

## **NYOUG EDITOR'S CHOICE AWARD - 2003**

# **Hack-proofing Oracle Application Server**

**Aaron Newman, Application Security, Inc.**

### **Introduction**

An undeniable trend in application development today is the movement toward web applications. Web applications originally lacked the tools to develop rich graphical user interfaces. The advantages of web applications have been attractive enough that any disadvantages have been outweighed. Today the number of people writing applications that are not web enabled is dramatically decreasing.

Advantages of web applications include:

- No need to install application software on client desktops
- Configuration only at source
- Cross platform support for users
- Works over the internet

One of the most promising application servers available is Oracle 9iAS. Oracle Corporation has had an application server available for many years, however until the 9i release it has failed to gain any traction. Older releases of Oracle's Application Server were based on its own proprietary web server that never proved itself as an enterprise solution. In the newer versions, Oracle has dropped its own technology and instead incorporated the Apache webserver into their solution. The change seems to have been a good decision as Oracle 9iAS is now considered a very scalable, enterprise capable application server.

Of course, as 9iAS gains market share, security concerns become very important. By being a market leader, Oracle has also become a target of hackers and security researchers looking for security holes. Oracle Corporation has aggressively worked to provide a secure platform, however administrators must understand that a tool is only secure if it is used securely. This means that administrators must configure the tools properly, establish proper security policies, and ensure the policies are being followed.

### **State of Oracle Security**

Oracle Corporation has taken a strong stance on "hack-proofing" their software. While a literal translation of this marketing campaign could be misinterpreted, this statement should be taken to mean that Oracle Corporation takes security seriously and has made a serious commitment to incorporating security from the ground up. Oracle has put its software through 15 independent security evaluations, ranging from FIPS certification for its encryption to Common Criteria and C2 reviews. This investment evidences Oracle's commitment to securing your data.

Determining if software is secure or not is a tricky task. In the security community, many organizations create an air of fear, uncertainty, and doubt (FUD). As Oracle professionals, it's important to see through the FUD, determine the actual risks, and investigate what can be done about the situation. The truth is that a default configuration of any software is usually hackable. Oracle's software is no different. This is not to say that Oracle application server cannot be properly secured. The onerous lies with the administrators, security officers, and auditors of your company to ensure that the proper lockdown procedures have not been taken and that the security policy exists and is in place.

Security is also about the weakest link. Your network is only as secure as the weakest computer on the network. If you have a secure network with an insecure database or application, the operating system or other devices on the network can be attacked or compromised through the application. Your applications should not provide a point of weakness.

Clearly, our web applications are not going to all be hacked tomorrow from improper security. However, it is important that administrators and security professionals take 9iAS security more seriously. We, as the Oracle community, should start by taking a proactive approach to understanding the risks and securing Oracle application server.

## **Firewalls**

We all know what firewalls are, but in order to understand what a firewall can and can not protect you from, you need to think carefully about what a firewall really does. At the core, a firewall is a filter. It stops packets by passing it through a set of tests and if it fails any of the tests, the packet is dropped. While this sounds relatively simple, for a large enterprise, the set of rules can get extremely complex and configuring the firewall incorrectly can occur. The firewall can be thought of as the “hard crunchy outside” of your network. It is the first level of defense, much like a fence around a house.

First, a firewall is typically configured to block traffic that is not destined for port 80 or 443. Secondly, a firewall is filtered so that it can only reach specific IP addresses. What this does is eliminate attacks on servers other than the web server and reduce the points of attack on the web server to port 80.

## **Application-layer Attacks**

Just because 9iAS sits behind a firewall does not mean that you do not need to worry about it being attacked. There are several other forms of attack that can be made through the firewall. These forms of attacks are known as application layer attacks. They leverage weaknesses in the logic of the application. It can be fairly complicated to eliminate these holes because they are dependent on the implementation details of your application.

- Parameter manipulation
- Cookie Poisoning
- Hidden fields
- Cross-site scripting
- SQL Injection

You should understand from the earlier section on firewalls that because these attacks are done over HTTP and against the web server, there are not blocked by the firewall at all.

## **SQL injection**

The most common of these application-layer attacks today is SQL Injection. SQL Injection is caused by the way in which web applications are developed. They are the product of using techniques that allow SQL to be manipulated in a manner in which it was not intended.

SQL Injection works by attempt to modify the parameters passed to a web application to change the SQL statements that are passed to the database. For instance, you may want the web application to select from the ORDERS table for a specific customer. If the hacker enters a single quote into the field on the web form and then enters another query into the field, it may be possible to cause the second query to execute.

A simple way to verify whether your web application is vulnerable to this attack is to enter a single quote in each field on each form and submit the form. If you get a response that there was a syntax error in the database query, it’s a good indication that you are vulnerable. Some sites will return the error results claiming a syntax error. Some sites will catch the error and not report anything. Of course, these sites are still vulnerable, but they are much harder to exploit if you do not get the feedback from the error messages.

This attack works against any application server, not just Oracle 9iAS. How this attacks works varies slightly depending on the back-end database, but the fundamental problem is the same for all databases and application servers.

## **SQL injection Sample #1**

So how does the exploit work? How would an attacker change a SQL Statement to another SQL statement?

SQL Injection is based on a hacker's attempt to modify a query, such as:

```
select * from my_table where column_x = '1'
to:
Select * from my_table where column_x = '1' UNION select password from DBA_USERS where
'q'='q'
```

In the preceding example, we see a single query being converted into two queries. There are also ways to modify the WHERE criteria to update or delete rows not meant to be updated or deleted. With other databases, you can embed a second command into the query. Oracle does not allow you to do this. Instead, an attacker would need to figure out how to supplement the end of the query. Note the 'q' = 'q' at the end. This is used because we must handle the second single quote that the JSP page is adding onto the end of the query. This clause simply evaluates to TRUE.

Here is an example of a Java Server Page that you might typically find in a web application. Here we have the case of a typical homegrown authentication mechanism used to login to a web site. You must enter your password and your username. Using these two fields, a SQL statement is generated that selects from the tables where the username and password match the input. If a match is found, the user is authenticated. If the recordset in our code is empty, then an invalid username or password must have been provided and the login is denied. Of course, a better idea would be to use the authentication built into the web server, but this form of "home grown" authentication is very common.

Package myseverlets;

```
<...>
String sql = new String("SELECT * FROM WebUsers WHERE Username='" +
request.getParameter("username") + "' AND Password='" + request.getParameter("password")
+ "'")
stmt = Conn.prepareStatement(sql)
Rs = stmt.executeQuery()
```

Exploiting the problem is much simpler if the hacker can access the source of the web page. You should not be able to see the source code, however there bugs in many of the common web servers that allow an attacker to view the source of scripts, and I'm sure there are still lots that have not yet been discovered. Fundamentally, you should not be relying on the fact that the hacker cannot see your applications logic.

The problem with our JSP code is that we can concatenate our SQL statement together without parsing out any single quotes. Parsing out single quotes is a good first step, but it's recommended that you actually use parameterized SQL statements instead.

For the following web page, I set the username to:

Bob

I also set the password to:

Hardtoguesspassword

The SQL statement for these parameters resolves to:

```
SELECT * FROM WebUsers WHERE Username='Bob' AND Password='Hardtoguess'
```

What if an attacker instead of using a regular password enters some letter, uses a single quote to end the string literal, then inserts another Boolean expression in the where clause. Obviously, this Boolean expression is TRUE which returns all the rows in the table. For instance, if an attacker instead enters the password as:

Aa' OR 'A' = 'A

The SQL statement now becomes:

```
SELECT * FROM WebUsers WHERE Username='Bob' AND Password='Aa' OR 'A' = 'A'
```

As you can see, this query will always return all the rows in the database, and the attacker will have convinced the web application that a correct username and password was passed in. The kicker is that when the recordset contains the entire set of users, the first entry in the list will typically be the Administrator of the system, so there is a good chance the attacker will be authenticated with full administrative rights to the application.

## SQL injection Sample #2

Various twists on SQL Injections can also be performed. An attacker can select data other than the rows from the table being selected from by using a UNION. Here's another example of how to pull data back from other tables that are not directly involved in the current query. The best way to exploit this issue is to find a screen that contains a dynamic list of items, such as a list of open orders or the results of a search.

The trick here for the attacker is to make the single query into two queries and UNION them. This is somewhat difficult because you must match up the number of columns and column types. However, if the server provides you the error messages, the task is relatively simple. The error returned will typically be one of the following:

Number of columns do not match

Or:

2nd column in UNION statement does not match the type of the first statement.

We look again at a sample JSP that might typically be found in an application.

Package myserverlets;

```
<...>  
String sql = new String("SELECT * FROM PRODUCT WHERE ProductName='" +  
request.getParameter("product_name") + "'")  
stmt = Conn.prepareStatement(sql)  
Rs = stmt.executeQuery()  
' return the rows to the browser
```

An attacker does not really need the source code, but it does make our lives easier for demonstration purposes. Once again, we are not using parameterized queries, but instead are concatenating strings to build our SQL statement. We try entering valid input by setting the product\_name to:

DVD Player

The SQL Statement is now:

```
SELECT * FROM PRODUCT WHERE ProductName='DVD Player'
```

An attacker would instead want to get a copy of the password hashes from your databases. Once he has these hashes, he can start brute-forcing them. The hacker would set the product\_name to:

```
test' UNION select username, password from dba_users where 'a' = 'a
```

The SQL Statement is now:

```
SELECT * FROM PRODUCT WHERE ProductName='test' UNION select username, password from dba_users where 'a' = 'a'
```

Instead of entering a single word, the attacker used a single quote to end the string literal then added a UNION and a second statement. Notice at the end that he must still handle the fact that the code will place another single quote at the end, so we end our second SQL query with:

```
'a' = 'a'
```

This last clause evaluates to true causing all rows to be returned from the DBA\_USERS table.

## Preventing SQL injection

Preventing SQL injection attacks from happening are simply once you understand the problem. Really, there are two strategies you can use to prevent the attacks.

- Validate user input
- Use parameterized queries

Validating user input involves parsing fields to restrict the valid characters that are accepted. In most cases, fields should only accept alphanumeric characters.

Also you can escape single quotes into 2 single quotes although this method is riskier since it is much easier to miss parsing input somewhere.

Using parameterized queries involves bind variables rather than concatenating SQL statements together as strings. The biggest challenge will be reviewing and updating all the old CGI scripts, JSP pages, etc... in your web applications to remove any instance of this vulnerability. It is also suggested that you setup a programming guideline for web programmers that includes emphasis on using parameterized queries and not constructing SQL by concatenating strings with input values.

## Conclusion

There are a small but growing number of resources out there to keep up with Oracle security. There are a few simple tasks that can be performed to reduce your security risk to a reasonable level.

- Stay patched. To download patches go to <http://metalink.oracle.com>.
- Stay aware of Oracle security hole. You can subscribe to a list to receive an email when a new security alerts is publicized by going to <http://www.oraclesecurity.net/resources/maillinglist.html>.
- To ask questions on Oracle security, check out <http://www.oraclesecurity.net/cgi-bin/ubb/ultimatebb.cgi>.
- Find the best partners and security tools to help you secure 9iAS

It's recommended that you use multiple layers of defense. It is recommended that you use at least the following layers of defense:

- Perform audits and pen tests on your applications on a regular basis
- Encryption of data-in-motion
- Encryption of data-at-rest
- Monitor your log files
- Implement intrusion detection

By staying informed and aware of the security risks, you should be able to keep the risks to a minimum.

Contact Information: Aaron C. Newman  
Application Security, Inc. - Protection Where It Counts -  
www.appsecinc.com  
Phone: 212-420-9270  
Fax: 212-420-9680

**Biography:**

Aaron Newman is the Founder and Chief Technology Officer of Application Security, Inc (ASI). In his current role, Aaron is responsible for defining the overall ASI product vision.

Widely regarded as one of the world's foremost database security experts, Aaron co-authored a book on Oracle security for Oracle press, delivered presentations on database security to various groups, and has written several white papers on database security.

Prior to founding ASI, Aaron founded several other companies in the technology area, including DbSecure, the pioneers in database security vulnerability assessment, and ACN Software Systems, a database security consulting firm. Aaron has spent the last decade managing and designing database security solutions, researching database vulnerabilities, and pioneering new markets in database security.

Aaron also worked for Intrusion Detection where he developed both intrusion detection and security assessment solutions for Windows NT. Aaron has held several other positions in technology consulting with Price Waterhouse, Internet Security Systems, and Banker's Trust.