

Securing Oracle Application Server

By Caleb Sima

Securing Oracle Application Server

Table of Contents

<i>Introduction</i>	3
<i>Oracle Application Server Architecture</i>	5
<i>Oracle Application Server Security</i>	10
<i>Before Installation</i>	10
<i>Installation</i>	11
<i>Securing Web Cache</i>	33
<i>Advanced Hardening</i>	35
<i>After Installation</i>	38
<i>Maintenance</i>	39
<i>Operating System Security</i>	40
<i>Network Security</i>	42
<i>Conclusion</i>	45
<i>The Business Case for Application Security</i>	47
<i>About SPI Labs</i>	49
<i>About S.P.I. Dynamics Incorporated</i>	50
<i>About the Author</i>	50
<i>Contact Information</i>	51
<i>Appendix A: Oracle Application Server Installation Security Checklist</i>	52
<i>Appendix B: Bibliography</i>	55

Securing Oracle Application Server

Introduction

Recent software development and deployment trends have caused a shift from the classic client-server two-tier architecture to a three-tier model. The software that occupies this new middle tier is commonly referred to as middleware, precisely because that is what level it occupies in the hierarchy. This transition was predominantly caused by the explosion in popularity of the Web, and the increasing dominance of the Web browser in the client tier. The programming logic, previously located in the client tier in what is now known as a “fat client”, was instead moved onto the server. Initially, the functionality of the middle tier was restricted to that of the HTTP server and the scripting engine. However, over time the complexity of the middle tier has increased to encompass a wide variety of components commonly used for enterprise application deployment.

Oracle Application Server 10g is Oracle’s middleware offering. Although Oracle is best known for their enterprise database software, they introduced a complete protocol stack, called Oracle Fusion Middleware, with the release of Oracle Application Server. While there is a lot of information detailing the security of Oracle Database products, there is not enough that covers the security of Oracle Application Server. This is somewhat unfortunate because in a typical setup the web server and the application server are exposed to the public, and represent the point that is the easiest to attack. The documentation provided with Oracle Application Server is not lacking in detail. On the contrary, the sheer amount of information combined with the

Securing Oracle Application Server

large number of possible installation architectures make it very difficult to develop the proper steps to ensure maximum security. Thus the aims we established in this document were to address the most important issues of an Oracle Application Server installation and configuration, and to produce a reasonably-sized document that addresses the core functionality typically used in every installation.

The end result should be useful in securing a brand new installation of Oracle Application Server, or to assess the security of an existing installation. The focus of this document is a minimal installation of Oracle Application Server. As will be readily apparent, the minimal installation is a big enough challenge on its own. We will be focusing on the Oracle Application Server 10g R2 (10.1.2.0.2) release with the following major components installed:

- Oracle HTTP Server
- Oracle Container for J2EE (OC4J)
- Web Cache
- Oracle Enterprise Manager 10g

We used two installations of Oracle Application Server for our tests, one running on Windows (Windows Server 2003) and the other running on Unix (Red Hat Enterprise Linux AS 4). We wanted to make our guidelines equally useful to both major platforms. At the time of writing Oracle has made the preview of the next-generation release, R3 (10.1.3), available to the general

Securing Oracle Application Server

public. Although the new release brings many improvements over R2, the core functionality (which is our primary focus here) remains stable. Thus we expect the document to apply equally well to Oracle Application Server 10g R3 when it becomes officially available. Either way, our approach is to provide you with a step-by-step guide. You should be able to apply the same steps to any future version of Oracle Application Server and secure the installation.

Oracle Application Server Architecture

Before we go into the details of the core components, it might be beneficial to take a look at the big picture in order to understand what Oracle Application Server entails. Figure 1 illustrates the components of OAS. They include:

- J2EE & Internet Applications (Oracle HTTP Server)
- Caching (Web Cache)
- E-Business Integration
- Management and Security
- Portals
- Wireless

Securing Oracle Application Server

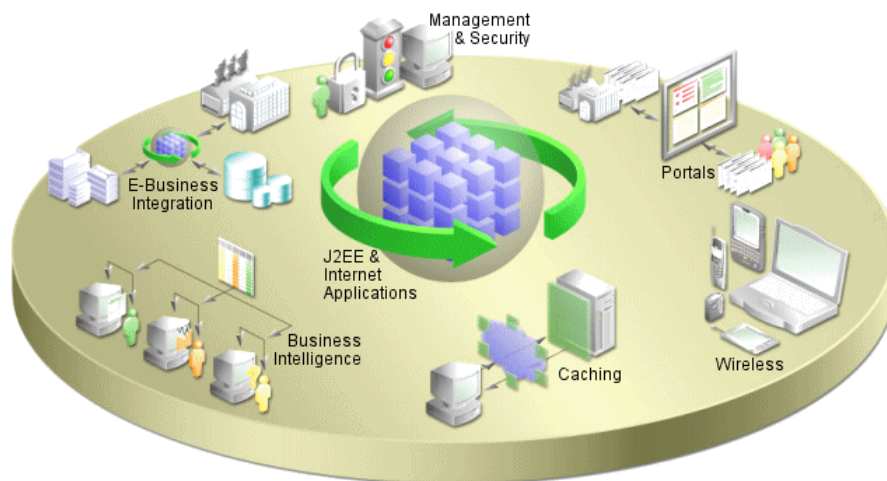


Figure 1: Oracle Application Server Components

The core component of Oracle Application Server is, of course, Oracle HTTP Server. Earlier versions of Oracle HTTP Server were entirely proprietary, but Oracle has decided to base the newer versions on the Apache web server. This has proven to be a good move. Not only is Oracle HTTP Server now based on a popular web server of proven quality, it is also compatible with a great number of third-party modules. It also benefits from widely available Apache expertise.

As installed by default, Oracle HTTP Server is based on the Apache 1.x branch (more specifically, on Apache 1.3.31 in 10g R2). Although Oracle provides a version based on Apache 2.x (available from the Companion Disks), there is very little reason to use this other version (especially since it is missing two modules you may want to use, `mod_plsql` and `mod_oradav`). There are some situations where an Apache 2.x solution is better, for

Securing Oracle Application Server

example if you need to deploy a third-party module that only supports Apache 2.x, or if you intend to develop your own modules - Apache 2.x offers a much better module-development environment. Also, starting a new module development based around Apache 1.x would be a waste of resources because Apache 2.2.x is expected to be released soon.

A deeper look into Oracle HTTP Server reveals support for most mainstream technologies:

- CGI scripts
- Persistent CGI scripts through FastCGI
- Compatibility with third-party Apache modules
- J2EE container (also known as OC4J)
- mod_perl
- mod_php (based on 4.3.9 in 10g R2)
- PL/SQL Server Pages
- SOAP/Web Services (through OC4J)
- XML (through OC4J)
- SSL
- WebDAV

Oracle HTTP Server utilizes two programming models, one on Windows, and the other on Unix platforms. Understanding the limitations of the programming model are important when implementing defenses against

Securing Oracle Application Server

Denial of Service attacks. Figure 2 represents Oracle Application Server operating using the Unix programming model, called prefork mode.

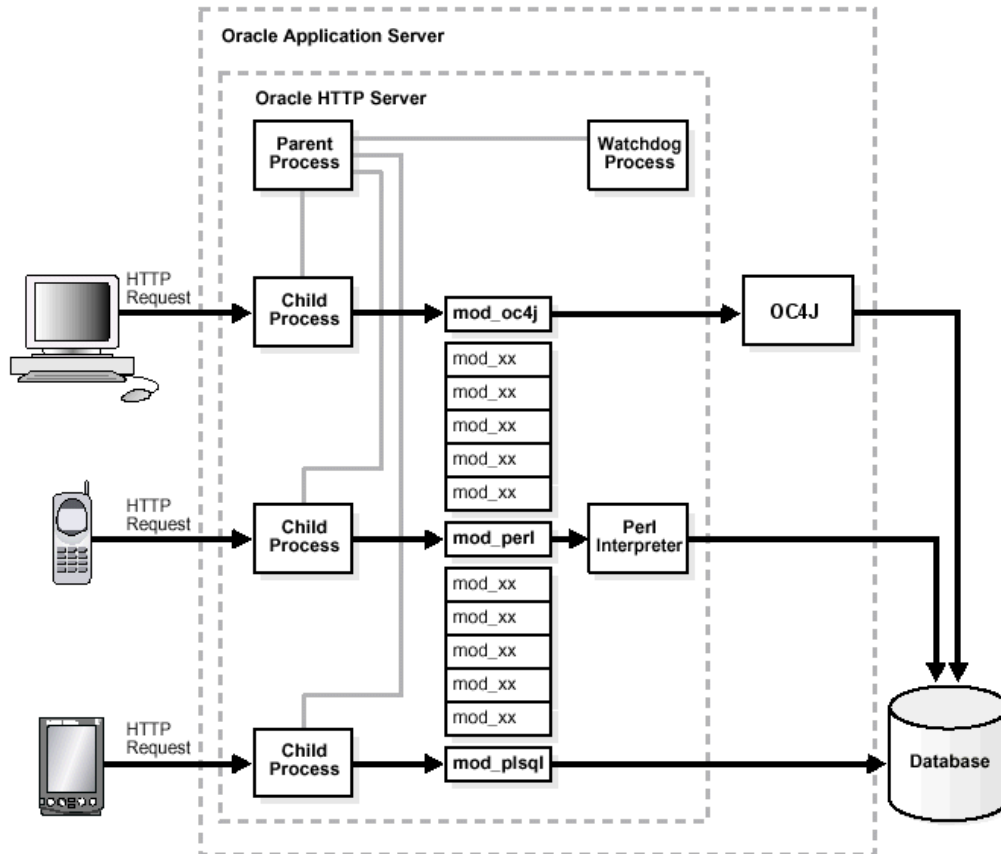


Figure 2: Oracle Application Server in Prefork Mode

Web Server Programming Model

There are two important things to note from the diagram. First, notice the number of processes used to serve client requests. In the prefork mode there is one main process that controls the web server (the "parent process"), and

Securing Oracle Application Server

then one “child process” for every request being processed. This is quite a robust programming model; if the web server crashes while it is processing a request the crash will affect one process only (the one that crashed). The parent process will simply create a new child process to replace it and carry on as usual. The requests processed at that time will remain unaffected. Consequently, the impact of certain types of vulnerabilities, typically the ones that can be used to cause Denial of Service attacks by crashing the web server processes, is lessened. On Windows, a threaded programming model is used. The one parent process is still there, but in this case there is only one child process which runs with multiple threads. This programming model is better suited to Windows, but a crash in the child process will also terminate all requests that are being processed at that time.

The other important detail to note from the diagram is the relationship between the web server and the components that provide the additional services. The core of the web server, on its own, only knows how to serve static files. A separate web server module needs to be used for all other required functionality. Modules are thus an essential and integral part of the web server. It is entirely up to the module to decide how the request will be processed. Some modules, such as `mod_perl`, do all of the work from within the web server. Other modules, such as `mod_oc4j`, will relay the request to some other process for processing. Having requests processed outside the web server is much better for security because if the external process is compromised it will not affect the work of the web server.

Securing Oracle Application Server

Oracle Application Server Security

The process of securing Oracle Application Server takes at least five steps, divided into the following areas:

- Before Installation
- Installation
- Configuration
- After Installation
- Maintenance

Before Installation

Time invested reading the Oracle Application Server documentation before installation is time well spent. The more you know about the software the easier it will be to make an installation plan and configure the necessary components. If you haven't already secured the operating system, go ahead and do that first. There is no point in building on top of something you aren't certain is secure. Make sure the computer you are using for the installation cannot be accessed from the outside. If you need to download any patches, do it before the installation, and then simply unplug the network connection until you finish.

At this point it may be a good idea to record the state the system is in. After the installation, you will be able to use this information as a baseline for comparison to determine what changes were introduced by the installation process. If you are using Unix more than likely you already have the tools you need for this task. On Windows, I recommend the free PsTools

Securing Oracle Application Server

(<http://www.sysinternals.com/Utilities/PsTools.html>) system utilities. You can start by recording the open ports on the server (`netstat -lnt` on Unix or `netstat -aon` on Windows):

```
# netstat -lnt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 0.0.0.0:6000            0.0.0.0:*               LISTEN
tcp      0      0 127.0.0.1:25           0.0.0.0:*               LISTEN
tcp      0      0 :::6000                :::*                    LISTEN
tcp      0      0 :::22                  :::*                    LISTEN
```

You should also record the running processes. On Unix, record the output of `ps auwxmf`. On Windows, execute `pslist -t`. The output is typically very long and for that reason we will not reproduce it here. You are now ready to proceed with the installation.

Installation

On Unix, first create the main Oracle Application Server account. Most people choose to go with `oracle`. This is what we will assume here for sake of simplicity. However, you should use another non-obvious name on a production system. Well-known account names are sometimes targeted by the attackers in brute force attacks against passwords. Choosing a different name could preempt such efforts.

```
# groupadd oinstall
# useradd oracle -g oinstall
```

Securing Oracle Application Server

It is usually a good idea to lock this account, especially if there will be more than one person performing the maintenance. If they all connect to the server as "oracle" it will be impossible to track who has done what, and when. You then need to start the installation process as the new user. The process is quite straightforward. You will need to execute certain scripts as root to allow OAS to configure privileges correctly.

On Windows, create a new local user account specifically for this purpose, add it to the Administrator group, and install OAS while running as that user. After the installation, download the latest patches for the installed components from Metalink (if you haven't already done so), and apply them.

Getting to know Oracle Application Server

Now you can proceed to examine the new installation. Start with the permissions of the installation directory. At this point there is no good reason for any other user on the same computer to have access to this directory. Next, list the open ports on the server and all the processes that are running, as you did before the installation, and examine the differences to determine the changes. Oracle Enterprise Manager (the web-based management interface of Oracle Application Server) comes with a very nice set of tools that will allow you to look at the infrastructure. However, these tools will not show all the related processes. The safest approach is to look at the processes directly from the operating system.

Securing Oracle Application Server

Below is a simplified process list after the pre-installation processes were removed. (We have added a brief description on each line. Those are not included as part of a normal `netstat` output.)

Proto	Local Address	PID/Program name	Description
tcp	127.0.0.1:7200	6180/httpd	HTTP Server diagnostic port
tcp	0.0.0.0:7778	6180/httpd	HTTP Server main port
tcp	0.0.0.0:1156	27122/java	Enterprise Manager
tcp	0.0.0.0:1157	27037/emagent	Enterprise Manager Management Agent
tcp	0.0.0.0:46927	27164/java	DCM
tcp	0.0.0.0:7777	6208/webcached	Main Web Cache process
tcp	127.0.0.1:12401	6176/java	OC4J RMI port
tcp	0.0.0.0:6003	6157/opmn	OPMN ONS Request
tcp	0.0.0.0:12501	6176/java	OC4J AJP port
tcp	127.0.0.1:6101	6157/opmn	OPMN ONS Local
tcp	0.0.0.0:9400	6179/webcached	Web Cache Admin Console
tcp	0.0.0.0:6200	6157/opmn	OPMN ONS Remote
tcp	0.0.0.0:12601	6176/java	OC4J JMS port
tcp	0.0.0.0:9401	6208/webcached	Web Cache invalidation port
tcp	0.0.0.0:1850	27122/java	Enterprise Manager RMI
tcp	0.0.0.0:9402	6208/webcached	Web Cache statistics port

As you can see, the installation has created a large number of new processes. We can make two conclusions from looking at the list of open ports. First, some of the processes are only listening on the loopback address (127.0.0.1), meaning they are only accessible from the server itself. Thus it is reasonable to assume these ports are used for the internal components of the installation. Of all the listening ports only one needs to be accessible publicly, which by default is Web Cache port 7777 (although you are likely to change it to port 80 later on). This means you need to protect all other ports

Securing Oracle Application Server

from the potential attackers with a firewall. Keep this thought in mind for later.

Now you can generate a list of newly created processes. You can do this either by comparing the process lists before and after the installation. But even if you do not have a process list from before you can take advantage of the fact that the new processes are all still running as the main installation user. You can list all processes belonging to that user with `ps -o pid,command -u oracle`.

The first two processes belong to the Oracle Process Manager and Notifier (OPMN), which is the main controlling component of Oracle Application Server. In the case of Oracle Application Server, OPMN controls all other processes except those belonging to the Enterprise Manager.

```
6155 /opt/oas/opmn/bin/opmn -d
6157 /opt/oas/opmn/bin/opmn -d
```

There are two Web Cache processes. If you look at the open port list and compare the process IDs you will find that one process is used to process HTTP requests (including invalidation and statistics), while the other is used for the control operations and to run the Web Cache Manager.

```
6179 /opt/oas/webcache/bin/webcached -A -OPMN -U 1386873166
6208 /opt/oas/webcache/bin/webcached -OPMN -U 1386873165
```

Securing Oracle Application Server

There is one Java process in which OC4J runs. (It is very unfortunate that Java processes do not have meaningful names; the process list would be much nicer to look at.) In an effort to make the process list useful we have removed most of the content from the command line, leaving just the important part - the value of the `-jar` parameter. This parameter contains the equivalent of process name in Java.

```
6176 /opt/oas/jdk/bin/java -jar oc4j.jar \  
-config /opt/oas/j2ee/home/config/server.xml
```

Enterprise Manager consumes four processes. If you compare the command lines of the Java processes you will find that the Enterprise Manager actually uses its own copy of the OC4J container to run.

```
27036 /opt/oas/perl/bin/perl /home/oracle/OraHome 3/bin/emwd.pl iasconsole \  
/home/oracle/OraHome 3/sysman/log/emiasconsole.nohup  
27037 /opt/oas/bin/emagent  
27122 /opt/oas/jdk/bin/java -jar /opt/oas/j2ee/home/oc4j.jar \  
-config /opt/oas/sysman/j2ee/config/server.xml  
27164 /opt/oas/jdk/bin/java -jar /opt/oas/dcm/lib/dcm.jar
```

A number of processes are used by the web server. The first process is the main web server process (the "Parent Process" in Figure 2). It is followed by two processes that are in charge of log rotation, one for the access log and the error log, respectively. The fourth process is the FastCGI process manager, which, if FastCGI is used, dynamically creates and destroys

Securing Oracle Application Server

FastCGI processes. What remains is a list of child web server processes. These are in charge of request processing. At this point they are just waiting for the next request to come in.

```
6180 /opt/oas/Apache/Apache/bin/httpd -d /opt/oas/Apache/Apache -U 1386873167
6222 /opt/oas/Apache/Apache/bin/rotatelog /opt/oas/Apache/Apache/logs/error_log
43200
6223 /opt/oas/Apache/Apache/bin/rotatelog /opt/oas/Apache/Apache/logs/access_log
43200
6224 /opt/oas/Apache/Apache/bin/fcgi- -d /opt/oas/Apache/Apache -U 1386873167
6225 /opt/oas/Apache/Apache/bin/httpd -d /opt/oas/Apache/Apache -U 1386873167
6227 /opt/oas/Apache/Apache/bin/httpd -d /opt/oas/Apache/Apache -U 1386873167
6230 /opt/oas/Apache/Apache/bin/httpd -d /opt/oas/Apache/Apache -U 1386873167
6233 /opt/oas/Apache/Apache/bin/httpd -d /opt/oas/Apache/Apache -U 1386873167
6235 /opt/oas/Apache/Apache/bin/httpd -d /opt/oas/Apache/Apache -U 1386873167
6236 /opt/oas/Apache/Apache/bin/httpd -d /opt/oas/Apache/Apache -U 1386873167
6270 /opt/oas/Apache/Apache/bin/httpd -d /opt/oas/Apache/Apache -U 1386873167
6273 /opt/oas/Apache/Apache/bin/httpd -d /opt/oas/Apache/Apache -U 1386873167
```

If you need to use the privileged ports

Immediately after the installation you will have the entire Oracle Application Server software stack running as the non-privileged user you installed it with. Chances are that this is not what you need - you will probably need a process to listen on the privileged ports 80 and 443. To do this you would need to run at least one component as `root`. Normally, that component is Web Cache. Only in situations where Web Cache is not used will you need to configure Oracle HTTP Server to run as `root`.

Securing Oracle Application Server

To configure Web Cache to run as `root` you first need to stop it, and then execute the following commands:

```
# cd $ORACLE_HOME/webcache/bin
# ./webcache_setuser.sh setroot oracle
```

To configure Oracle HTTP Server to run as `root` do the following:

```
# cd $ORACLE_HOME/Apache/Apache/bin
# chown root .apachectl
# chmod 6750 .apachectl
```

Because the `apachectl` script simply calls the `apachectl` script it is a good idea to change the ownership of the `apachectl` script to `root` to prevent anyone from modifying it. Otherwise you would be allowing anyone with write privileges over `apachectl` to execute any command as `root`!

```
# chown root.root apachectl
# chmod 755 apachectl
```

Securing HTTP Server

In order to properly secure an Oracle Application Server installation you first need to determine what you actually have. A good way to do this is to examine the configuration files. There are many configuration files which you can examine by starting with

`$ORACLE_HOME/Apache/Apache/conf/httpd.conf` and following the

Securing Oracle Application Server

references to other files. You will need a good knowledge of the Apache web server together with access to Apache documentation in one browser window and Oracle HTTP Server documentation in another. If you encounter something you don't need simply comment it out. There is a slight chance you will remove something useful, but if you perform this step right after the installation (as opposed to right before production) any problems you introduce now should be discovered in the development and testing phase.

Remove default content

Most applications come with default content and pre-installed examples. The default content is not very dangerous but in some circumstances it may provide the attacker with one too many details about your infrastructure. You need to remember that the more they know about you the easier it is for them to find and exploit a vulnerable spot. It is often several seemingly innocuous pieces of information added together by an attacker that leads to a successful attack.

Pre-installed examples have the potential to be more dangerous than default content. After all, unlike default content, examples contain programming code. This code is usually not given the attention it deserves because everyone assumes it will be removed from a production server. This often does not happen, which can easily lead to the example code getting exploited.

Securing Oracle Application Server

Oracle Application Server comes with many examples that **must** be removed because they do contain vulnerable code. The following is a complete list that applies to Oracle Application Server 10g R2 (10.1.2.0.2):

1. FastCGI examples: `echo` and `echo2` on Unix, `echo.exe` and `echo2.exe` on Windows (located at `$ORACLE_HOME/Apache/Apache/fcgi-bin/`).
2. Additional FastCGI content from `$ORACLE_HOME/Apache/Apache/fastcgi/`.
3. J2EE application `IsWebCacheWorking`.
4. J2EE application `ADFBCManager`.
5. J2EE application `BC4J`.
6. Default JSP examples, which are part of the default OC4J context (`$ORACLE_HOME/j2ee/home/default-web-app/examples/`).
7. Default Servlets, which are also part of the default OC4J context (`$ORACLE_HOME/j2ee/home/default-web-app/WEB-INF/classes/*`). If you delete everything in the specified folder you will also take care of the "hidden" servlets that are not linked from the example pages but can, nevertheless, be executed by someone that knows they are there.

After removing the files you may also want to remove the following aliases (if you won't be using them, that is):

- `/perl/`
- `/fcgi-bin/`
- `/cgi-bin/`

Securing Oracle Application Server

- `/j2ee/` (introduced via `Oc4jMount` in `mod_oc4j.conf`)

You will need to change the aliases in the configuration files (directives `Alias`, `ScriptAlias`, `Oc4jMount`).

Replace the default home page

Change the name of the folder `$ORACLE_HOME/Apache/Apache/htdocs` to something else and create an empty folder with a single `index.html` file. I usually leave the file empty.

It is a good idea to avoid adding content to the default location, and instead creating a new virtual host. You may think if you create a new virtual host for your application you won't need to go through the boring process of removing all those examples. You may be surprised to learn that even in that case the examples would follow you and appear on the new virtual host, too! This is because the examples are defined in the body of the main server. It is the configuration of the main server that is used as a template for virtual hosts; virtual hosts will simply inherit the configuration. In practice this means that unless you remove them, all the aliases from the main server will be present in the virtual hosts. This will be true even for the OC4J aliases because the value of the `Oc4jMountCopy` directive is `On` by default.

Prevent information leaks

Giving away as little information about yourself is always the preferred option. Oracle HTTP Server already comes pre-configured not to show

Securing Oracle Application Server

directory contents, but there is room for two more improvements. First, you want to remove the web server information from the error pages by changing the value of the `ServerSignature` directive (in `httpd.conf`):

```
ServerSignature Off
```

There is another place where web servers usually put their make and model, in the `server` header of every response. To minimize the amount of information disclosed there, add the following line to `httpd.conf`:

```
ServerTokens ProductOnly
```

The name of the web server will remain visible but the version number will be protected.

If you intend to use PHP, make a change in the `php.ini` file:

```
expose_php = Off
```

Deactivate unused modules

Oracle HTTP Server is very modular. A separate module is usually used to implement every major piece of functionality. By default many of these modules are included in the configuration, in spite of the fact that you may never use them. Still, being active, there is a chance that they can be

Securing Oracle Application Server

exploited by an attacker. As part of the hardening process you should go through the list of modules and remove the ones you won't use.

The main candidates for removal are, of course, the ones that contain large chunks of functionality. Remember, the more complex the module the more likely is to contain an undiscovered problem of some kind. Suggested modules to remove include:

- Proxy (`mod_proxy`)
- Perl (`mod_perl`)
- CGI (`mod_cgi`)
- FastCGI (`mod_fastcgi`)
- PHP (`libphp4`)
- PL/SQL (`mod_plsql`)
- WebDAV (`mod_oradav`)
- SSL (`mod_oss1`)
- Server-Side Includes (`mod_include`)

To remove a module you need to find and comment out the corresponding `LoadModule` directive that loads it from a shared library, and then look for a similar `AddModule` directive (which is sometimes not used). If the

Securing Oracle Application Server

configuration file contains a directive that is implemented by that module, you should be aware that module removal may lead to the web server not being able to start. In most cases this won't happen because Oracle HTTP Server uses the `<IfModule MODULE_NAME>` and `</IfModule>` tags to surround the directives. With these tags in place the web server simply skips over the module-specific directives. But if it does happen in your case all you need to do is comment out the offending directives (or put the `<IfModule>` tags around them, which is probably better).

There is little to gain from removing some of the other default modules because they don't contain much code. Some of them are also quite useful so you may happen to use them in the future after all. Still, if you are very paranoid you may want to consider also removing the following modules:

- `mod_log_agent`
- `mod_cern_meta`
- `mod_log_referer`
- `mod_usertrack`
- `mod_speling` (that is the correct name)
- `mod_imap`
- `mod_info`
- `mod_vhost_alias`

Configure limits

Properly configured web server limits are essential for the stability of the system. You do not have to be attacked to experience problems. Problems

Securing Oracle Application Server

with limits can happen for a variety of reasons: too many users (or a few very demanding users), badly written applications (use too much of your resources to process one request), traffic spikes, and so on. Ultimately, if you allow too many requests to be processed at the same time the web server may attempt to allocate too much of the operating memory, which will cause the machine to start swapping, and generally make the machine unresponsive. It may even crash completely. When you *do* get attacked, properly configured limits will make the server more resistant. In the worst case the web server may become unusable, but the machine will survive, allowing you to access it and implement manual defense systems.

The first limit to think about is the communication timeout:

```
Timeout 60
```

This setting is used in various places in the Apache code, and even in the third-party modules. The default value (300 seconds) is far more than necessary in most situations. You are advised to reduce the value to at least 60 seconds. If you experience problems (unlikely, but that depends on your circumstances), you can always increase it to a larger value.

The fundamental problem with Apache is its request processing model. As soon as a connection from the client is established Apache will allocate one instance to deal with it (a process in prefork mode, and a thread in

Securing Oracle Application Server

multithreaded mode). This instance will remain exclusively attached to the connection until the connection is terminated. This processing model, while easy to implement from a programming model (or at least, easier than the alternatives), has two major disadvantages.

There must be a upper limit on the number of Apache instances active at any one time. This is because each instance consumes some amount of resources on the server. The upper limit must be put in place to protect the server from running out of the resources. This means that if there is a maximum limit of 150 instances at any one time, and all of them are engaged, the next connection request that arrives will simply not be processed until one of the instances closes a connection. In most cases the problem is manifested by a long wait experienced by the users. In extreme cases the connection requests will simply be lost.

The other problem is related to the Keep-Alive feature of the HTTP protocol. In the early days of HTTP each request was processed on its own TCP/IP connection. This wasteful approach placed increased performance requirements on the servers, and increased latency. The Keep-Alive feature, introduced in HTTP/1.1, allows a client to maintain a connection with the server across many requests. It sounds like a good idea, but it does not work well in combination with the Apache processing model which was described earlier. Remember how there is one instance of Apache dedicated to each connection? A typical client will not simply send its request one after another

Securing Oracle Application Server

and terminate the connection. Instead, it will send one request, do something with the data it receives, then submit another request, and so on. While this is fine for the client, the poor Apache instance can do nothing but sit idly in between requests. In effect, with the Keep-Alive features activated you can no longer look at the rate of requests per second but instead must look at the number of busy Apache instances in a second. The numbers are not the same. The differences will depend on whether the clients want to use persistent connections or not, and the way you have your web server configured.

The problem can be seen in Figure 3, which contains a snapshot of activity of the web server running Apache. They make their Apache status public on the address <http://www.apache.org/server-status/>. I recorded the activity using the `apache-monitor` script from the Apache `httpd` tools project (<http://www.apachesecurity.net/tools/>). You can see the number of servers sitting idle is mostly greater than the number of servers actually working processing requests.

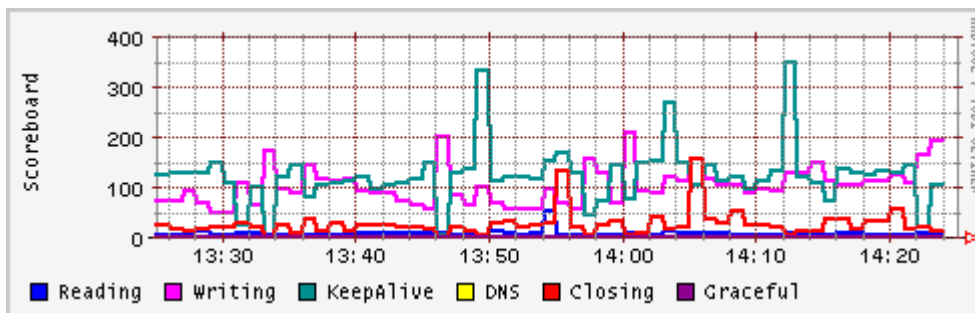


Figure 3: Apache Server Activity

Securing Oracle Application Server

There are two directives relevant to the problem:

```
KeepAlive On  
KeepAliveTimeout 5
```

By default the Keep-Alive feature is enabled and the timeout is set to 15 seconds. It is recommended to leave the Keep-Alive feature enabled, but to reduce the timeout to a lower value, such as 5 seconds. This will allow the server to take advantage of the persistent connections to increase the performance, but reduce the side-effects coming from clients that sit on the connection idle for too long. If you start to experience problems, reduce the Keep-Alive timeout further, or turn off the Keep-Alive feature altogether.

The exact method you will use to control the maximum number of requests processed at any one time depends on the processing model. The `MaxClients` directive is used with the `prefork` model:

```
MaxClients 150
```

The directive to use with the threaded process model is `ThreadsPerChild`:

```
ThreadsPerChild 50
```

Securing Oracle Application Server

There is no easy way to determine the correct value for the maximum number of instances. You must *choose* one using your best estimation and basing that on the information you have from testing and production. Ideally, you would minimize guessing by performing a proper testing phase and simulating a varying number of clients accessing your system. You will arrive at the correct value by monitoring the system parameters (memory consumption, CPU utilization) and choosing a value that yields maximum performance but which also keeps the server stable. As a rule of thumb, if you don't have time to do proper testing, choose a lower value and increase it if you encounter the limit.

To lower limits you will use the four Limit directives. Again, it is difficult to choose a one-size-fits-all value here, although we have tried to provide decent recommended values below:

```
# Maximum size of the request body
LimitRequestBody 65536

# Maximum number of request headers
LimitRequestFields 32

# Maximum size of each request header
LimitRequestFieldSize 4096

# Maximum size of the request line (first line in every request)
LimitRequestLine 4096
```

Securing Oracle Application Server

Your request body sizes are likely to be larger than 65536 bytes if you plan to support file uploads. In that case, increase the value accordingly. (But whichever value you choose it is likely to be better than the default value of unlimited.)

Enhance logging

By default Oracle HTTP Server will produce the access logs in the so-called `common` format. At the very least you should change the logging format to `combined`, which starts as `common` but adds the referrer and the user agent information at the end. In the interest of security, also consider talking to your developers and working out a way for the web server to log the session ID to the logs somehow. This will help you immensely if you ever find yourself investigating a potential breach of security through the web application.

As for the error log, by default levels `warn` and `higher` are logged. You may want to change this to `info`, as that level provides much more information.

```
LogLevel info
```

Note:

Oracle Application Server comes with Log Loader, an optional component used for log storage. Log Loader reads the error messages produced by other components and stores them into its own repository. However, Log Loader

Securing Oracle Application Server

will not take care of the access logs. This means you need to establish your access logs retention policy as part of the installation process.

SSL

If you are deploying SSL on the web server level (as opposed to terminating SSL at the Web Cache) you may want to disable certain protocols and ciphers. When we speak of SSL we usually refer to versions 2 and 3, and to the TLS 1.0, which is commonly accepted to be SSL v3.1. However, we now know that version 2 is not very secure. Unless you have really old clients that you must support (and they don't understand SSLv3 or better) you can safely disable SSL v2.

```
SSLProtocol All -SSLv2
```

You may also consider looking at disabling certain low-strength ciphers too, as illustrated in the following example:

```
SSLCipherSuite All:!EXP:!NULL:!ADH:!LOW
```

Interaction between Web Cache and HTTP Server

When you deploy Web Cache in front of your web servers they are no longer in touch with the actual clients. All they see is Web Cache itself. While this can increase security to some extent, the drawback is that they are no longer able to see the real IP addresses of the clients. Oracle HTTP Server uses the `UseWebCacheIP` directive to work around this problem:

Securing Oracle Application Server

```
UseWebCacheIP On
```

Since Web Cache "understands" this problem it always sends the real client IP address in the headers of the requests sent to the web servers. All `UseWebCacheIP` directive does is makes use of this information and "tricks" the web server into thinking the client is behind the IP address provided in the headers.

You need to choose the correct setting for `UseWebCacheIP` after the installation. It is set to `OFF` by default, which is correct (assuming the web server receives requests directly). However, if you install the web server together with Web Cache and leave the value at `OFF` the access controls that are configured on the web server will no longer work correctly.

As an example, if you have installed both the web server and Web Cache on the same machine, try accessing the server status page `http://servername.example.com/server-status/` (replace "servername.example.com" with your own URI or IP address) from any computer other than the server itself. You should be allowed access to the inner workings of the web server, but this is not what you would expect. The server status page is configured with:

```
<Location /server-status>  
    SetHandler server-status
```

Securing Oracle Application Server

```
Order deny,allow
Deny from all
Allow from localhost SERVERNAME SERVERNAME.EXAMPLE.COM
</Location>
```

Do you see the problem? Because of Web Cache being on the same machine, the web server will always use the local address for its access control! The server status is not the only vulnerable example. There are other OAS components that expect some requests to come only from the `localhost`. Changing `UseWebCacheIP` to `On` solves this problem.

There is another, more subtle problem involved with operating a cache in front a web server - that of caching protected content. Web Cache will cache the content that it considers *cacheable*. But, as in the case we discussed earlier, if it does not know the rules the web server is using to accept or reject requests, it may end up turning private content into public content. Consider the following configuration (which uses `mod_access`):

```
<Location /internalStuff/>
Order deny,allow
Deny from all
Allow from LOCALNETWORK
</Location>
```

If it happens that the first request for the resource comes from the internal network (and assuming `UseWebCacheIP` is set to `On`), the web server will look at the IP address and allow the request to go through. Since the response

Securing Oracle Application Server

travels through Web Cache it will observe and cache it, seeing no reason not to. The resource in question will be delivered straight from the cache for all subsequent requests until it expires from cache.

You can resolve this problem simply by not relying on network access controls on the web server level. Use passwords instead. Or, if you really must use network access controls, use `mod_headers` to tell Web Cache the content is not cacheable:

```
<Location /internalStuff/>
  Order deny,allow
  Deny from all
  Allow from LOCALNETWORK
  Header set Cache-Control private
</Location>
```

Securing Web Cache

There are fewer opportunities for Web Cache hardening, compared to the ground we covered in the web server hardening section. This is not only because Web Cache is a simpler more focused tool, but also because there is room to increase its configurability. This section applies only if you decide to use Web Cache as the first point of contact for the public.

Limits

The following aspects of Web Cache functionality can be limited:

Securing Oracle Application Server

- *Maximum individual header size* (8192 bytes by default). Try reducing it to 4096.
- *Maximum combined header size* (819000 bytes by default). This appears to be quite high and amounts to 100 headers at full size. You should reduce it to a sensible value, for example 16384.
- *Keep-Alive timeout* (5 seconds by default). Leave this at the default value.
- *Maximum cache size* (500 MB by default). What you choose here depends on the amount of RAM you have. Web Cache uses memory for its operation so you need to make sure you have this amount available for it to use.
- *Maximum incoming connections per web server* (100 by default). The value you configure here should be the same or slightly lower than the value you configured in the web server.
- *Maximum incoming connections* (500 or 700 by default). You should probably reduce this to a lower value but increase accordingly. For example, if you have 10 web servers behind a Web Cache and each server is capable of serving 150 requests at one time, configuring anything less than 1500 here would mean wasting capacity. However, you also need to remember it is impossible to scale this value indefinitely. Sooner or later you will reach the maximum of the ability of the server and probably need to migrate either to a more powerful server, or to a cluster of machines running Web Cache.

Securing Oracle Application Server

SSL

Disabling SSLv2 in Web Cache is not as straightforward as it is for the web server. Although it is supported in the configuration, the web-based GUIs (neither the Enterprise Manager nor the Web Cache Manager) do not allow it. You could edit the file \$ORACLE_HOME/webcache/webcache.xml directly, and find all lines like the following (there will be one for each SSL-enabled port):

```
<LISTEN IPADDR="ANY" PORT="7776" SLENABLED="SSL" PORTTYPE="NORM" >
```

Then change SSL to SSLV3_V2H (possible values are NONE, SSL, SSLV2, SSLV3, and SSLV3_V2H):

```
<LISTEN IPADDR="ANY" PORT="7776" SLENABLED="SSLV3_V2H" PORTTYPE="NORM" >
```

Unfortunately, any changes you make this way will be overwritten the next time you choose to configure the ports through a GUI. Consequently, you should only spend time doing this for cases where very high security levels are required.

Advanced Hardening

Before you complete the installation and the configuration process, you have one final opportunity to decide whether you want various Oracle Application Server components to run under different user accounts. This discussion only applies when you have all the components on the same machine. If you have multiple machines to play with, skip this section and read the "Network

Securing Oracle Application Server

Security" section instead. We will also warn you in advance that running as separate user accounts, although possible, is not at all straightforward. Things you will need to fix will break in the installation. Ultimately you will need to make up your own mind whether the end result justifies the effort. But at the very least the discussion below may save you a lot of time trying to get it right. Therefore consider it as a set of guidelines and be prepared to improvise.

There is a conflict between the fact that that you want components to run under different user accounts, but the main account (the one used to install Oracle Application Server) still needs to maintain control over the files. This creates all sorts of problems. For example, if you change the default permissions, when the time for an upgrade comes you may need to (temporarily) change the permissions back to the original values.

If you really want to benefit from the separation of components and you want them to run on the same machine, it may be a better (and cleaner) solution to simply install standalone versions of each component and maintain completely separate installations. This approach will produce the least number of problems. On the other hand, it does make it more difficult to connect the components together and to control them from only one GUI.

Securing Oracle Application Server

Note:

Create a separate account for every component. Do not use the `nobody` account. If you use the `nobody` account for two or more services and one of them gets compromised, the attacker will be able to take control of the remaining services running using the same account. Keep in mind the new accounts must at least have read access to the installation directory.

Running HTTP Server under own account

Getting HTTP Server to run under a separate account is not very difficult. It consists of four easy steps:

1. Create a new user account and a group (e.g. `oas_httpd`).
2. Change `httpd.conf` to make use of them (directives `User` and `Group`).
3. Configure HTTP Server to start as `root` as explained earlier (this is necessary for the process to change its identity into `oas_httpd`).
4. Make sure only the `root` user can write to `$ORACLE_HOME/Apache/Apache/logs`. However, allow `oas_httpd` to write to `$ORACLE_HOME/Apache/Apache/logs/fastcgi` (this is where the FastCGI manager needs to write at run time).

Running Web Cache under own account

There is a handy script that changes the user account Web Cache runs under. Assuming you have created the user account (e.g. `oas_cache`) and stopped Web Cache, execute the following as `root`:

Securing Oracle Application Server

```
# $ORACLE_HOME/webcache/bin/webcache_setuser.sh setidentity oas_cache
```

Then edit `$ORACLE_HOME/webcache/webcache.xml` and change:

```
<IDENTITY USERID="oracle" GROUPID="oracle" Key="USERID GROUPID"/>
```

to

```
<IDENTITY USERID="oas_cache" GROUPID="oas_cache" Key="USERID GROUPID"/>
```

Finally, change the `webcache.xml` file permissions to not allow `oas_cache` to write to it, while making writing for the main account possible (because this is the account Enterprise Manager uses to run).

Running OC4J under own account

To run OC4J under a separate account is officially not possible, but Enterprise Manager does allow you to specify a path to the Java executable on disk. If, instead of the Java executable, you decide to call a `setuid` root binary that changes its identity into some other user account and then executes the real Java executable, you should be able to make it work.

After Installation

If you have made changes to the process list in the hardening phase you now have a good opportunity to record the running processes and the listening ports. This information should become part of the installation report. Other things you may consider doing:

Securing Oracle Application Server

- Start the installation diary.
- Implement and initialize file integrity monitoring.
- Make a full backup and store it somewhere safe. This step is strongly recommended. But if you decide against it, at least use the built-in backup tool to store the active configuration (Backup/Recovery option in Enterprise Manager).

Maintenance

Maintenance is often a neglected part of the deployment process, probably because it is a very unattractive activity. Still, it is a vital activity to maintain the required security levels. There are three general rules of thumb to follow in this section:

1. Know your environment and keep track of the changes.
2. Be aware of the security of the software you have deployed and patch it regularly.
3. In today's world it is no longer enough just to look at the advisories published by the vendor. The software security scene is very lively and you need to take the time to stay in touch. In the case of Oracle Application Server, which includes many other open source components, you may want to keep an eye on the security issues related to them, too.

Securing Oracle Application Server

The following is a list of resources you may find useful:

Oracle security alerts web site:

<http://www.oracle.com/technology/deploy/security/alerts.html>

Oracle security alerts RSS feed:

http://www.oracle.com/technology/syndication/rss_otn_sec.xml.

The Oracle support site, Metalink:

<http://metalink.oracle.com>.

Security mailing lists:

bugtraq@securityfocus.com

Blogs of people involved with Oracle security:

Pete Finnigan (<http://www.petefinnigan.com/weblog/entries>).

Operating System Security

If you take one step back from Oracle Application Server security, then host (or operating system) security is the next layer you encounter. This section includes a quick overview of the steps needed to secure the operating system before Oracle Application Server is installed.

Securing Oracle Application Server

Limit user access. Since OAS is a powerful beast you are not likely to install it on a multi-purpose machine but instead only allow access to those who need it for administrative work. Lock down all other accounts. You should not allow many users to use the same account. Configure the machine to force each administrator to log in with a separate account. Once on the machine they can change to the account they need to perform the work.

Deploy minimal services. Each service you leave running is a potential entry point for the attackers. Shut down the services that are not needed.

Patch regularly. Buffer overflows and other vulnerabilities are still a reality. If you patch regularly you keep the window of opportunity for an attacker down to a minimum.

Configure user limits for user accounts. If you do this, misbehaving processes won't be allowed to take down the whole machine.

Deploy a host firewall. Not only will this protect against misconfiguration at the central firewall, but will protect the machine from the hostile traffic on the internal network. Deny everything by default, then allow only the ports that are really needed (e.g. 80 and 443 from the public, 22 and 3389 from the administrative stations). Although this step may take a bit longer to complete it will substantially improve the security of the host and the network as a whole.

Securing Oracle Application Server

Deploy file integrity monitoring. This task, too, can require a significant amount of time, but it is the only way to know for sure if you have been infiltrated or not.

Network Security

Taking another step back from Oracle Application Server leads us to network security. Network security is a vast subject - we will not attempt to cover it, not even in bullet points. Instead, we will look at the mechanisms supported by Oracle Application Server to increase security by breaking up the system into standalone components. This is only a brief overview of what is well-explained in the "Recommended Deployment Topologies" of the Oracle Application Server Security Guide.

Although all components are installed onto the same machine by default, you don't have to accept this as your deployment architecture. Most Oracle Application Server components can be installed in a standalone format from the Application Server Companion CD. For reasons of scalability and security you should consider splitting the various components onto completely separate machines, and even completely separate network segments. If your installation size and budget do not allow for separate machines, consider using virtualization technologies to maintain separation while saving on hardware costs.

Securing Oracle Application Server

The basic idea behind component isolation is to make intrusion containment possible. Your goal is to separate the exposed components with low value (e.g. Web Cache) from more private components with high value (e.g. database). On a hardened network a compromised component in the outside layer can be quickly replaced with minimal damage and downtime. If the network is not hardened, the attacker will quickly move from the outside layer into the inner layer. You can see an example of a secure topology in the Figure 4.

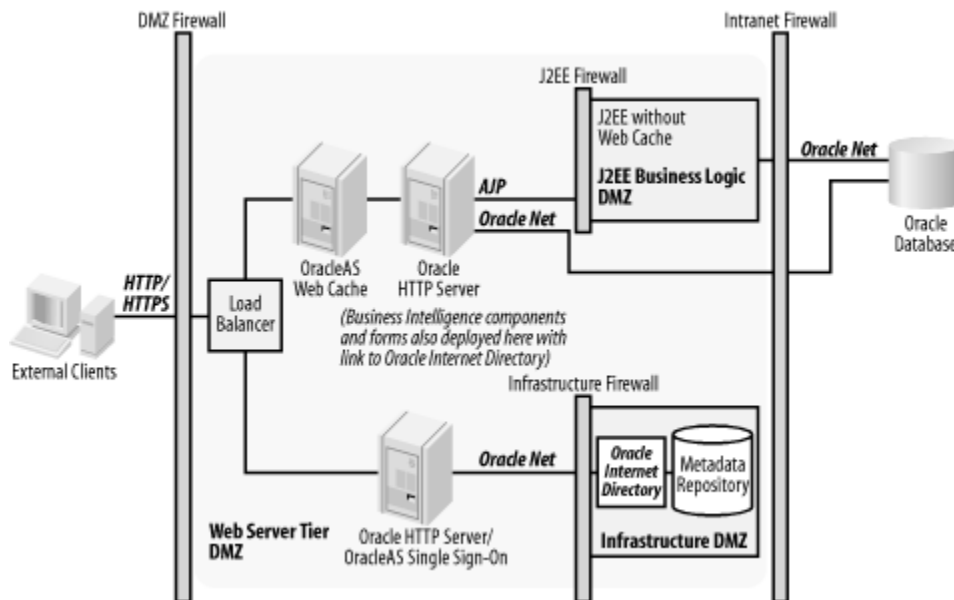


Figure 4: Secure Topology

A rough step-by-step guide to securing Oracle Application Server on the network level is as follows:

Securing Oracle Application Server

*Group components
according to their value*

There is little value to gain from separating components of the same value. As seen above, the Web Cache and HTTP Server share a network segment. This is all right because their value is roughly the same. The next component further up the value chain is the business logic component (J2EE on the diagram), and you can see it residing in its own segment. Finally, the database that holds all of the data is in a separate segment.

Network segmentation

Put component groups into separate network segments. Restrict traffic flow between segments to allow only what is absolutely necessary.

*Use the port tunneling
tool distributed with OAS*

Do this to reduce the number of ports that must be left open. See "Understanding Port Tunneling" in the Oracle HTTP Server Administrator's Guide for more information.

*Secure inter-component
communication*

Consider the data flow between various components. If the data flows across insecure segments (where someone could eavesdrop) consider using SSL internally. For example, the

Securing Oracle Application Server

communication between HTTP Server and OC4J components can now be encrypted. Even the port tunneling tool supports SSL.

Host firewall

By deploying host firewalls you will further harden the network segments, but you will also protect machines from other machines within the segment.

Conclusion

With the exception of the code examples, the default configuration of Oracle Application Server 10g R2 is generally secure. Working against OAS, however, is the complexity of the product and the large number of components that are installed. To maximize security, administrators need to carefully review each installation before production, and make the right changes to suit their unique circumstances. The key to success is to gain a full understanding of the software stack and the interactions between each of the various components. This document contains step-by-step instructions that point to the areas of interest and provides a framework to secure Oracle Application Server.

There are some areas we would be happy to see improve in the future. Doing so would increase security and make maintenance of Oracle Application Server easier. Suggestions include:

Securing Oracle Application Server

- Providing a cleaner and better-documented Apache configuration.
- Making it easier for different components to run as different users (especially a way to run OC4J containers under different user accounts).
- Making the installation of sample code and other default content an option in the installation process.

Securing Oracle Application Server

The Business Case for Application Security

By their nature, web applications must be accessible to be useful.

Unfortunately, the same avenue that legitimate users travel to visit your web application is also one well worn by attackers. Web application attacks have significant consequences. According to IDC/IBM Systems Sciences Institute, the inability to test and remediate vulnerabilities before an application goes into production leaves confidential data within a web application at great risk for attack and misuse. The losses generated by this security gap are significant and expensive - up to \$60 billion a year. That is only expected to increase with the passage of time.

Whether a security breach is made public or confined internally, the fact that a hacker has accessed your sensitive data should be a huge concern to your company, your shareholders and, most importantly, your customers. S.P.I. Dynamics has found that the majority of companies that are vigilant and proactive in their approach to application security are better protected. In the long run, these companies enjoy a higher return on investment for their e-business ventures. Delivering secure web applications can bolster an organization's bottom line by:

Mitigating risks: What would be the financial cost of a successful attack upon your web application? Of application downtime? Of fines levied because of a breach of private consumer information? From delays in shipping and billing

Securing Oracle Application Server

because of a security incident that led to corruption of data? From a loss of competitive advantage?

Increasing revenues/reducing costs: Web applications tend to rely on publicly accessible functionality as a means of generating revenue. Put simply, no e-business can occur if a site has been disabled by a successful attack. Secure applications are far more likely to stay available, and ultimately serve to increase the confidence of users that their transactions and interactions are being conducted in a safe environment. According to Gartner, implementing security during the development process is a far more cost-effective solution than adding it after deployment. By that point, application complexity and interdependency will serve to increase update costs exponentially.

Meeting policy compliance demands: Most legislative compliance packages that address online security contain these key requirements:

- Corporations must conduct a risk assessment of their applications.
- Web applications must contain the proper authentication, access control, and logging systems to ensure security.
- Ongoing auditing of information systems must be conducted to test for newly discovered vulnerabilities.
- Companies must implement an internal security policy, and show due diligence in following its provisions.

Securing Oracle Application Server

Failure to implement the requirements of compliance legislation such as Sarbanes-Oxley or Gramm-Leach-Bliley could lead to fines, loss of revenues, bad publicity, and more.

About SPI Labs

SPI Labs is the dedicated application security research and testing team of S.P.I. Dynamics. Composed of some of the industry's top security experts, SPI Labs is specifically focused on researching security vulnerabilities at the Web application layer. The SPI Labs mission is to provide objective research to the security community and give organizations concerned with their security practices a method of detecting, remediating, and preventing attacks upon the Web application layer.

SPI Labs continuously maintains the world's largest database of more than 5,000 application layer vulnerabilities and attack methodologies. This direct research is utilized to provide daily updates to S.P.I. Dynamics' suite of security assessment and testing software products. SPI Labs engineers comply with the standards proposed by the Internet Engineering Task Force (IETF) for responsible security vulnerability disclosure. Information regarding SPI Labs policies and procedures for disclosure are outlined on the S.P.I. Dynamics Web site at: <http://www.spidynamics.com/spilabs.html>.

Securing Oracle Application Server

About S.P.I. Dynamics Incorporated

Start Secure. Stay Secure.

Security Assurance Throughout the Application Lifecycle.

S.P.I. Dynamics' suite of Web application security products help organizations build and maintain secure Web applications, preventing attacks that would otherwise go undetected by today's traditional corporate Internet security measures. The company's products enable all phases of the software development lifecycle to collaborate in order to build, test and deploy secure Web applications. In addition, the security assurance provided by these products help Fortune 500 companies and organizations in regulated industries — including financial services, health care and government — protect their sensitive data and comply with legal mandates and regulations regarding privacy and information security. Founded in 2000 by security specialists, S.P.I. Dynamics is privately held with headquarters in Atlanta, Georgia. For more information, visit www.spidynamics.com or call (678) 781-4800.

About the Author

Caleb Sima is co-founder and CTO of S.P.I. Dynamics where he is responsible for directing the lifecycle of the company's Web application security solutions. He is also the director of SPI Labs R&D team within S.P.I. Dynamics. Caleb has been engaged in the Internet security arena since 1996, and has become widely recognized as an expert in the industry. He is a frequent speaker and expert resource for the press on Internet attacks. Caleb is a member of

Securing Oracle Application Server

ISSA, one of the founding visionaries of the AVDL standard within OASIS, and a founding member of the Web Application Security Consortium (WASC).

Contact Information

S.P.I. Dynamics
115 Perimeter Center Place
Suite 1100
Atlanta, GA 30346

Telephone: (678) 781-4800
Fax: (678) 781-4850
Email: info@spidynamics.com
Web: www.spidynamics.com

Securing Oracle Application Server

Appendix A: Oracle Application Server Installation Security Checklist

Use the following checklist as a guide when securing an installation of Oracle Application Server.

Pre-Installation

- Understand the components and the architecture of Oracle Application Server.
- Secure the operating system
- Make a record of active system processes
- Make a record of listening ports

Installation

- Create new user account
- Install Oracle Application Server
- Download and install patches from Metalink
- Start Oracle Application Server
- Examine new processes, listening ports, and the filesystem
- Configure setuid scripts that are needed when using privileged ports

Securing the HTTP Server

- Examine the configuration files in detail, starting with `httpd.conf`
- Remove default content (static pages)

Securing Oracle Application Server

- Remove default examples (code)
- Replace the default home page
- Reduce information leakage
- Deactivate unused HTTP Server modules
- Configure limits
 - Timeout
 - Keep-Alive Timeout
 - Maximum number of Apache instances
 - Limit request body
 - Limit request fields
 - Limit request field size
 - Limit request line size
- Enhance logging
 - Access log
 - Error log
 - Establish access log retention policies
- Disable SSLv2
- Disable weak SSL ciphers
- Specify correct value for UseWebCacheIP

Securing Oracle Application Server

Securing Web Cache

- Secure network-based access controls
- Configure limits
 - Maximum individual header size
 - Maximum combined header size
 - Keep-Alive timeout
 - Maximum cache size
 - Maximum connections per backend server
 - Maximum connections (total)

Advanced Hardening

- Disable SSLv2
- Configure HTTP Server to run as own user
- Configure Web Cache to run as own user
- Configure OC4J instances to run as own user
- Consider using standalone installations

Post-Installation

- Start installation diary
- Initialize file integrity monitoring
- Make a full backup and store it off-site

Maintenance

- Stay informed about the security of the installation
- Patch regularly
- Keep the installation diary up-to-date

Securing Oracle Application Server

Appendix B: Bibliography

- Apache httpd 1.3 documentation (<http://httpd.apache.org/docs/1.3/>).
- *Apache Security* by Ivan Ristic (O'Reilly). Two chapters relevant for this whitepaper are available for free download from <http://www.apachesecurity.net>.
- *Configuring a Secure Oracle 9i Application Server Environment* by Stephen Comstock (Oracle), (<http://otn.oracle.com/deploy/security/oracle9iAS/pdf/securingias.pdf>).
- *Hack-Proofing Oracle9i Application Server* (http://www.oracle.com/technology/oramag/webcolumns/2003/techarticles/newman_hackproof_pt1.html) by Aaron Newman (Application Security).
- *Hackproofing Oracle Application Server* (www.nextgenss.com/papers/hpoas.pdf) by David Litchfield (NGSSoftware).
- *Oracle 9i/10g Benchmark Ver. 2.0*, Center for Internet Security (<http://www.cisecurity.org>).
- *Oracle Application Server 10g Essentials* by Donald Bales, Rick Greenwald, and Robert Stackowiak (O'Reilly).
- *Oracle Application Server 10g R2 Documentation* (<http://www.oracle.com/technology/documentation/appserver1012.html>).
- *Oracle Security: Step-by-Step Version 2.0* by Pete Finnigan (SANS Press).
- PHP Manual: Security (<http://www.php.net/manual/en/security.php>).
- *Special OPS: Host and Network Security for Microsoft, UNIX, and Oracle* by Erik Pace Birkholz et al. (Syngress).