Secure Coding.
Practical steps to defend your web apps.

Copyright SANS Institute
Author Retains Full Rights

This paper is from the SANS Software Security site. Reposting is not permitted without express written permission.

Interested in learning more?
Check out the list of upcoming events offering "()"
at http://software-security.sans.orghttp://software-security.sans.org/events/
Introduction

Encryption is an important control used to protect private data, intellectual property and other sensitive data from external and internal attackers. With network perimeters dissolving, encryption is the one control that can prevent attackers from accessing sensitive data when other controls fail. Moreover, encryption of sensitive data is becoming a clear requirement for several compliance regulations.

Encryption traditionally applies to either data at rest or data in flight, and both crucial states of data are usually managed with different solutions. Oracle Advanced Security with Oracle Database 11g provides full-path encryption for database data while the data is on the network and at rest within database storage systems.

This paper is a review of Oracle Advanced Security encryption. It covers important product capabilities including network encryption for data in flight and Transparent Data Encryption (TDE) for data at rest. Although not covered in this paper, Oracle Advanced Security also provides support for strong user authentication to the database utilizing techniques such as Kerberos and Public Key Infrastructure (PKI).

Overall, Oracle Advanced Security encryption was easy to deploy in the testing environment, with nearly immediate usability. The solution provides built-in key management and common-sense configuration options such as encrypting in the database at the column or tablespace level, depending on business needs.
About the Oracle Database Test Configuration

Oracle Advanced Security network encryption can be used with all of the currently supported versions of the Oracle database. Oracle Advanced Security TDE, however, was introduced for column-level encryption with Oracle Database 10g Release 2. Oracle Advanced Security TDE tablespace-level encryption is supported in Oracle Database 11g Release 1 and higher. For this reason, we selected an Oracle Database 11g database for testing.

For data-at-rest encryption, we used TDE tablespace encryption. Although Oracle Database 11g also supports column-level encryption, we reviewed tablespace encryption because that is Oracle's default recommendation starting with Oracle Database 11g. For review purposes, we chose the command line interface to encrypt data; however, Oracle Enterprise Manager supports an easy-to-use graphical user interface.

For data-in-flight encryption, we used Oracle Advanced Security native network encryption for convenience, because it does not require PKI certificates. Note that Oracle fully supports standard SSL/TSL network encryption.
To begin the review, we captured data that was not encrypted over the network, which is the default configuration for Oracle Databases that do not have Oracle Advanced Security protection. Next, we ran two SQL statements to query data from the `banking.customer` table and to create a user, as shown in Figure 1.

![Figure 1: Querying Data without Encryption](image-url)
The tcpdump utility on the database server captured the SQL statements from network traffic. The output shown in Figure 2 provides some of the data captured with tcpdump.

```
1640502230:1640502453(223) ack 1628512343 win 386 <nop,nop,timestamp 3088081 3047240>
 0x0000:  4500 0113 120a 4000 4006 fa02 c0a8 d643 E....@...C
 0x0010:  c0a8 d643 1ffe 05f1 61c8 13d6 6111 2057 ...C...a...a.W
 0x0020:  8018 0182 2ede 0000 0101 080a 002f 1ed1 ........../.
 0x0030:  002e 7f48 00df 0000 0600 0000 0000 1169 ..H........i
 0x0040:  00e4 0000 0000 0000 0000 0000 0000 0000 ..........^.
 0x0050:  002f 1ed6 00e4 0000 0600 0000 00001169 ..........^.
 0x0060:  00e4 0000 0000 0000 0000 0000 0000 0000 ..........^.
 0x0070:  00e4 0000 0000 0000 0000 0000 0000 0000 ..........^.
 0x0080:  00e4 0000 0000 0000 0000 0000 0000 0000 ..........^.
 0x0090:  00e4 0000 0000 0000 0000 0000 0000 0000 ..........^.
 0x00a0:  00e4 0000 0000 0000 0000 0000 0000 0000 ..........^.
 0x00b0:  00e4 0000 0000 0000 0000 0000 0000 0000 ..........^.
 0x00c0:  00e4 0000 0000 0000 0000 0000 0000 0000 ..........^.
 0x00d0:  00e4 0000 0000 0000 0000 0000 0000 0000 ..........^.
 0x00e0:  00e4 0000 0000 0000 0000 0000 0000 0000 ..........^.
 0x00f0:  00e4 0000 0000 0000 0000 0000 0000 0000 ..........^.
 0x0100:  00e4 0000 0000 0000 0000 0000 0000 0000 ..........^.
 0x0110:  00e4 0000 0000 0000 0000 0000 0000 0000 ..........^.

NOTE: Some data has been discarded due to length.
```

Figure 2: Unencrypted Data Captured with tcpdump
As you can see in Figure 2, the traffic was not encrypted, and the SQL statements were easy to read with just a small amount of additional capturing.

After verifying that the network data was being transmitted in the clear, it was time to conduct the same test with encryption enabled. To set up network encryption, you must first modify the `sqlnet.ora` file. In this case, we used the SQLNET.CRYPTO_CHECKSUM_SERVER and SQLNET.ENCRYPTION_SERVER parameters to specify that integrity (CHECKSUM) or encryption (ENCRYPTION) were used. You can set the parameters to REJECTED, ACCEPTED, REQUESTED or REQUIRED. We set the values to REQUIRED.

Once the checksum and encryption parameters were setup, we specified the algorithms to be used. In this case, we set the parameter SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER to MD5 and the SQLNET.ENCRYPTION_TYPES_SERVER to DES40, RC4_40 encryption. Other encryption algorithms, including AES, 3DES and RSA RC4 could all be leveraged depending on needs. In production systems with defined security requirements, you would typically select options like AES128 and SHA-1 for improved cryptographic strength.

Once again, we ran the same SQL statements to query data from the `banking.customer` table and create a user. However, this time, no cleartext data was returned. All data was encrypted when the output was viewed. The output is provided in Figure 3.

```
06:28:01.527467 IP dbsecurity.oracle.com.18321 >dbsecurity.oracle.com.ncube-lm: 
P 2792682313:2792682541(228) ack 2796218916 win 386 <nop,nop,timestamp 32483036 32457536>
  0x0000: 4500 0118 a02b 4000 4006 6bdc c0a8 d643 E....+@.@.k....C
  0x0010: c0a8 d643 4791 05f1 a674 f349 a6aa ea24 ..CG....l...$
  0x0020: 0818 1082 2ee3 0000 0101 080a 01ef a6dc ..........
  0x0030: 01ef 4340 00e4 0000 0600 0000 0000be18 ..C@........
  0x0040: 2408 81b4 b031 c2e9 b166 21ea 23f4 bb05 S......
  0x0050: aee645b b018 c0ad 6271 1d3d de96 68fd ..d[.bq......h.
  0x0060: eb64 93fc a4a7 e507 9b71 1685 f7a0 d70e .d.......q......
  0x0070: 459d 6c34 fdd8 2940 84b9 c1b8 1b33 3bcb E.l4..)...3<
  0x0080: 4da2 d862 096d 34e0 9a22 fa55 27a0 2c5a M..m4..."U",Z
  0x0090: a07a e20f 03f9 0839 620b 40a7 8bd5 beaa :z.....9i.@..]
  0x00a0: 9886 cfb8 3b1a 9ae8 e5fd da7e f080 02fa .......~......
  0x00b0: 7922 72b7 ce1a 20cf 9fbc 9835 480b 7d8b y"r......5H}.}
  0x00c0: bbf7 7f0a 9a23 3438 2aad 86b3 759b a66f .....#48*..u..o
  0x00d0: 51d9 3bb6 05b4 04d3 5ed5 d5b4 5e8d 6221 Q;......^...^...b!
  0x00e0: f79d 459e f651 c57f 7e7e 0b06 97fd 5e2a ..E.Q.{~...^*
  0x00f0: e425 d563 cfe4 53a2 c610 aeb3 5e2f 3a46 %.c...S...^\F
  0x0100: e59d f510 f12b aca5 e55e 5b5d 06e5 8ce4 ......+/\}.e...
```

**NOTE:** Some data has been discarded due to length.

Network encryption was easy to setup, and we were able to obtain the benefits in short time. The fastest deployment of network encryption comes from a single configuration change to the `sqlnet.ora` file on the database server. This configuration drives the encryption decision from the server side, using only default settings that exist on the clients. Additional configuration options enable you to drive the encryption decision from the client side or from combined settings taken from both sides.
The second objective of this paper was to review how Oracle Advanced Security TDE encrypts data at rest. Oracle Advanced Security TDE is implemented in the database kernel, providing strong encryption for stored data. With Oracle Advanced Security TDE properly in place, malicious scans of database storage will not reveal encrypted information. From the standpoint of end users and applications, the presence of TDE is completely transparent because executed queries return real data as long as they are properly authorized by database access controls.

Oracle Advanced Security TDE also includes a command-line syntax to rotate the master key. The master key is always stored outside the database, whereas the table and tablespace keys are stored within the database. The master key can be stored in an external Oracle Wallet (a PKCS-12 compliant key management file) or optionally in a Hardware Security Module (HSM) from a certified vendor. When HSMs are used, the master key is exchanged over a standard PKCS-11 interface.

There are two encryption options with TDE: column-level encryption and tablespace encryption. In this review, we implemented tablespace encryption. In order to set up the encryption, we added the following to the sqlnet.ora file:

```
ENCRYPTION_WALLET_LOCATION=
(SOURCE=(METHOD=FILE)(METHOD_DATA=
(DIRECTORY=/home/oracle/wallet))
```

After modifying the sqlnet.ora file, our next step was to create the wallet. To do this, we issued the command “alter system set encryption key identified by “abcdefg12#.”” This created a wallet in the /home/oracle/wallet directory. Next, we ran another SQL command to make sure the wallet was in an open state and ready to encrypt data.
After enabling the database to use encryption, but before actually encrypting any data, we used a hex editor to view the data as it was currently stored in the data files. In its original, unencrypted state, the data in the `banking01.dbf` tablespace file was readable with the hex editor, as shown in Figure 4.

![Figure 4: Viewing Data Files in Their Unencrypted State](image-url)
The next step involved encrypting data in the **banking.customer** table by moving it into a new encrypted tablespace. To accomplish this, we issued the commands:

```sql
SQL> create tablespace bankingENC datafile 
   > '/u01/app/oracle/oradata/orcl/banking01ENC.dbf' size 1m ENCRYPTION 
   > default storage(ENCRIPT);

SQL> alter table banking.customer move tablesapce bankingENC;
```

These commands created an encrypted tablespace and then moved the **banking.customer** table into it. To verify that the destination tablespace was encrypted, we executed a query command, as shown in Figure 5.

![Figure 5: Viewing the Encrypted Tablespace Attributes](image-url)
Next, we searched the encrypted tablespace file for the same data (TEMASEK) that was discovered in the unencrypted tablespace file. Because the data was now being encrypted, the hex editor was unable to find it, as illustrated in Figure 6.

Alternatively, we could have encrypted specific table columns with TDE column-level encryption. Tablespace encryption can be useful when it is difficult to identify where all your sensitive data resides or when you need to encrypt a lot of data. Column-level encryption is useful when you only need to encrypt a few columns in known locations.

Overall, TDE encryption was easy to set up within the database. With just a few commands, we were able to implement database encryption while Oracle automatically setup, managed and stored the encryption keys for the application.
Conclusion

Today’s threats against databases and the private, intellectual property and other sensitive data stored in them call for additional, scalable protection. This includes encryption, the last line of defense to ward off attackers who may be using stolen operating system credentials, man-in-the-middle techniques such as sniffing, and other common attacks aimed at collecting critical database data.

The closest protection for data in transit and data at rest is encryption of the data itself. Such encryption ensures data in the database is not discoverable by unauthorized users and data sniffed on the network is rendered unreadable.

Oracle Advanced Security network encryption and Transparent Data Encryption for database data in motion and in storage has made it easy for users to meet business needs and compliance requirements.

Working closely with Oracle Database 11g, we found the tools to be intuitive to set up and we were able to start using them right away for both database encryption and network encryption. This ease of use eliminates one of the major barriers to organizations implementing encryption and provides immediate protection and compliance for sensitive database data.
Tanya Baccam is a SANS senior instructor as well as a SANS courseware author. She is the current author for the SANS Security 509: Securing Oracle Databases course. Tanya works for Baccam Consulting, where she provides many security consulting services for clients, including system audits, vulnerability and risk assessments, database audits, and web application audits. Today much of her time is spent on the security of databases and applications within organizations. Tanya has also played an integral role in developing multiple business applications. She currently holds the CPA, GCFW, GCIH, CISSP, CISM, CISA and OCP DBA certifications.

SANS would like to thank its sponsor:

 oracle
Upcoming SANS App Sec Training