Developing Secure Web Applications

Oracle WebServer 1.0, 2.0, and 2.1

Introduction

Today, over 60% of all relational databases on the Internet are using Oracle7. Although Oracle7 in itself has the industry's most advanced and comprehensive security model supporting a multitude of authentication mechanisms, security can always be compromised if applications are not carefully designed and deployed in a secure fashion. The technology to do this exists, but it is up to the developer to use it.

It has come to Oracle's attention that many Internet sites using Oracle WebServer are using the simple example applications distributed with the product as skeletons for real applications. The danger in this is that the samples were not designed with maximum security in mind, but rather to demonstrate the basic functionality of the product. To address this issue, Oracle is releasing this white paper documenting how to design secure applications for the web. Many of the simple adjustments described at the end of this paper are automated by a security utility, distributed as a patch for older versions of Oracle WebServer (note that technically speaking this is not a patch, just a script to modify configuration files). You should download this utility if you are running any version of Oracle WebServer prior to 2.1.1. If you do not have web access, please contact Oracle Worldwide Support.

Scope

The scope of this paper is to explain how to design and deploy secure PL/SQL applications using the PL/SQL Cartridge or the PL/SQL Agent CGI program distributed with all releases of Oracle WebServer. Note that all security concerns raised here apply to any language and any system, but PL/SQL is one of the few languages that offers a secure method to avoid exposures. Most systems offer ways of minimizing the risk of unauthorized viewing and/or tampering with data, but this paper will teach you how to remove the risk completely using sound configuration and development guidelines.

This paper assumes familiarity with the Oracle WebServer product and the PL/SQL Cartridge and/or PL/SQL Agent[1] in particular.

What is the Risk ?

As part of the introduction to this paper, let me be perfectly clear about what the two main concerns are. First, there is the risk of allowing unauthorized access to confidential data.

The banal example is of course eavesdropping at the network level, but this is not what this paper is about[2]. Instead, and much more serious, is to allow an unauthorized person to gain access to an Oracle schema[3]. The results of this may be devastating: entire order-tables may be erased, transaction-logs may be modified, user-accounts may be copied, etc. Even worse, if the Oracle account is a privileged account with for instance DBA privileges, the system administrators worst nightmare may very well come true. A malicious user gaining unauthorized access to a DBA account can wipe out an entire database.

The second risk is the inability of an application function to determine whether it is invoked out of context. Although unauthorized users may not gain full access to an Oracle schema, the results may be equally devastating and harder to detect. Imagine that you have a stored procedure that is used by other applications to delete a customer account in your electronic store. If anybody can invoke this procedure directly from a web-browser, it is imperative that the procedure itself can determine whether it has been invoked in a proper context and restrict access accordingly.

[1] This paper will use the terms PL/SQL Cartridge and PL/SQL Agent to describe the Web Request Broker cartridge and CGI implementations respectively.
[2] This is an important issue also, but is easily solved using cryptographic network protocols such as SSL.
[3] A schema is the security domain controlled by one single Oracle user, but is often equated to the complete collection of database objects owned by that user.

Secure PL/SQL Design

This section describes how to design secure PL/SQL applications. Through the nature of the PL/SQL Cartridge and Agent, *any* stored procedure may be considered an application in its own right, even if it doesn't return anything to the user. Therefore, care must be taken to prevent users from invoking procedures that were never intended to be invoked directly. Let's look at an example: the following procedure was developed to encapsulate a table in the database:

```
procedure raise_salary ( p_empno in number, p_sal in number ) is
        old_sal emp.sal%type;
begin
        select sal into old_sal
        from emp
        where empno = p_empno;

        insert into sal_history
        (the_date, empno, old_sal, new_sal)
        values
        (sysdate, p_empno, old_sal, p_sal);

        update emp
        set sal = p_sal
        where empno = p_empno;

end;
```

This is a simple example of using a stored PL/SQL procedure to encapsulate database tables. The reason why this type of data-encapsulation is attractive is that it allows you to implement your business rules directly in the database. In this case, a record of every salary modification is recorded in a history table each time the procedure is invoked. An example of usage is detailed below.

```
procedure salary_form ( p_empno in number, p_sal in number ) is
begin
        .
        .
        .
        if (user_is_manager and user_dept = get_dept(p_empno)) then
                raise_salary ( p_empno, p_sal );
        else
                display_error('Not Authorized');
        end if;
        .
        .
        .
end;
```

This may all seem very fine, but the fundamental problem is that nothing prevents the malicious user from invoking the procedure *raise_salary* directly. Bypassing intended application logic in this manner is not a problem restricted to the PL/SQL language. This loophole exists for all web applications, regardless of language and whether they are implemented using Oracle's Web Request Broker, CGI or a direct API to the HTTP protocol server.

However, Oracle7 provides a bulletproof solution to this dangerous exposure. The first level of protection is obtained by wrapping all procedures in PL/SQL packages and only exposing true web-application entry-points through the public package-specification. The second level of protection is to remove all tables and packages from the DCD-schema[4]. In many cases it is desirable to share the same procedure between multiple packages, in which case the first method will not suffice. Both methods are described in detail in the following.

Using PL/SQL Packages

A PL/SQL package consists of two parts: a package specification declaring all externally accessible objects in the package, and a package body, which contains the internal definitions of all objects in the package. An object in this context may be any of the following:

- **Variable**. Any PL/SQL datatype including arrays[5] may be packaged. A packaged variable maintains its value during the duration of a database session, but may be manipulated just like a local PL/SQL variable. If the variable is declared in the package specification, it may be manipulated directly by any external source. This is generally bad practice, since there is no way of guaranteeing that the variable is manipulated in a valid manner.

- **Function**. A PL/SQL function may be packaged. If the function is declared in the package specification, it is accessible by any external source. The function is always defined in the package body.

- **Procedure**. A PL/SQL procedure may be packaged. If the procedure is declared in the package specification, it is accessible by any external source. The procedure is always defined in the package body.

- **Cursor**. A SQL cursor may be packaged[6]. If the cursor is declared in the package specification, it may be manipulated by any external source.

The package construct is similar to C++ private and public attributes and methods. However, PL/SQL has no concept of “friend” functions, and therefore any object that is not explicitly declared in the package specification is invisible and completely inaccessible

[4] The schema that the PL/SQL Agent or Cartridge connects to.
[5] PL/SQL tables
[6] Currently only SQL SELECT statements may be used in explicit cursors in PL/SQL

from any external source. Let's revisit the previous example, but this time we will use a package to encapsulate both procedures:

```
create package hr as
        procedure salary_form ( p_empno in number, p_sal in number );
end;

create package body hr as

        procedure raise_salary ( p_empno in number, p_sal in number ) is
                old_sal emp.sal%type;
        begin
                select sal into old_sal
                from emp
                where empno = p_empno;

                insert into sal_history
                (the_date, empno, old_sal, new_sal)
                values
                (sysdate, p_empno, old_sal, p_sal);

                update emp
                set sal = p_sal
                where empno = p_empno;

        end;

    procedure salary_form ( p_empno in number, p_sal in number ) is
        begin
                .
                .
                .
                if (user_is_manager and user_dept = get_dept(p_empno)) then
                        raise_salary ( p_empno, p_sal );
                else
                        display_error('Not Authorized');
                end if;
                .
                .
                .
        end;

end;
```

The procedures shown here are exactly the same as earlier. The only difference is that the procedure *salary_form* must now be prefixed by the package name: *hr.salary_form*. This will work fine with the Oracle WebServer, but if it causes problems for other reasons, a synonym may be created for the packaged procedure:

```
create synonym salary_form for hr.salary_form;
```

The important thing is that the *raise_salary* procedure is now completely encapsulated within the package since it is not declared in the package specification. It is no longer accessible from external sources, and may only be invoked by other functions and procedures within the package itself. There are no exceptions to this rule. Even if you are connected to the database as the SYS or SYSTEM user with full database system privileges, any attempt to access the packaged *raise_salary* procedure directly will fail.

A similar approach may be taken for all procedures and functions. In general, all objects should be packaged for maximum security when using PL/SQL in stored procedures. Note

that this does not only apply to Oracle WebServer applications, but to any application accessing the database. Remember, whether you like it or not, all procedures and functions that are either declared in package specifications or not packaged at all will be fully accessible by anybody with access to the database schema in question. This includes possibly unauthorized users accessing the schema via the Oracle WebServer.

Using Multiple Schemas

In complex applications it may not be possible to fully encapsulate all private procedures in a single PL/SQL package. Also, if a procedure is to be shared between multiple packages, it must be declared in a public specification or not be packaged at all. This may defeat the security requirements addressed in the previous section. The next level of security involves using multiple schemas to separate in effect public and private objects.

Oracle's security model is distributed, meaning that the creator of any object determines what access control mechanisms should be used to prevent unauthorized access to the object. Access privileges are thus granted at the object level, and may be given to either individual Oracle users or to Oracle roles. The set of available access privileges differ slightly depending on which type of object is being referred to, the most common being:

- select, insert, update, delete privileges for a table or view
- execute privileges for a procedure, function, or package[7]

A very important feature of stored program units is that they always execute with the privileges of their creator (owner). This enables a user to grant other users the privilege to execute a program unit which in turn accesses a table without granting the user any permissions on the table whatsoever. This also applies to other program units. E.g. if procedure A invokes procedure B, and the owner grants execute privileges on A to a different user, that user will be able to invoke B indirectly through A, but never directly. Similarly, if procedure A accesses table X, the user to whom execute privileges on A has been granted is able to access X indirectly through A, but never directly.

[7] Collectively referred to as stored program units

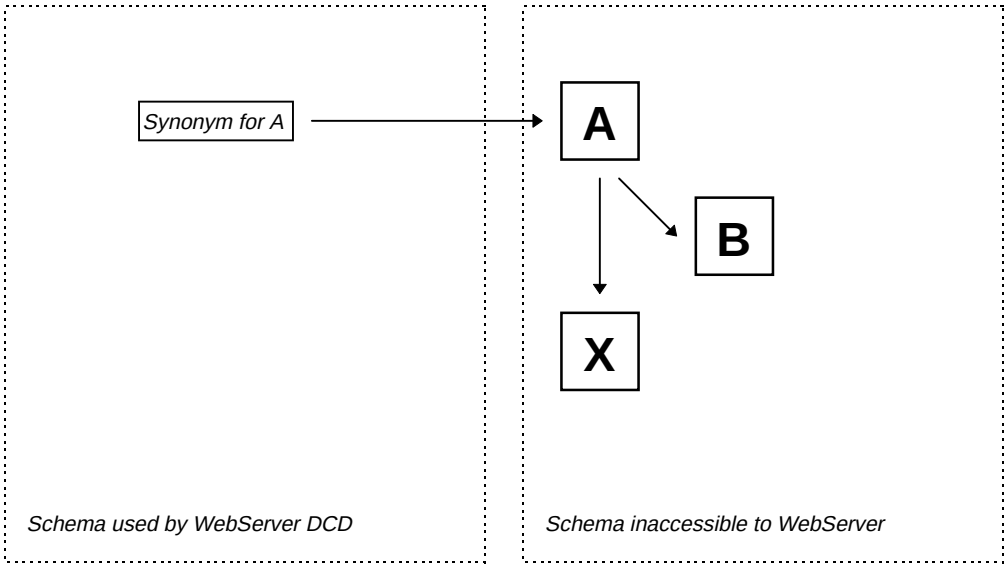

Fig 1: The WebServer DCD schema has been granted execute privileges on procedure A.

This can be used to further enhance security and does not impose the same strict requirements on encapsulating sensitive procedures and functions in packages. In fact, this technique makes it possible to define different access levels to the same schema by defining multiple DCDs with access to different sets of procedures.

Using the Oracle WebServer PL/SQL Toolkit

The key to setting up your system to run PL/SQL application through Oracle WebServer with maximum security is understanding how the PL/SQL Cartridge and PL/SQL Agent work. Functionally, these two components are nearly identical[8] in that they both make use of the same toolkit packages in the database to generate dynamic HTML. The toolkit consists of various functions and procedures but the key procedure is called *htp.print*. This procedures is the delivery vehicle for getting content back to the client. In the current implementation[9], using multiple schemas for your application has a direct impact on how this key procedure may be used.

The *htp.print* procedure places content into a buffer implemented as a packaged PL/SQL table (essentially equivalent to an array of character strings), and the PL/SQL Cartridge or PL/SQL Agent retrieves this buffer and sends it back to the client. This buffer is kept in the *OWA* package, and is expected to exist in the DCD schema that is being accessed. However, if you examine the previous diagram, the schema containing the actual

[8] The Web Request Broker in WebServer 2.0 and higher allows a finer level of granularity in setting up HTTP access control to stored procedures.
[9] Applies to release 1, 2, and 3 of Oracle WebServer

application will expect the PL/SQL toolkit[10] to reside in the same schema. Using two different sets of PL/SQL toolkit packages in each schema will not work, since the applications would then be writing to a different buffer than what the PL/SQL Cartridge or Agent would expect.

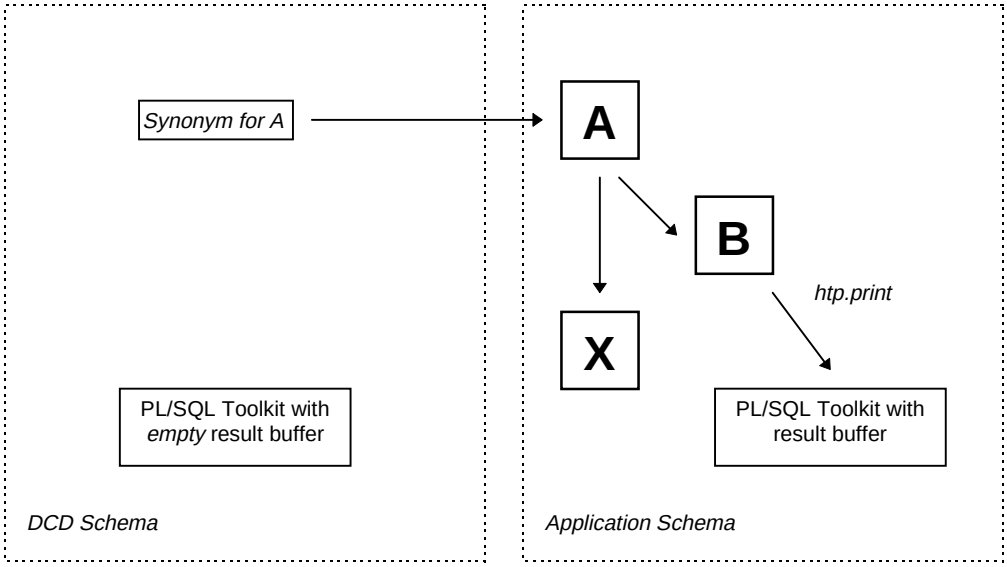

Fig. 2: Installing the PL/SQL web toolkit in each schema will not work.

There are several solutions to this problem. One is to modify the PL/SQL packages directly, which is certainly possible since Oracle includes the full source-code in every distribution of the Oracle WebServer. The drawback of doing this is that you will have to repeat the exercise with each new release. A better alternative is available, which includes maximum flexibility and very little modifications to the standard distribution. The recommended procedure involves using a third “toolkit” schema as shown in the next diagram.

The major benefit of this approach is that multiple DCDs can share the same PL/SQL toolkit and thus reduce overall memory usage. Another benefit is that the two schemas may reside in different databases (even on different nodes). To minimize network traffic, the “toolkit” schema should be installed close to the “application” schema. Finally, this approach does not require any changes to the application when new versions of the Oracle WebServer are installed.

[10] For example the HTP and HTF procedures and functions

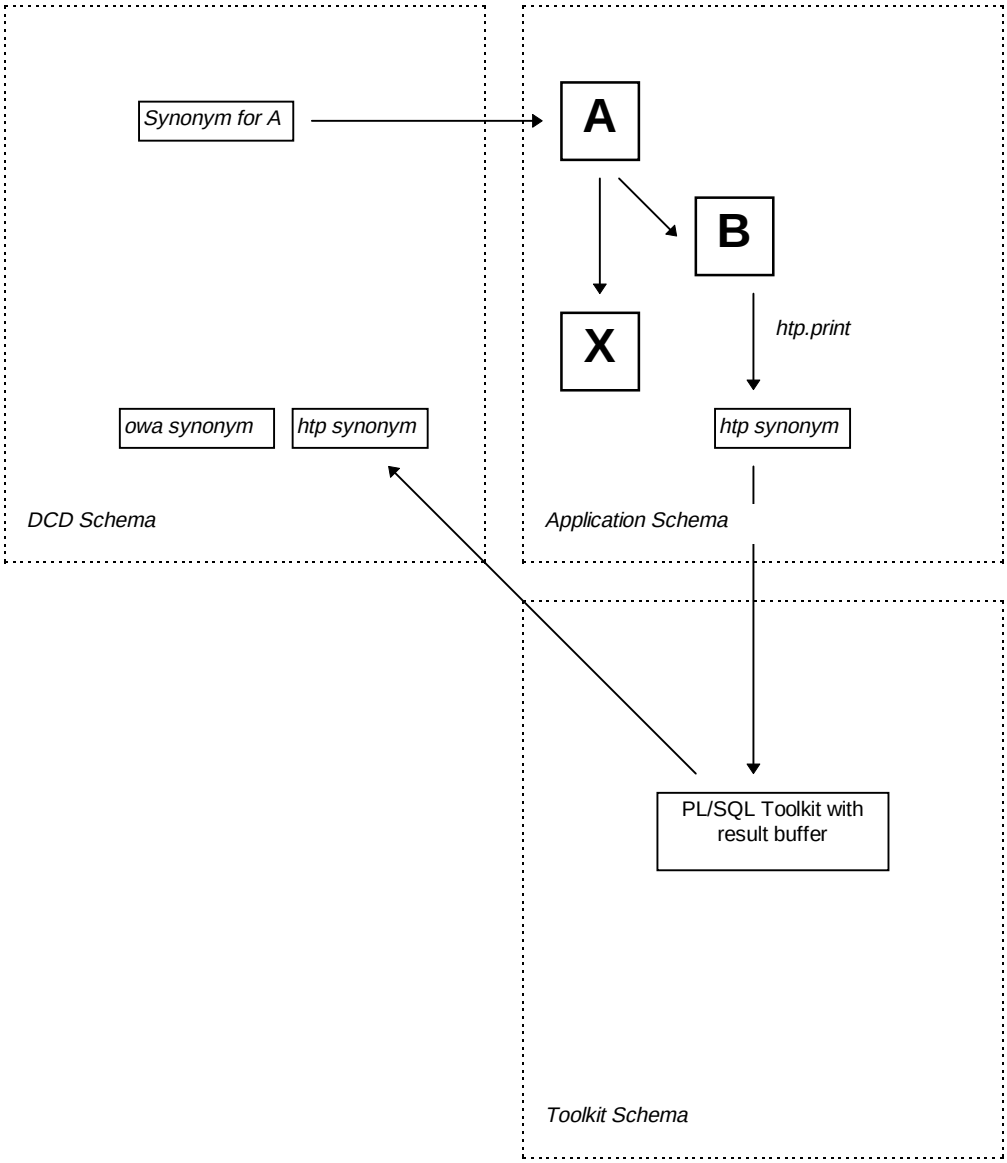

Fig. 3: Using three schemas is the best solution since it offers maximum security and flexibility.

To configure your system to use three schemas, you should follow these steps:

1. Create a standard PL/SQL DCD using the HTML-based WebServer administration utility, but do *not* install the PL/SQL toolkit (there is a checkbox for this).

2. Install the PL/SQL web toolkit manually in a third schema. This may be done by running the file *owains.sql* located in the *$ORACLE_HOME/ows2/admin* directory.

3. Grant execute permissions on the relevant packages to your existing application schema and the DCD schema. It is mandatory to grant access to the owa package to the DCD schema, and access to the htp and htf packages to the

application schema. All other packages are optional (and some are not recommended as you will see in the next section).

4. Create synonyms for the toolkit packages you choose to expose in each of the other schemas.

5. For each application package or procedure in the application schema that you wish to web-enable, you must explicitly grant execute permission to the DCD schema and create a synonym for it in the DCD schema.

Although this does enhance security significantly, it is not completely secure, because there is still a risk of your web applications being invoked out of context. This is explained in the following section.

Maintaining Context

The final step in designing a secure PL/SQL application[11] is ensuring that specific procedures are not invoked out of context. At the time of writing, Oracle is currently evaluating how this may be fully automated. But for now, the only possible solution is to programmatically check the context upon invocation in critical procedures. This can be done by storing a random number[12] along with an application context and the username of the authenticated user in a table. The random number is also stored in the client using a standard HTTP cookie or a hidden HTML form field. When a client makes a request, the username and random number is checked against the values previously stored. If these match, it is reasonably safe to assume that the application context previously stored is valid and can be used to determine whether the current procedure is being invoked in proper context.

The following two sample utility function may be used to store and retrieve an application context[13]. For readability, the application context is simply the name of the procedure being invoked. A call to *get_state* should be included at the beginning and a call to *save_state* should be performed at the end of each application procedure. After calling *get_state*, the application should be able to determine whether the context is valid or not.

```
function save_state ( the_context in varchar2 ) return number is
        the_user app_context.who%type;
        the_key app_context.seqno%type;
begin
        the_user := owa_util.get_cgi_env ( 'remote_user' );
```

[11] Again, this applies to all web applications, not just PL/SQL accessed through the Oracle WebServer
[12] Essentially a session ID
[13] Note that the save_state procedure must never be directly accessible to the DCD schema, as this would defeat its purpose by allowing intruders to create their own contexts!

```
        the_key := my_random;
        insert into app_context
        (seqno,who,context)
        values
        (the_key, the_user, the_context);
        return the_key;
end;

function get_state ( the_key in number ) return varchar2 is
        the_user app_context.who%type
        the_context app_context.context%type;
begin
        the_user := owa_util.get_cgi_env ( 'remote_user' );
        select context into the_context
        from app_context
        where seqno = the_key
          and who = the_user;
        return the_context;
exception
        when no_data_found then
                return 'INVALID';
        when others then
                raise;
end;
```

An example of using these two utility functions follows. The procedure *raise_salary* knows that the only proper context in which it may be invoked is from the *salary_form* procedure, and it therefore verifies that this is indeed the current context before allowing processing to continue.

```
        procedure raise_salary ( p_empno in number, p_sal in number ) is


        old_sal emp.sal%type;
        the_key app_context.seqno%type;
begin
        the_key := to_number ( owa_cookie.get ( 'hr_app_context' ));
        if upper( get_state ( the_key )) != 'SALARY_FORM' then
                raise_application_error( -20001, 'Invalid Context' );
        end if;
        select sal into old_sal
        from emp
        where empno = p_empno;

        insert into sal_history
        (the_date, empno, old_sal, new_sal)
        values
        (sysdate, p_empno, old_sal, p_sal);

        update emp
        set sal = p_sal
        where empno = p_empno;

end;
```

Although the random number can theoretically be spoofed, the application context itself is inaccessible from the client and can not be modified. Also, note that this method assumes that the user has been authenticated using standard HTTP authentication. Therefore, unauthorized users are prevented from accessing any part of the application, and authorized users are programmatically prevented from running a procedure if the context is invalid. A simple example of this type of context checking has been demonstrated in the sample “Travel Demo”, which has been shipping as part of Oracle WebServer since version 1.0.

Summary

This concludes the first section of this paper. In the previous you have been introduced to methodologies that enable you to create secure applications using the Oracle WebServer and PL/SQL. The next section explains how PL/SQL applications may be protected from unauthorized access using Basic Authentication in the Oracle WebServer.

Protecting Your Applications

One of the great things about web technology, and also certainly one of the major threats, is the whole concept of anonymous users. There are several million potential users of your application out there, and some of them have destructive intentions for various reasons not discussed here. The point is that there are most likely applications that you are deploying or thinking of developing, that you wish to constrain access to. This section tells you how to do it.

Virtual Paths

For historic reasons, all web servers operate with virtual paths that translate into physical file locations. In the simplest example, a web server may be configured to have its root filesystem in */usr/local/httpd/docs*, in which case a request for the virtual path */info/index.html* would translate into the physical path */usr/local/httpd/docs/info/index.html*.

To limit access to these files, different web servers employ slightly different mechanisms, but they all translate into two basic types of access control which may be assigned to virtual paths (or physical paths in some cases):

- Basic Authentication. This method prompts the user for a username and password which are validated against a user database maintained by the web server. Digest Authentication is similar in functionality, but causes the client to transmit the password as an MD5 checksum to the server.

- Domain and hostname filtering. This method examines the IP address or hostname of the client, and compares this to rules allowing or disallowing access to virtual paths according to complete or partial specifications using wildcards.

Both regular files and CGI programs may be protected using either or a combination of both of these methods.

When it comes to PL/SQL applications accessed through the Oracle WebServer, things start getting complicated. First of all, these authentication mechanisms were designed to deal with regular files, not stored procedures. In fact, many web servers, including the Spyglass server distributed as the HTTP server component of the Oracle WebServer, do not allow access control to be configured for objects that do not exist in the filesystem.

Oracle WebServer 1.0

The best we could do in release 1.0 was to protect the PL/SQL Agent CGI program. By defining multiple virtual paths for the same CGI program, and at the same time encoding the virtual path with the name of a DCD, it was possible to configure access control at the DCD level, but not for individual procedures. Here is a sample excerpt from a Spyglass configuration file:

```
[DirMaps]
/home/oracle/ows/bin/   CR      /hr
/home/oracle/ows/bin/   CR      /demo
/home/oracle/ows/bin/   CR      /finance

[Security]
Basic {
        (Users)
                admin: fhjd87dy
                scott: tiger
                system: manager
                martin: fdisu8ddi99
                blake: qwerty
        (Groups)
                dba: admin system
                guests: scott
                clerks: martin blake
        (Realms)
                Admin Server: dba
                Finance: clerks
                Demo: guests
)

[Protection]
/finance Basic(Finance)
/demo Basic(Demo)
```

The first section defines the virtual paths. Note that they all translate into the same physical directory. If a user were to request the path */finance/owa/salary_form*, the access control defined for the Finance realm would be used, and when the PL/SQL Agent started up, it would extract the token preceding the *owa* keyword and look for a DCD with that name.

Oracle WebServer 2.0

When the Web Request Broker (WRB) was introduced, the PL/SQL Agent was replaced by the PL/SQL Cartridge, which reads its configuration from the WRB's configuration file. The main difference between release 1.0 and 2.0 in terms of access control is that a finer granularity was introduced. The PL/SQL Cartridge can now provide access control at the procedure level. The following sample configuration file for the Web Request Broker demonstrates how to configure Basic Authentication for the *empLogin* procedure. Relevant lines have been hilighted. Note that the security realm Finance refers to the HTTP server's security realm from the previous example.

```
[Apps]
OWA /home/oracle/ows21/lib/libndwoa.so      ndwoadinit      0       100
```

```
SSI /home/oracle/ows21/lib/ndwussi.so      ndwussinit      0       100
JAVA /home/oracle/ows21/lib/libjava.so     ojsdinit        0       100
HELLO /home/oracle/ows21/sample/wrbsdk/helloworld.so testentry 0   100
MYAPP /home/oracle/ows21/sample/wrbsdk/mywrbapp/mywrbapp.so MyWRBApp_Entry 0 100
;
[AppDirs]
/ssi                 SSI               /home/oracle/ows21/sample/ssi
/hr/owa              OWA               /home/oracle/ows21/bin
/tr/owa              OWA               /home/oracle/ows21/bin
/java                JAVA              /home/oracle/ows21/java
/sbjava              JAVA              /private/home/sbutton/Java_Apps
/sample/wrbsdk/hello HELLO             /home/oracle/ows21
/sample/ssi          SSI               /home/oracle/ows21/sample/ssi
/sample/java/run     JAVA              /home/oracle/ows21/sample/java
/mywrbapp/bin        MYAPP             /home/oracle/ows21/
/owa_dba/owa         OWA               /home/oracle/ows21/bin
/docs/owa            OWA               /home/oracle/ows21/bin
;
[SSI]
EnableLiveHTML     = TRUE
ParseHTMLExtn      = FALSE
EnableExecTag      = TRUE
;
[JAVA]
CLASSPATH          =
/home/oracle/ows21/java/classes.zip:/home/oracle/ows21/java/oracle.zip:/home/oracl
e/ows21/java
LD_LIBRARY_PATH    = /home/oracle/ows21/lib
;
[MYAPP]
state              = CA
tax                = 8.25
;
[AppProtection]
/docs/owa/empLogin Basic(Finance)
```

Note that in release 2.0 and higher, any cartridge application may be protected in this manner.

Minimizing the Risks

If all the guidelines for application design and deployment described this paper were followed to the letter, Oracle WebServer would provide a secure environment. However, to further ensure the integrity of your system, Oracle has recently started recommending that due diligence be applied in the area of access control. There are a number of aspects of the default configuration of Oracle WebServer that make it very easy for an intruder to gain unauthorized access to applications believed to be secure. Or put more correctly: there are some aspects of the Oracle WebServer's default installation that make it very easy for an intruder to *determine* if and how intrusion is possible. Oracle is very concerned with Internet security, and is therefore distributing a utility to all current versions that will make it much more difficult to breach security even if user-written applications still contain security flaws. Please contact Oracle Worldwide Support to get the security utility for your operating system. Details on how to manually enhance security are given below.

Note: *None of the following actions will make your applications secure. They will only make it more difficult to gain unauthorized access. A truly secure system can only be obtained by designed your applications correctly according to the first part of this paper.*

The OWA_DBA DCD

This DCD is created during a default installation and runs with DBA privileges in order for the sample database-browser demo to be able to browse the entire database. This is a potential security threat, because if anybody gains unauthorized access to this DBA account, unlimited damage may be inflicted to the database. *Oracle recommends either removing this DCD or stripping it of its privileges.*

The *owa_util.showsource* Procedure

This standard PL/SQL toolkit procedure enables anybody to obtain the full source code to all applications in the DCD schema. This information may subsequently be used to identify weak such as procedures that do not check if they are invoked out of context. *Oracle recommends disabling this procedure*[14]*.*

The *owa_util.tableprint* and *cellsprint*[15] Procedures

These standard PL/SQL toolkit procedures enable anybody to inspect all rows and columns of any table accessible to the DCD schema. If you store sensitive information, such as credit card numbers, in any table accessible to your DCD schema, these procedures are a security threat. *Oracle recommends disabling these procedure in WebServer 1.0, and protecting them in release 2.0 and higher*.

Conclusion

As demonstrated in this paper, the Oracle WebServer and Oracle7 database is designed to allow maximum security and integrity. But like any other system, this is undone if applications are not designed properly with security in mind. If you have comments about this paper or if you have discovered anything that you believe is a security flaw, please contact us directly at owshelp@us.oracle.com.

[14] The security utility mentioned earlier also disables any PL/SQL toolkit procedures to be invoked directly from a client.

[15] The cellsprint procedure is derived from the owa_sql package distributed by Thomas Kyte in Oracle Government. The owa_sql package has also been widely distributed through consulting and constitutes a similar risk.