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.org
http://software-security.sans.org/events/
Data Encryption and Redaction: A Review of Oracle Advanced Security

A SANS Product Review

Written by Dave Shackleford

September 2014

Sponsored by

Oracle
Overview

The need for organizations to protect sensitive information has never been greater. The risks of data breaches and sensitive data exposures are driving organizations to look for solutions, because an increasing amount of data is being stored and processed outside the perimeter, in cloud applications and service environments. Organizations must protect this sensitive data at its heart, in the database.

In the past, database security has focused on protecting data from access by a database administrator (DBA) or other internal users. Although this is still a valid use case, especially in cloud environments where DBAs may be responsible for databases with numerous customer records, the more pressing focus for many is to ensure records are protected from malicious intruders or accidental exposure. Data breaches linked to the loss of backup tapes, disk drives or flash drives can result in the loss of payment card information, personal health information and many other types of data that are sensitive and regulated by compliance mandates such as HIPAA or PCI DSS.

For compliance reasons alone, many organizations have sought to encrypt data; today, the use cases go far beyond compliance. Security teams are looking to encrypt database information as a fundamental control to safeguard the organization if any storage media or data files are stolen or lost.

We had the opportunity to review Oracle Advanced Security for Oracle Database 12c, which offers two main features for protecting sensitive information in databases. The first, Oracle Transparent Data Encryption (Oracle TDE) is a flexible encryption solution that allows for either column encryption or complete tablespace encryption. The second is Oracle Data Redaction, which removes or redacts columns of sensitive data on the fly during output to applications.

We found Oracle Advanced Security's encryption and redaction capabilities to be top-notch. The product has a wide range of features and—after spending some time with the tools and management interface—we were able to easily and transparently encrypt and redact data. In our testing, performance was barely affected at all, making this an attractive option for database administrators as well as security teams.
As one might guess from the name, the concept of transparent data encryption (TDE) enables encryption of individual table columns or an entire tablespace without any special effort on the part of the application designer or end users. When a user (either directly or through an application) inserts data into an encrypted column or tablespace, a TDE-enabled database automatically encrypts the data; when authorized users select the column (or tablespace) the encrypted data is automatically decrypted and returned.

TDE offers several benefits to organizations:

- **Encrypted data is transparently decrypted for the database user.** By storing encrypted data, organizations protect themselves from breaches related to the storage system.

- **Developers and users do not have to create triggers or views to decrypt data.** No special actions are required, which provides a better user experience.

- **Applications don’t need to be modified to handle encrypted data.** The database engine alone manages all encryption and decryption functions.

Our evaluation of Oracle TDE followed the steps for encrypting a specified tablespace and viewing the data before and after the encryption operation.

### Key Management

Key management is perhaps the most critical part of any encryption scheme, so we began our evaluation with this fundamental element. Oracle Advanced Security uses a tiered key management infrastructure, where keys can be stored in a software keystore on a local file system, on a centralized key server, or in a hardware security module (HSM). A software keystore is likely more flexible and initially costs less to implement; however, the security of the software keystore is tied directly to the local file system and the platform where it is installed. This risk in this case is largely mitigated with implementation of HSM platforms, but these can be more expensive to implement and may be incompatible with some applications.

---

1 Oracle Key Vault, the company’s centralized key management platform ([www.oracle.com/us/products/database/security/key-vault/overview/index.html](http://www.oracle.com/us/products/database/security/key-vault/overview/index.html)), was released after our testing concluded and is not part of this review.
An Oracle wallet is a software keystore that typically contains master encryption keys, TLS certificates, private keys and the Oracle Secure External Password Store (SEPS), which stores user and password information for automating database server logins; it can be one of the following types:

- **Password-protected keystores.** These are secured with a password that you create. You must open the keystore before the keys can be retrieved or used. This is a simple type of keystore to generate that still affords some access control security.

- **Auto-login keystores.** These are automatically opened when accessed. Auto-login keystores don't need to be explicitly opened by a security administrator and are, therefore, less secure than password-based keystores, since they do not have any explicit access control measures built in, relying instead on the file system's permissions.

- **Auto-login local keystores.** These are, naturally enough, auto-login keystores that cannot be opened on any computer other than the one on which they are created. Thus, even if they are stolen, they cannot be used elsewhere.

Encrypting data using Oracle TDE starts with creating a keystore file to store the master encryption key by using the `ADMINISTER KEY MANAGEMENT` SQL command; as part of this, the keystore is secured with a password of your choice. Once the keystore was created, we could then use the password to open the keystore that enables the database to access the master encryption key.

When a user writes data to an encrypted tablespace or column, Oracle Database:

1. Retrieves the master key from the keystore (performed only the first time the keystore is opened, because the key is cached for continued use).
2. Decrypts the specific encryption key associated with the column or tablespace using the master key (again, the key is cached after the initial query of the tablespace).
3. Uses the encryption key to encrypt the data entered by the user.
4. Stores the data in encrypted format in the database.

If the user is selecting data, Oracle Database follows the same steps, but decrypts the data and then returns the original data.
We found encrypting with Oracle TDE to have a minimal impact on performance, although the method employed can have an effect. Column encryption affects performance only when data is retrieved from or inserted into an encrypted column. Tablespace encryption provides better performance than column encryption because Oracle Database encrypts and decrypts at the I/O block layer. Once blocks are decrypted, they are cached in Oracle Database's working memory for optimal performance. (Although we did not test performance in our review, Oracle claims its customers report minimal performance impact when using TDE.)

To evaluate Oracle TDE's key management, we logged into a test environment the company provided for us. First, we validated the encryption keys and keystore that Oracle's lab team had set up for our review, then set the keystore to "open" status to receive data, and the master key was made available for encryption and decryption operations, as shown in Figure 1.

```
SYS@<db> SQL> alter session set container=<db>$root;
Session altered.
SYS@<db> SQL> select status from v$encryption_wallet;
STATUS --------------------------------------- CLOSED
SYS@<db> SQL> administer key management set keystore open identified by "Oracle123";
keystore altered.
SYS@<db> SQL> select status from v$encryption_wallet;
STATUS --------------------------------------- OPEN
SYS@<db> SQL> alter session set container=<db>$pdb1;
Session altered.
SYS@<db> SQL> select status from v$encryption_wallet;
STATUS --------------------------------------- CLOSED
SYS@<db> SQL> administer key management set keystore open identified by "Oracle123";
keystore altered.
SYS@<db> SQL> select status from v$encryption_wallet;
STATUS --------------------------------------- OPEN
```

**Figure 1. Opening Keystore and Readying Master Key**

In these steps, the master key was stored in the Oracle wallet on the local platform, which is secured with an administrator password for access control. This keystore was available for encryption operations.

\(^2\) Naturally, encrypted data requires more storage space than plain text data; a single column requires 32 to 48 bytes of additional storage for each row when encrypted.
After logging into the environment, we pointed a browser to the management interface—Oracle Enterprise Manager Cloud Control—where we could manage database configurations as well as the Oracle Advanced Security settings. After logging in and selecting the database “PDB1” (created as part of the testbed setup), we used the top menu bar controls to select (in order) Administration, Security and, finally, Oracle Advanced Security, enabling us to review the selected database’s encryption keys and status. The encryption keystore and keys for PDB1 are shown in Figure 2.

The next step in our walkthrough was to create a new encrypted tablespace to hold our sensitive data. This was a relatively simple process from the menu bar: in sequence, we selected Administration, Storage and, finally, Tablespaces. In this screen, we created a new tablespace called HR ENC to hold encrypted human resources (HR) data in the PDB1 database.
Setting the encryption options for this tablespace was also simple, as shown in Figure 3.

**Figure 3. Setting Tablespace Encryption Options**

For this example, we chose to use the Advanced Encryption Standard algorithm with 256-bit encryption (AES-256); the default is AES-192. We were also able to review the commands used to generate the encrypted tablespace, as shown in Figure 4.

**Figure 4. SQL for Encrypted Tablespace Generation**
The newly encrypted tablespace (accessed in command-line operations by the tablespace file `hr_enc.dbf`) is shown in Figure 5.

**Figure 5. Final Encrypted Tablespace**

**Oracle TDE in Action**

Once the encrypted tablespace was ready, we needed to move the sensitive HR data into it by performing a “reorganization” of the existing HR tablespace. Before this, however, we wanted to verify that the existing tablespace could be searched and that data was visible in cleartext. Figure 6 shows a simple command to search for the term `Shareholder` in the `DEPARTMENTS_TOO` tablespace.

```
bash-4.1$ strings /app/oracle/dbsec12c/oradata/cdb/pdb1/example01.dbf | grep -n Shareholder
1618:Shareholder Services
762806:Shareholder Services
bash-4.1$
```

**Figure 6. Cleartext Data in Existing Tablespace**
Having verified searchability and visibility, we performed the tablespace reorganization, shown in Figure 7.

![Figure 7. Reorganizing the Tablespace](image1)

The resulting table (DEPARTMENTS_TOO) was moved to the HR_ENC tablespace, as shown in Figure 8.

![Figure 8. HR Table in the Encrypted Tablespace](image2)

After moving the data into the encrypted tablespace, we ran a generic `strings` query against the new encrypted tablespace file. The results, shown in Figure 9, show that no cleartext discernible words were present in the table (having been encrypted).

![Figure 9. Encrypted Tablespace](image3)

All in all, we found the process and tools available with Oracle TDE to be simple and easy to use; we did not have to change the applications on our testbed at all to take advantage of its features. Many more granular options are available as well, depending on the type of encryption operations desired.
Oracle Data Redaction gives security teams the ability to perform on-the-fly redaction of sensitive data in query results prior to display by applications, preventing unauthorized application users from viewing sensitive data. For example, a customer relationship management (CRM) application should return only nonsensitive data to a call center team and redact sensitive or personally identifiable information such as birth dates or Social Security numbers.

Even when the source code is available, changing the application to redact data completely can be error-prone, laborious and a drag on performance. When the redaction tools are built into the database platform—as they are with Oracle Advanced Security—stripping out sensitive data fields dynamically can be much more efficient and effective.

Oracle Data Redaction is ideal for organizations that must comply with regulatory or data security requirements that call for masking sensitive data when it is displayed (e.g., PCI DSS requirement 3.3, which covers account number masking).\(^3\) It reduces implementation costs because developers don’t have to modify applications to accommodate different data formats or manage encryption keys. Oracle Data Redaction’s declarative policy functions can apply different data transformations in the form of full, partial or random redaction and do so conditionally based on factors tracked by the database or on external variables. This redaction has no impact on database operations such as clustering, backup and restore, or upgrades and patching; organizations can therefore deploy it without changing their operating procedures.

Oracle Data Redaction policies are enforced directly in the database kernel, and a number of granular options are available to control when redaction is applied, as well as the input and output formats for redaction. Once enabled, policies are enforced immediately, even for active sessions.

---

To evaluate data redaction, we performed another basic test, this time with individual employee records in the testbed’s mock HR database. The original, unedited employee record is shown in Figure 10.

For this phase of our review, our goal was to redact several of the columns in the “Supplemental Data” category of employee records.
We started by creating a new redaction policy, which was done by clicking (again, in sequence) on the Oracle Enterprise Manager pages Administration, Security and, finally, Oracle Data Redaction; we then clicked Create to start a new policy. We proceeded to create a number of policy elements that corresponded to specific columns in the HR database; Oracle Data Redaction has a number of out-of-the-box pattern matches for sensitive data, as shown in Figure 11.

![Figure 11. Redaction Function Pattern Matches](image)

Custom redaction functions can also be included; specifics about the output of redaction functions are granular and simple to apply and verify. In Figure 12, you can see how the redaction function was set up to follow specific patterns in the TAXPAYER_ID column.

![Figure 12. Partial Redaction Function](image)
After we created several redaction functions, we had a final policy that redacted several fields in the HR database. In the Supplemental Data category of each employee record, we were looking to redact the values for bonus amount, specifics of the last insurance claim, payment account number and taxpayer ID. All of these except the taxpayer ID (stored in this case as TAXPAYER_ID) are full redaction functions, with the data redacted entirely; in contrast, TAXPAYER_ID is a partial redaction operation, replacing the first five characters with X's.

Once implemented, the new employee record appears as seen in Figure 13.

![Figure 13. Employee Record with Redaction Enabled](image-url)
Another important consideration for our assessment was to ensure the data would remain redacted when read by someone with casual or even unintentional SQL*Plus access. In this instance, we queried the database via SQL*Plus and verified that the same type and format of redaction functions were present; our query returned the redacted output, rather than the real data, as shown in Figure 14.

```
HR@pdb1 SQL>
HR@pdb1 SQL> select username, taxpayer_id from supplemental_data;

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>TAXPAYER_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKING</td>
<td>XXX-XX-9197</td>
</tr>
<tr>
<td>NKOHCHMAR</td>
<td>XXX-XX-6889</td>
</tr>
<tr>
<td>LEHMAN</td>
<td>XXX-XX-6186</td>
</tr>
<tr>
<td>AHUNOLD</td>
<td>XXX-XX-8611</td>
</tr>
<tr>
<td>BERNST</td>
<td>XXX-XX-6746</td>
</tr>
<tr>
<td>DAUSTIN</td>
<td>XXX-XX-4563</td>
</tr>
<tr>
<td>VPATARAL</td>
<td>XXX-XX-9759</td>
</tr>
<tr>
<td>DLORENZ</td>
<td>XXX-XX-3219</td>
</tr>
<tr>
<td>NGREENE</td>
<td>XXX-XX-2841</td>
</tr>
<tr>
<td>DAFAVET</td>
<td>XXX-XX-5931</td>
</tr>
<tr>
<td>JCHEN</td>
<td>XXX-XX-3828</td>
</tr>
<tr>
<td>ISCIARRA</td>
<td>XXX-XX-4302</td>
</tr>
<tr>
<td>JNURMAN</td>
<td>XXX-XX-8422</td>
</tr>
<tr>
<td>LEPP</td>
<td>XXX-XX-2577</td>
</tr>
<tr>
<td>DRAPERAL</td>
<td>XXX-XX-3299</td>
</tr>
<tr>
<td>AKHOD</td>
<td>XXX-XX-3226</td>
</tr>
<tr>
<td>SBAIDA</td>
<td>XXX-XX-8724</td>
</tr>
<tr>
<td>STOBIA</td>
<td>XXX-XX-5249</td>
</tr>
<tr>
<td>GHIMROD</td>
<td>XXX-XX-3810</td>
</tr>
<tr>
<td>KCOLMERA</td>
<td>XXX-XX-7473</td>
</tr>
<tr>
<td>MKESS</td>
<td>XXX-XX-6656</td>
</tr>
<tr>
<td>AFRIII</td>
<td>XXX-XX-9710</td>
</tr>
<tr>
<td>PFAULIN</td>
<td>XXX-XX-7019</td>
</tr>
<tr>
<td>SULLMAN</td>
<td>XXX-XX-7458</td>
</tr>
<tr>
<td>KNOUGOS</td>
<td>XXX-XX-8601</td>
</tr>
<tr>
<td>PRAYER</td>
<td>XXX-XX-8807</td>
</tr>
</tbody>
</table>
```

**Figure 14. Redaction Demonstrated in SQL*Plus**

The process of creating and applying redaction functions was easy and straightforward. For organizations looking to protect sensitive data without the need for any modifying applications and without any performance penalty, Oracle Data Redaction is a good fit.
Data encryption and redaction are effective means of protecting sensitive data; the problem for many organizations is implementing them without upsetting their existing database schemas or making things extremely difficult for database managers, developers and end users. Oracle Advanced Security provides such protection without causing performance or functional issues with database schemas.

Oracle Advanced Security was easy to configure and implement, and its encryption and redaction functions operated efficiently and securely. Encryption key management was easy to set up, and keys can be stored in a secure wallet or hardware module. Redaction functions were easy to configure and automatically deploy by setting a few parameters.

Oracle makes a declarative policy-based approach to encryption and redaction simple to create, manage and change, thanks to Oracle Advanced Security’s data redaction and transparent data encryption features. In addition, applying the encryption and redaction functions to the data, as well as verifying that these functions were operating properly, was straightforward and easy to document, which is important from any compliance or regulatory perspective.
Dave Shackleford is the founder and principal consultant with Voodoo Security, a SANS analyst, instructor and course author, and a GIAC technical director. He has consulted with hundreds of organizations in the areas of security, regulatory compliance, and network architecture and engineering. He is a VMware vExpert and has extensive experience designing and configuring secure virtualized infrastructures. He has previously worked as chief security officer for Configuresoft and CTO for the Center for Internet Security. Dave is the author of the Sybex book, Virtualization Security. Recently, Dave co-authored the first published course on virtualization security for the SANS Institute. Dave currently serves on the board of directors at the SANS Technology Institute and helps lead the Atlanta chapter of the Cloud Security Alliance.

SANS would like to thank this paper’s sponsor:
Upcoming SANS App Sec Training