLookups is off for Apache Web Server
```

We have application transactions that take more than one minute to execute. This includes Discoverer report queries and EBS transactions. We have to keep the transaction timeout at a higher value and therefore we do not meet this recommended standard.

```
[Section 1.13] Denial of Service (DoS) Protective General Directives
[FAILED]      TimeOut value "300" is greater than the recommended "60"
```

There are a number of directory permission checks in the benchmark. Some of these recommend that an OHS directory be owned by root, while others alter the permissions to read only. We have found that this can cause issues with operations managing the instance because we limit access to the root user. We believe the multiple user setup discussed earlier in the paper provides a sufficient level of separation. We relocate log files outside of the Oracle home directory, since these files can grow substantially in size. We do nightly rotation of logs and manage log files with cron jobs to compress and remove old log files. This means we do not pass these checks in the benchmarks.

```
[Section 1.24] Update Ownership and Permissions for Enhanced Security
[FAILED]      Owner of Log directory should be root.
```

Overall we are satisfied with the results we reached and the fact that we have measured the setup we are using against an independent benchmark. We recommend that you just don't take the defaults and instead make conscious choices about how to setup your OHS configuration. You may find that some of the techniques in this paper are applicable to your environment and you can use them "as is". In other cases you may decide to use these items as a starting point and then tailor the setup to something that is a better fit for your requirements.

If you follow all the techniques in this paper, and someone asks if your OHS environment is secure, you can confidently answer NO! Any system that is exposed to the Internet has some vulnerabilities. What you can say is that you have done appropriate due diligence to harden your OHS environment and minimize security risks.

TIPS AND TRICKS FOR MANAGING OHS

Oracle has a Metalink note entitled “*Everything You Wanted to Know About the Apache-Based OHS Version*” (Doc ID: 260449.1). Well there are a lot of things about OHS that are not contained in this Metalink note that you need to know. Many of the items are contained in other Metalink notes. Unless you happen to be looking specifically for these items you might overlook a lot of good information. In this section of the paper we cover tips and tricks for managing OHS. There are lots of fun facts, and some not so fun, that you should know to manage your OHS installation. There is a summary of Metalink notes at the end of this section, which we recommend you read when planning your OHS deployment.

THE BEST FEATURE OF OHS BASED ON APACHE 2 IS NOT ENABLED BY DEFAULT!

The best feature of OHS based on Apache 2 for Unix platforms is the threaded implementation called MPM Worker. The OHS 1.3 version uses a process based model for handling each incoming request. This is a heavyweight design that uses a large footprint on the server. The process based design is not optimal for today’s modern multi-core chip architectures that are optimized for threaded processing. The Apache 2 design allows either the threaded version or the process version to be used. Some applications may require the process based implementation because of plug-ins that are not thread safe. The version of OHS 2 that ships is configured by default to use processes not threads. It’s an easy change to modify OPMN settings to enable the MPM Worker configuration. The Metalink note “How To Configure Worker MPM In HTTP Server Based On Apache 2.0” (Doc ID: 299125.1) shows the steps to follow.

BUILD YOUR OWN MOAT

A common defensive strategy is to build an outer perimeter to ward off attacks. This first line of defense can halt the most common attacks and protect the vital infrastructure. One way to do this with web architecture is to setup a reverse proxy. This provides a central point to filter all incoming traffic and block any suspicious traffic. Two of the most common web attacks use Cross Site Scripting and SQL Injection to exploit vulnerabilities in applications. We have found that a lot of software does not have sufficient input validation routines to protect against these attacks. It is not always an option to wait for fixes to the code, particularly if this is a COTS product. An effective strategy to defend this insecure code is to setup a reverse proxy with mod_security and/or mod_rewrite rules enabled. This provides a way to block bad requests at the outermost point, and does not require code changes from the existing applications. The one downside is that you could potentially block valid activity, which means you will need to fine tune any rules that you implement to meet your specific environment. You may have two sets of rules, one for intranet and another for extranet, if you want to allow intranet users additional capabilities that you do not want to expose on the extranet.

LISTEN UP!

The E-Business Suite has used a design pattern called the two listener strategy for a long time. This strategy is documented in the Metalink note “*Dual OHS Listener Configuration for Mid-Tier PL/SQL in E-Business Suite 11i*” (Doc ID: 239176.1). This approach addresses the performance issue with the process version of Apache 1.3. The standard 10gAS installation includes HTTP processes that listen on the main OHS port, 7778, and additional ports for SSL, 4443, and DMS, 7200. When hardening the OHS configuration you need to be aware of all open ports and confirm that the directives you are adding apply to the entire configuration. You may need to repeat directives in different sections of the configuration to insure that the rules are applied to each listening port.

USE AN INCLUSIVE OHS CONFIGURATION

OHS supports the use of the include directive to incorporate additional configuration directives into the main configuration. A number of the Oracle products have a separate conf file with OHS configuration directives. You should use the same technique to incorporate any local rules into the configuration. Creating rules in different configuration files keeps your rules separate from the Oracle configuration rules making upgrades and patching easier. You can also create environment specific configuration files that isolate difference between dev, test, and production environments, such as IP addresses. Well organized conf files are easier to manage. Wrap the rules you create in a set of IfModule directives to test for the presence of a load module. There are syntax differences between OHS 1.3 and 2.0 for various modules. One example is mod_headers, which has additional options that are supported in version 2. Rules that are specific to a particular version can be isolated into separate conf files to allow a common compatible set of rules to be shared by both versions reducing overall maintenance.

CAN YOU USE MOD_PLSQL WITH THE OHS VERSION BASED ON APACHE 2.0?

Yes, you can. The Metalink note *"Is it Possible To Configure Mod_plsql With Apache 2.0?"* (Doc ID: 428391.1) has been updated with information about the 11g release of OHS and mod_plsql support. If you run Apex applications and are required to run mod_plsql, then you should consider this version. The threaded version of Apache 2 means a true connection pool can be used with mod_plsql. A test was run using the Apache Bench (ab) component that ships with OHS. This performs a load test by simulating end user transactions. The results were compared with a normal load of Apex users. The OHS version based on Apache 1.3 used hundreds of database connections and had a substantial impact on memory consumption on the database. The OHS version based on Apache 2.0 and configured to use the threaded MPM Worker configuration used two database connections. The major Oracle products are discontinuing use of mod_plsql, such as EBS 12 (see Doc ID: 726711.1). The mod_plsql design does not fit in well with modern web architecture and is difficult to secure. Maybe in the future Oracle will produce a servlet version of mod_plsql similar to the forms servlet that will improve the Apex product.

WHICH MODULE SHOULD I USE, MOD_SECURITY OR MOD_REWRITE?

Why not use both. The Metalink note *"DMZ Configuration with Oracle E-Business Suite 11i"* (Doc ID: 287176.1) suggests this approach:

1. mod_security - Reject obviously bad requests before anything else happens
2. mod_rewrite - Check for allowed URL before mod_proxy hands the request over to the external web tier
3. mod_proxy - Only proxy request that seem valid (have passed the 2 above filtering steps) to the external web tier

These modules were designed with different purposes in mind. The combination of both tools provides a rich set of capabilities that you can leverage. Oracle shows examples using both tools, and EBS distributes configuration files with various rules that apply to mod_rewrite and mod_security. It's not really important which module you pick. The key thing is to deploy rules that properly protect your environment. Use the tool that you are most comfortable using and that best fits your environment.

USING OHS BASED ON APACHE 2 WITH THE LATEST ORACLE PRODUCTS.

Oracle has acquired a large number of new products the last few years. The products that included integration with Apache are based primarily on Apache 2, not Apache 1.3. While Oracle stayed with Apache 1.3 the industry has moved on to Apache 2.0 and Apache 2.2. In some cases the products will not work, or will perform poorly with OHS based on Apache 1.3. We learned this lesson the hard way when deploying the Universal Content Management product based on the Stellent product. We attempted to set it up with OHS based on Apache 1.3 and it did not work. Of course, before attempting this we verified in Metalink that this was a certified configuration. We entered an SR and found out from the UCM developers that they just assumed OHS would work, since it was based on Apache, and never actually tested it. This was a new concept because in the past the definition of certified configuration meant that Oracle had tested it. If you plan to use one of the newer Oracle products you should consider using OHS based on Apache 2.

A BIT OF NOSTALGIA

Many of the Apache load modules deployed with OHS are based on much older versions of the modules. As an example, the mod_security module that is packaged with all versions of OHS is version 1.84, which was released in August 2004. The latest version of mod_security is 2.5. The format of the rule sets for mod_security changed with version 1.9 and changed again with version 2.0. There are a number of widely available rule sets for mod_security that you can download and adapt to your environment. However, many of these rules are based on the later versions of mod_security. This means it takes additional effort to convert rules designed for version 2 to the 1.8 format. This should change once the next major upgrade of the Oracle Application Server is released because it is expected to include a 2.5 version of mod_security.

VIRTUALIZATION

OHS has its own virtualization technique using the Apache virtualhosts directive. Even a basic 10gAS installation sets up a virtual host for SSL configuration. A common topology used for large scale web deployments is to terminate SSL on the load balancer and use hardware acceleration to speed up SSL performance. Even with this setup you may still use SSL in the 10gAS configuration to secure the administrative portion of the site, AS Console. Anytime you use virtual host configurations you need to make sure the security rules apply to each virtual host. The rules for load modules vary when supporting virtual host directives. You may need to explicitly set an inherit rule to make sure that rules loaded in the main configuration apply.

LOAD MODULE ORDER IS IMPORTANT

The order of Apache load modules is important and will affect the behavior of the configuration rules that you implement. This is particularly true in OHS 1.3. The modules are executed in the reverse order to the sequence listed in `httpd.conf`. You may experience issues when using combinations of `mod_rewrite` and modules like `mod_oc4j`. Simply changing the sequence of the `LoadModule` directives may fix the issue, see Doc ID: 403585.1. This is not as much an issue with later versions of OHS based on Apache 2 because Apache provided an option for modules to declare the phase when they should be invoked. You may still see issues when you mix a variety of modules together including `mod_proxy`, `mod_security`, and `mod_rewrite`. If you are not getting the results you expect, check the module sequence to see if that is an issue. Setting the OHS `LogLevel` to `debug` may help or setting explicit debug options for these modules can be helpful in debugging virtual host setups and URL rewrite rules.

TEST THOSE CHANGES

The techniques outlined in this paper require a number of changes to the OHS configuration. You may find it easier to modify the OHS configuration files directly rather than using Application Server Control to modify the entries. We typically make a change and then use the `apachectl configtest` option to verify the configuration before enabling it. Just ignore the warning you get when using this tool: *“WARNING!! Direct use of apachectl within Oracle9iAS is deprecated.”* The option to test the Apache configuration is not part of OPMN. If the configuration change is valid, you will get a message: “Syntax OK”. Once you have confirmed the syntax you can execute a graceful restart of the OHS process with the command `opmnctl restartproc type=ohs`. The graceful restart allows existing requests to complete, and then reads the updated configuration to pick up the changes. Remember just because the syntax is valid does not mean the change you made will work. To test the change after restarting OHS you can execute a simple telnet test simulating a browser request, see below. If you are using 10gAS 10.1.2 don't forget to execute `dcmctl updateconfig` after modifying any OHS configuration files.

```
telnet localhost 7777
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
TRACE / HTTP/1.0
```

```
HTTP/1.1 403 Forbidden
```

NEED A LITTLE CACHE?

Whether concerned about privacy, performance or both, web site administrators have several means at their disposal to manipulate client-side caching behavior. In particular, there are two Apache modules, `mod_expires` and `mod_headers` that allow administrators to modify HTTP response headers in order to control browser caching. Let's walk through some examples showing coarse and fine-grain control.

MOD_EXPIRES MODULE

This Apache module controls the generation of Expires headers which tell browsers how long a document should be cached locally based either on access or modification time.

EXPIRESACTIVE DIRECTIVE

The first step is to make sure that Expires headers will be generated. To ensure they are, you must activate them as in the following example:

```
<IfModule mod_expires.c>
    ExpiresActive on
</IfModule>
```

Alternatively,

```
ExpiresActive off
```

will turn off Expires header generation.

The `ExpiresActive` directive can be specified anywhere within the Apache configuration hierarchy from server level down through virtual host, directory or `.htaccess` file. Lower level directives will override higher level directives so that, for instance, an `ExpiresActive` directive at the virtual host level will override the server level directive within that virtual host. Note that all the `mod_expires` directives operate within the Apache configuration hierarchy in this same way, with lower level directives overriding higher level directives.

Now that Expires headers are enabled, you must specify rules for generating them. This is done with the `ExpiresDefault` and `ExpiresByType` directives.

EXPIRES SYNTAX

Documents can be expired based on the amount of time elapsed from either a) the last time the client accessed the document; or b) the last time the server modified the document.

There are two different syntaxes available to specify expiration. The first uses a code ('A' or 'M') to specify one of the two types above (access or modification); followed by the number of seconds to add to the base time to calculate the expiration time. For example:

- `A604800` means expire the document 1 week after access; while
- `M604800` means expire the document 1 week after it is modified.

It is important to repeat that the “by access” method is determined at the client side while the “by modification” method is determined at the server side. Care should be used when specifying the “by modification” method since all client caches will expire simultaneously and once expired, will remain expired, and therefore not cached, until the next modification. The best use case for the “by modification” method is for documents that change on a regular basis such as a weekly status report or a quarterly financial report. The “by access” method is far more commonly used.

For the mathematically challenged (“How many seconds are there in month?”) among us there is an alternative syntax that is somewhat easier to use. This second syntax is “<base> plus <number> <type>” and is best explained by example.

Transforming the above 1 week examples into this for format we get either:

- “access plus 604800 seconds”

- “modification plus 604800 seconds”

or, more simply

- “access plus 1 week”
- “modification plus 1 week”

With the syntax in hand let’s get back to our example where we’ll use the more user-friendly syntax above.

EXPIRESDEFAULT DIRECTIVE

The ExpiresDefault directive is used to set the default expiration time for all documents within the realm of the directive unless otherwise overridden by an ExpiresByType directive.

```
<IfModule mod_expires.c>
  ExpiresActive on
  ExpiresDefault "access plus 4 hours"
</IfModule>
```

EXPIRESBYTYPE DIRECTIVE

Use the ExpiresByType directive to override ExpiresDefault by type of document thus gaining a finer grain control of caching behavior.

```
<IfModule mod_expires.c>
  ExpiresActive on
  ExpiresDefault "access plus 4 hours"
  ExpiresByType image/gif "access plus 1 month"
  ExpiresByType image/jpeg "access plus 1 month"
  ExpiresByType image/tiff "access plus 1 month"
  ExpiresByType image/bmp "access plus 1 month"
  ExpiresByType text/css "access plus 1 hour"
  ExpiresByType application/x-javascript "access plus 1 hour"
</IfModule>
```

Here we are expiring images 1 month after they are accessed and style sheets and javascript 1 hour after access.

MOD_HEADERS MODULE

So far we have activated Expires headers, set default expiration for all documents and overridden the default expiration for some document types. The unset, set and append options of the Header directive within the mod_headers module when used in conjunction with the FilesMatch directive, allows us to override our ExpiresByType directives to give us the finest grain cache control possible. Consider the following two use cases.

Let’s assume our working example was specified at the server level, meaning it applies to all virtual hosts on the server. Let’s further assume that we have one web site that handles sensitive content that we prefer not to cache on client machines. Assuming that site has its own virtual host we can specify the following directives within that site’s virtual host:

```
<FilesMatch "\.(html?|gif|jpe?g|jsp?|css|js|pdf)$">
  Header unset Cache-Control
  Header append Cache-Control "private, no-cache, no-store"
  Header append Pragma "no-cache"
</FilesMatch>
```

These directives unset any previous cache control headers and append new headers indicating not to cache the selected file types.

Finally, within this same virtual host, there is a particularly large file that is frequently downloaded and you want to override the above no-cache directives just for this file. The following directives will accomplish this:

```
<FilesMatch "myLargeFile\.pdf">  
    Header unset Cache-Control  
    Header unset Pragma  
    Header append Cache-Control "public"  
</FilesMatch>
```

ISSUES WITH ORACLE SINGLE SIGN-ON (SSO) AND ORACLE CONTAINERS FOR J2EE (OC4J)

There are a couple of common issues that come up with using SSO and or OC4J together with OHS. This section covers two of the most common.

ORDER OF LOAD MODULES

The proper order of load modules is critical for the functioning of SSO and OC4J. Specifically, the SSO modules must be loaded before the OC4J module which in turn must be loaded before mod_rewrite. The following example shows the proper configuration:

```
# Include the configuration files needed for mod_osso
include "/OH/ohs/conf/mod_osso.conf"
include "/OH/ohs/conf/sso_apache.conf"

# Include the configuration files needed for mod_oc4j
# Loading mod_oc4j.conf here because it must load AFTER mod_osso.conf
include "/OH/ohs/conf/mod_oc4j.conf"
# Loading mod_rewrite module here as it has to load after mod_oc4j
LoadModule rewrite_module modules/mod_rewrite.so
```

This issue is sometimes further confused by the default configuration that some version of OHS provide. See one of those below:

```
# Include the mod_oc4j configuration file
include "d:\infra904\Apache\Apache\conf\mod_oc4j.conf"
# Include the mod_dms configuration file
include "d:\infra904\Apache\Apache\conf\dms.conf"
# Loading rewrite_module here so it loads before mod_oc4j
LoadModule rewrite_module modules/ApacheModuleRewrite.dll
```

Note the confusing last comment above. It says that rewrite must be BEFORE OC4J but it actually puts it after OC4J. The comment is not correct.

SSO, PRAGMA "NO-CACHE", AND MISBEHAVING BROWSERS

SSO by default will always prevent client side caching of SSO-protected content. This is a good security practice since it prevents information fragments from being cached on local machines. However, with some versions of Internet Explorer (IE), this behavior can cause problems.

If you use SSO and IE and your application allows the download of Word, Adobe or other documents that IE reads via plugin, users will sometimes get the following error when attempting to open the document:

```
"Internet Explorer cannot download xxxx.pdf from yoursite.com
Internet explorer was not able to open this internet site. The requested site
is either unavailable or cannot be found. Please try again later."
```

The only workaround for this issue is to allow the file to be cached locally. There are two ways to accomplish this. If you are using SSO, you can use the

```
OsoSendCacheHeaders off
```

directive. This directive will override the default SSO behavior and SSO will stop sending Pragma “no-cache” headers for SSO-protected content. For security reasons, be judicious in the use of this directive. It should be used in the narrowest context as possible by embedding it in other Apache directives.

For example, if the above file is the only file downloadable, use a FilesMatch directive as follows:

```
<FilesMatch "xxx\.pdf">
    OsoSendCacheHeaders off
</FilesMatch>
```

If you have multiple Word and Adobe files then do it by file type as in:

```
<FilesMatch "\.(doc|pdf)$">
    OsoSendCacheHeaders off
</FilesMatch>
```

Note this poor browser behavior is possible whenever you allow document downloads with “no-cache” headers which you may chose to do even if you are are not using SSO. The above solution is SSO specific. If you encounter this same issue and you are not using SSO, simply use the unset and append directives to fix the problem as in:

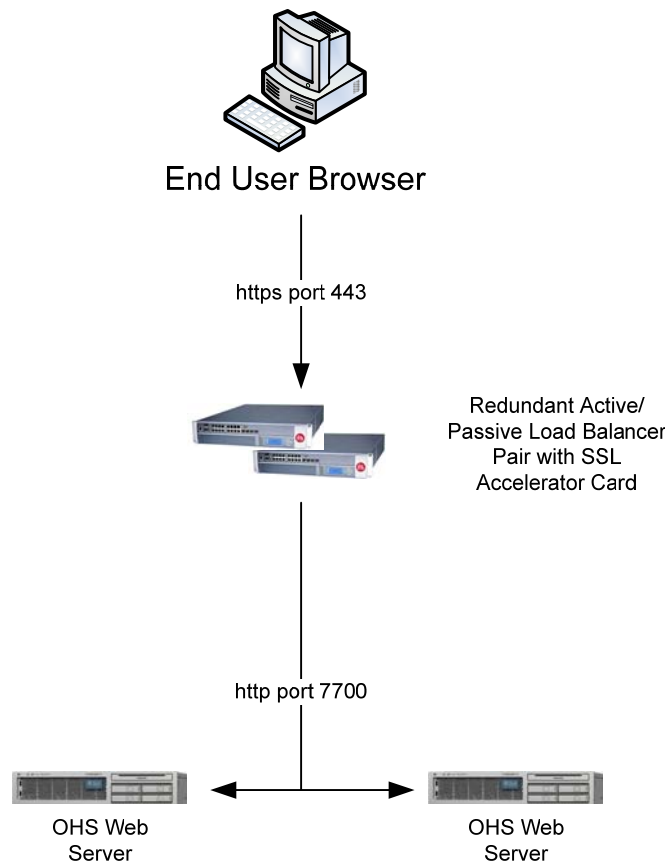
```
<FilesMatch "xxx\.pdf">
    Header unset Cache-Control
    Header unset Pragma
    Header append Cache-Control "public"
</FilesMatch>
```

or

```
<FilesMatch "\.(doc|pdf)$">
    Header unset Cache-Control
    Header unset Pragma
    Header append Cache-Control "public"
</FilesMatch>
```

TERMINATING SSL IN FRONT OF OHS

A typical production environment will have multiple servers running OHS to provide scalability and high availability. In such environments a hardware load balancer is often used to both monitor the health of and distribute traffic to the pool of web servers. Because encryption via hardware SSL accelerator cards is much faster than software-based SSL encryption, the load balancer is often used as the SSL termination point. In this case all traffic between the load balancer and the client browser is encrypted (https) but from the load balancer to OHS the traffic is in the clear (http) as in the diagram below:



In this configuration, OHS is not aware of the SSL terminating above it and will therefore send any self referencing URLs via http instead of https thus causing problems at the browser. To make OHS aware of the SSL in front of it, use the SimulateHttps directive as follows:

```
<VirtualHost MyVirtualHost:7700>
    SimulateHttps on
    .
    .
    .
</VirtualHost>
```

This directive can also be implemented at lower levels using Location or Directory directives and will require loading of mod_certheaders.

ORACLE METALINK REFERENCE NOTES

We recommend you read the following Metalink notes when planning your OHS deployment.

ORACLE HTTP SERVER (OHS)

- Subject: Everything You Wanted to Know About the Apache-Based OHS Version
- Doc ID: 260449.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=260449.1>)
- Subject: How Apache Works
- Doc ID: 334763.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=334763.1>)
- Subject: How To Change The Information That Apache Returns About Itself In The Header ?
- Doc ID: 258231.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=258231.1>)
- Subject: How to Setup Oracle HTTP Server as a Virtual Host Reverse Proxy
- Doc ID: 314381.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=314381.1>)
- Subject: How to Configure Virtual Hosts on HTTP_Server for LBR Access or SSL Termination
- Doc ID: 378003.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=378003.1>)
- Subject: Is it Possible To Configure Mod_plsql With Apache 2.0?
- Doc ID: 428391.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=428391.1>)
- Subject: Preparing and Configuring Virtual Hosts on Oracle Application Server 10g
- Doc ID: 293697.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=293697.1>)
- Subject: Steps to Maintain Oracle Database 10.2 Companion CD Home (for Oracle HTTP Server)
- Doc ID: 400010.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=400010.1>)

MOD_SECURITY

- Subject: How to enable and test mod_security in HTTP Server
- Doc ID: 456388.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=456388.1>)
- Subject: How To Enable Logging When mod_security is Enabled?
- Doc ID: 733329.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=733329.1>)
- Subject: How to Setup Discoverer Viewer 10.1.2 With E-Business Suite 11i Using Apache Server as Reverse Proxy
- Doc ID: 435184.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=435184.1>)
- Subject: Cannot Access Pls Pages: 'mod_security: Access denied with code 400'
- Doc ID: 389558.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=389558.1>)

MOD_REWRITE

- Subject: How to Redirect Based on REQUEST_METHOD: Disabling HTTP TRACE Requests as an Example
- Doc ID: 259404.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=259404.1>)
- Subject: Hints and Tips for Troubleshooting the URL Firewall
- Doc ID: 460564.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=460564.1>)
- Subject: How to Use HTTP Server MOD_REWRITE to Disable Certain Report Servlet Parameters?
- Doc ID: 360001.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=360001.1>)
- Subject: How To Configure & Run Apache in Restricted Mode During Maintenance
- Doc ID: 310969.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=310969.1>)

E-BUSINESS SUITE

Subject: Dual OHS Listener Configuration for Mid-Tier PL/SQL in E-Business Suite 11i

Doc ID: 239176.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=239176.1>)

Subject: Mod_plsql and Oracle E-Business Suite Release 12

Doc ID: 726711.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=726711.1>)

Subject: DMZ Configuration with Oracle E-Business Suite 11i

Doc ID: 287176.1 (<https://metalink2.oracle.com/metalink/plsql/showdoc?db=NOT&id=287176.1>)

ADDITIONAL REFERENCE LINKS

The following additional reference links were used in preparing this white paper:

Apache 2.0 Documentation - Multi-Processing Modules

<http://httpd.apache.org/docs/2.0/mod/worker.html>

A Complete Guide to the Common Vulnerability Scoring System Version 2.0

<http://www.first.org/cvss/cvss-guide.html>

CVSS Calculator

<http://nvd.nist.gov/cvss.cfm?calculator&version=2>

Recommended Security Controls for Federal Information Systems - NIST 800-53

<http://csrc.nist.gov/publications/nistpubs/800-53-Rev2/sp800-53-rev2-final.pdf>

Center for Internet Security

<http://www.cisecurity.org/>

Vendors using FIPS 140-2 Certified Cryptographic Modules

<http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/1401vend.htm>

Apache Module Mod_rewrite

http://httpd.apache.org/docs/2.0/mod/mod_rewrite.html

Mod_rewrite URL Rewriting Guide

http://httpd.apache.org/docs/2.0/rewrite/rewrite_guide.html

Mod_security

<http://www.modsecurity.org/projects/modsecurity/apache/index.html>

Got Root Site maintaining a set of Mod_security rules

http://www.gotroot.com/tiki-index.php?page=mod_security+rules

Mod_security Pro

<http://www.breach.com/products/modsecurity-pro-m1100.html>

Apache Security Book by Ivan Ristic

<http://www.amazon.com/Apache-Security-Ivan-Ristic/dp/0596007248>

ABOUT THE AUTHORS

KEVIN SHEEHAN

Kevin has over 28 years experience in Information Technology. That makes him either a grizzled veteran or an old coot depending on your state of mind. His 15 years at AT&T, seven at Oracle and six at Unisys, has given him experience on platforms ranging from IBM mainframes to Unix-based mid-range servers to Unix-based mainframes to Linux-powered grid technologies.

Kevin has worked with Oracle technology for the past 15 years and for the last six years has worked in the federal government space, particularly, Homeland Security environments. He is familiar with securing systems based on the Department of Homeland Security's Secure Baselines and takes a special interest in applications and service monitoring using tools like Quest Foglight and Oracle Grid Control.

Kevin blogs; however irregularly, at <http://securedba.com> and can be reached at kpsheehan@gmail.com.

BRIAN MULREANY

Mr. Mulreany started his career with AT&T Business Services Division. He held a number of positions with AT&T including Technical Director for Software Architecture.

He left AT&T and moved to Oracle Consulting. At Oracle, he worked directly with a variety of companies deploying Internet facing applications based on Oracle and Java.

Mr. Mulreany joined Unisys as a senior architect. He currently supports a large DHS contract which uses a wide range of Oracle technology. Mr. Mulreany can be contacted at bjm-uva@alumni.virginia.edu.