An Assessment of the Oracle Password Hashing Algorithm

Joshua Wright
SANS Institute
jwright@sans.org

Carlos Cid
Information Security Group
Royal Holloway, University of London
carlos.cid@rhul.ac.uk

18. Oct. 2005

Abstract

In this paper the authors examine the mechanism used in Oracle databases for protecting users' passwords. We review the algorithm used for generating password hashes, and show that the current mechanism presents a number of weaknesses, making it straightforward for an attacker with limited resources to recover a user's plaintext password from the hashed value. We also describe how to implement a password recovery tool using off-the-shelf software. We conclude by discussing some possible attack vectors and recommendations to mitigate this risk.

1 Introduction

Passwords are the most common form of user authentication in computer systems. To secure against unauthorized disclosure of users' credentials, passwords are usually protected by hashing them using a one-way algorithm. Password hashes (instead of cleartext passwords) are then stored in a password table. During log in, the user-provided password is hashed using the same algorithm and compared with the stored hash to authorize user access.

The general approach for protecting passwords is given in Morris and Thompson article [4] describing password security on UNIX systems. The main principles are:

- **Enforcement of Complex Passwords**: One of the weakest aspects of password-based authentication is the low entropy of commonly chosen passwords. The main attacks for recovering cleartext passwords from hash values consist of computation of all possible passwords up to a certain number of characters (exhaustive search attack), or perhaps a list of typically chosen passwords (dictionary attack). The computation is usually performed offline and the attacker simply compares the values on the password table with the precomputed list. Systems should therefore enforce password complexity rules, such as minimum length, requiring letters to be chosen from different sets of characters (e.g. lower-case, upper-case, digits, special characters), etc;
- **Use of Salted Passwords**: Each password hash is associated to a small, usually random value called salt. The salt does not need to be kept secret, and it is used together with the password to generate the password hash. While the use of salted passwords does not increase the task for recovering a particular password, a salt of sufficient length should preclude precomputed, offline dictionary attacks, as it becomes impractical to compute a large table of hashes corresponding to possible passwords and salt values in advance. The original UNIX system uses 12-bit salt values, while today it is recommended that the salt should be at least 64 bits [7]. Another feature of salted passwords is that it should become nearly impossible to infer a user's password based on another user's credentials and the corresponding hash values. Since users are likely to have different, random salt values, users with

same password should have completely different hash values;

- **Slow One-way Algorithm**: A slow one-way algorithm will not noticeably increase the cost of one operation (e.g. for the legitimate user when logging in), but it should substantially increase the task of mounting an exhaustive search attack. A common approach is to iterate the original one-way function many times. The original UNIX system one-way function encrypts a known string 25 times with DES using a key derived from the user's password (another feature of the UNIX mechanism is that the salt value actually modifies the DES algorithm itself, making it harder for an attacker to use dedicated DES hardware to mount an attack [4]). Given the current computer resources available, [7] recommends a minimum of 1000 iterations for constructing the hash algorithm.

2 Oracle Password Mechanism

Passwords for local user accounts in Oracle databases are stored as 8-byte password hashes in the Oracle SYS.USER$ table using an undocumented hashing algorithm. We have however identified several weaknesses associated with its password handling method, which significantly weakens the protection offered by password-based authentication mechanisms. These weaknesses include:

- Weak password salt selection;
- Lack of alphabetic case preservation;
- Weak hashing algorithm.

By exploiting these weaknesses, an adversary with limited resources can mount an attack that would reveal the plaintext password from the password hash for a known user. In the following sections we review the weaknesses associated with this mechanism, and identify potential attack vectors that can be exploited by an adversary.

Weak Salt Selection

Oracle password hashes use a non-conventional technique for salt selection by prepending the username to the password before calculating the hash. There are a number of weaknesses with this procedure. Firstly, it is quite possible to obtain information about a user password based solely on its hash value and the known credentials of another user. We can illustrate that by creating two accounts with similar usernames and passwords, as shown below:

```
SQL> create user oracle identified by password;

User created.

SQL> create user oracl identified by epassword;

User created.

SQL> select username, password from dba_users where username like 'ORACL%';

USERNAME                       PASSWORD
------------------------------ ------------------------------
ORACL                          427458AF9CC65444
ORACLE                         427458AF9CC65444

SQL>
```

By simply inspecting the password hash values from the dba_users table, an adversary would

have strong evidence of the relationship between both users' passwords.

A second weakness is the use of non-random salt values. Although the salt used can still reduce the effectiveness of a precomputed dictionary attack against a large password hash table, an attacker could still precompute a table of possible passwords using a common username (e.g. SYSTEM), and use it to attempt to recover the password for this particular user in many different systems.

Lack of Case Preservation

Another weakness in the Oracle password hashing mechanism is the lack of alphabetic case preservation. Before the password hash is calculated, the user's password is converted to all uppercase characters, regardless of the input case selection. We can observe this behavior by attempting to use mixed case in multiple passwords, viewing the hash values that are produced with each password.

```
SQL> alter user oracle identified by "PaSSwOrD";

User altered.

SQL> select username, password from dba_users where username = 'ORACLE';

USERNAME                       PASSWORD
------------------------------ ------------------------------
ORACLE                         427458AF9CC65444

SQL> alter user oracle identified by "password";

User altered.

SQL> select username, password from dba_users where username = 'ORACLE';

USERNAME                       PASSWORD
------------------------------ ------------------------------
ORACLE                         427458AF9CC65444

SQL> alter user oracle identified by "PASSWORD";

User altered.

SQL> select username, password from dba_users where username = 'ORACLE';

USERNAME                       PASSWORD
------------------------------ ------------------------------
ORACLE                         427458AF9CC65444

SQL>
```

From the output above we can see that the password hash remains consistent, regardless of the password selection. This behaviour represents a significant weakness in the password hashing algorithm, as it reduces the number of possible passwords, and therefore the overall password entropy (for example, if passwords are chosen using alphanumeric characters, then the mechanism allows 36^n possible passwords of length n, instead of the expected value of 62^n). Another effect is that many organizations that require a mix of uppercase and lowercase characters in passwords may have a false sense of security regarding the quality of database passwords.

Weak Hashing Algorithm

The one-way algorithm used to calculate password hashes is not openly documented by Oracle, but references on-line and in printed materials provide sufficient information to reproduce the algorithm.

A 1993 post on the comp.databases.oracle newsgroup describes the algorithm in detail, identifying an unknown fixed key as an input parameter [1]. This key value was later published in the book "Special Ops", providing sufficient information to reproduce the algorithm [2]. The algorithm can be described as follows:

1. Concatenate the username and the password to produce a plaintext string;
2. Convert the plaintext string to uppercase characters;
3. Convert the plaintext string to multi-byte storage format; ASCII characters have the high byte set to 0x00;
4. Encrypt the plaintext string (padded with 0s if necessary to the next even block length) using the DES algorithm in cipher block chaining (CBC) mode with a fixed key value of 0x0123456789ABCDEF;
5. Encrypt the plaintext string again with DES-CBC, but using the last block of the output of the previous step (ignoring parity bits) as the encryption key. The last block of the output is converted into a printable string to produce the password hash value.

Although we are not aware of any cryptographic weakness in this construction, we have however identified a significant computational weakness: it is extremelly inexpensive to compute a password hash using the one-way algorithm described above. If we assume that a typical username and password string has a length of approximately 12-16 characters, the algorithm above will perform between 6 and 8 DES encryptions to compute the hash value. Even considering that the two encryption operations use different keys (so the algorithm requires an extra key setup), this algorithm should still be very fast with current computational resources.

When performing an offline dictionary attack, the attacker can also take advantage of the prepended username prior to encryption by caching the encrypted username blocks. On average this will save one or two encryptions (corresponding to the first encryption of the username with the known fixed key), preventing the attacker from repeatedly computing this value when generating a large number of hashes. This may further reduce the computation cost for such attack by approximately 25%.

Using the sources cited above, the authors were able to reproduce the algorithm and verify the implementation by comparing calculated hashes with the observed hashes collected from publicly available resources including the Oracle Default Password list [6]. A simple implementation of this algorithm using the OpenSSL library will be included in the extended version of this article.

3 Attack Vectors

In order to mount an attack against the password hash, an adversary must obtain knowledge of the hash value itself. By default, access to the Oracle SYS.USER$ table used to store the password hashes is limited to administrative users who are members of the DBA role, as well as users who are granted the SELECT ANY DICTIONARY privilege. The SELECT ANY TABLE privilege also grants access to the SYS.USER$ table for Oracle database releases 8i and earlier, as well as later releases when the o7_dictionary_accessibility initialization parameter is set to true.

An attacker, however, has several other options to obtain the password hashes. That includes the ability to capture unencrypted Oracle transport network substrate (TNS) traffic or to exploit vulnerabilities in applications that permit SQL injection. With local access to the operating system of the database, an attacker can also examine the SYSTEM tablespace file (system01.dbf, by default) or the oraPW{SID} file with a tool such as the Unix strings utility to locate password hashes and associated usernames.

Once an attacker has recovered one or more usernames and password hashes, they can use the details of the hashing algorithm to mount an attack to recover user passwords.

Exhaustive Search and Dictionary Attacks

A common password hash attack is to use a dictionary file of words, computing the hash for each entry and comparing the hashed value to the hash of the password the attacker wishes to recover. A matching hash value indicates the dictionary word used to generate the hash is the user's password.

Given the weak Oracle password hashing mechanism, it is practical for an attacker with modern hardware to exhaust all possibilities for a limited password length to brute-force the password hash. Using a standard Intel Pentium 4 2.8 GHz workstation with OpenSSL 0.9.8-beta3, the authors achieved a rate of approximately 830,000 password hashes/second on a 32-byte data block. With a password length of 8 alphanumeric characters and a known username of 8 characters, an attacker could compute all possible possible passwords for a particular account in approximately 39.3 days using similar hardware, expecting to successfully recover the plaintext password in approximately 20 days. This is especially problematic for organizations with a password expiration duration that is shorter than 20 days, since it is likely an attacker will be able to produce the plaintext password before the account password is changed.

Rainbow Tables

In 2003, Philippe Oechslin published a paper illustrating a time/memory trade off that could be leveraged to reduce the amount of time needed to recover a plaintext password from the password hash [5]. The technique was later implemented by Zhu Shuanglei in a tool called RainbowCrack [8].

Using the time/memory trade off mechanism, an attacker precomputes password hashes and stores the results in one or more files (rainbow tables) that optimize the storage of passwords and the corresponding hashes. Once the rainbow tables are generated, an attacker can quickly recover the plaintext password from a password hash by searching the rainbow tables for a known hash value. This technique is especially advantageous to an attacker if they plan to recover multiple passwords, since the rainbow table precomputation only needs to be done once.

The use of rainbow tables is generally limited to password algorithms that do not use a salt. However, since the salt mechanism used by Oracle is the account username, an attacker can build rainbow tables for a fixed username. One username selection candidate would be the SYSTEM account, since this account exists on all Oracle databases, and would provide privileged access to the database if the hash can be reversed.

To verify these findings, the RainbowCrack tool was patched to include the Oracle hashing algorithm as a cipher selection, using the SYSTEM account as a fixed username. Using the limited character set of upper-case letters up to 8 characters in length (217,180,147,158 possible passwords) and a standard Pentium 2.8 GHz workstation, five rainbow tables were

generated over the course of three weeks. With this configuration, there is a probability of 98.1% that the password for the SYSTEM user will be revealed as plaintext.

An example of using the rainbow tables method to recover the plaintext password is shown below:

```
SQL> alter user SYSTEM identified by "lqucrcdj";

User altered.

SQL> select password from dba_users where username = 'SYSTEM';

PASSWORD
------------------------------
245C42547E8AE6E2

SQL>
```

For testing purposes, the hash for a randomly-selected password was supplied to the RainbowCrack rcrack tool, using the Oracle password hash rainbow tables, as shown below. The output from rcrack has been trimmed for space considerations.

```
$ ls oracle_alpha#1-8_[0-4]*
oracle_alpha#1-8_0_11300x20000000_alphaonly.rt
oracle_alpha#1-8_1_11300x20000000_alphaonly.rt
oracle_alpha#1-8_2_11300x20000000_alphaonly.rt
oracle_alpha#1-8_3_11300x20000000_alphaonly.rt
oracle_alpha#1-8_4_11300x20000000_alphaonly.rt
$ ./rcrack oracle_alpha#1-8_[0-4]* -h 245C42547E8AE6E2
oracle_alpha#1-8_0_11300x20000000_alphaonly.rt:
1579008 bytes read, disk access time: 0.00 s
verifying the file...
searching for 1 hash...
<trimmed output>

statistics
---------------------------------------------------------
plaintext found:          1 of 1 (100.00%)
total disk access time:   0.48 s
total cryptanalysis time: 247.78 s
total chain walk step:    63828051
total false alarm:        5611
total chain walk step due to false alarm: 21250996

result
---------------------------------------------------------
245c42547e8ae6e2  LQUCRCDJ  hex:4c5155435243444a
```

In little over four minutes, the password for the supplied hash value was recovered.

The corresponding patch to the RainbowCrack tool will be included in the extended version of this article.

4 Recommendations

Organizations who use Oracle databases can mitigate the weakness associated with the password hashing algorithm by enforcing strong password selection policy, and by enforcing the principle of least privilege. Specifically, organizations can implement the following steps to protect the confidentiality of password hashes:

- Use non-privileged users for web applications
- Restrict access to password hashes
- Audit SELECT statements on the DBA_USERS view
- Encrypt TNS traffic
- Enforce a minimum password length

These recommendations are detailed in more depth below.

Use non-privileged users for web applications

One technique for capturing password hashes is to exploit excessive permissions for database users configured to execute web-based application code that is susceptible to SQL injection attacks. When selecting a user account to provide access to a web application, ensure the account has only the minimum privileges granted to run the application. In no cases should a user that is a member of the DBA group be allowed to run web applications that are exposed to public or less-privileged users.

Restrict access to password hashes

While Oracle password hashes are stored in the SYS.USER$ table, several default views are created that can also be used to obtain password hash information from the database. The DBA_TAB_COLS view can be used to identify any database objects that include a column called PASSWORD, as shown below:

```
SQL> select owner, table_name from dba_tab_cols where column_name = 'PASSWORD';

OWNER                          TABLE_NAME
------------------------------ ------------------------------
SYS                            USER$
SYS                            LINK$
SYS                            USER_HISTORY$
SYS                            USER_DB_LINKS
SYS                            DBA_USERS
SYS                            EXU8ROL
SYS                            EXU8PHS
SYS                            EXU7ROL
SYS                            KU$_DBLINK_VIEW
SYS                            KU$_USER_VIEW
SYS                            KU$_ROLE_VIEW

OWNER                          TABLE_NAME
------------------------------ ------------------------------
WKSYS                          WK$_AUTHBASIC
WKSYS                          WK$AUTHBASIC
WKSYS                          WK$$AUTHBASIC

14 rows selected.

SQL>
```

Organizations can use a tool such as who_can_access.sql by Pete Finnigan to assist in identifying the users who have access to these objects, available at www.petefinnigan.co.uk/tools.htm. A sample report from the who_can_access.sql tool is shown below:

```
SQL> @who_can_access
who_can_access: Release 1.0.3.0.0 - Production on Thu May 26 15:03:02 2005
```

```
Copyright (c) 2004 PeteFinnigan.com Limited. All rights reserved.

NAME OF OBJECT TO CHECK       [USER_OBJECTS]: DBA_USERS
OWNER OF THE OBJECT TO CHECK          [USER]: SYS
OUTPUT METHOD Screen/File                [S]:
FILE NAME FOR OUTPUT              [priv.lst]:
OUTPUT DIRECTORY [DIRECTORY  or file (/tmp)]:
EXCLUDE CERTAIN USERS                    [N]:
USER TO SKIP                        [TEST%]:

Checking object => SYS.DBA_USERS
======================================================================


Object type is => VIEW (TAB)
               Privilege => SELECT is granted to =>
               User => CTXSYS (ADM = NO)
               Role => SELECT_CATALOG_ROLE (ADM = NO) which is granted to =>
                    User => SH (ADM = NO)
                    Role => DBA (ADM = YES) which is granted to =>
                           User => SYS (ADM = YES)
                           User => WKSYS (ADM = NO)
                           User => CTXSYS (ADM = NO)
                           User => SYSTEM (ADM = YES)
                    User => ODM (ADM = NO)
                    User => SYS (ADM = YES)
                    User => ODM_MTR (ADM = NO)
                    Role => OLAP_DBA (ADM = NO) which is granted to =>
                           Role => DBA (ADM = NO) which is granted to =>
                                  User => SYS (ADM = YES)
                                  User => WKSYS (ADM = NO)
                                  User => SYSTEM (ADM = YES)
                           User => SYS (ADM = YES)
                           User => OLAPSYS (ADM = NO)
                    Role => EXP_FULL_DATABASE (ADM = NO) which is granted to =>
                           Role => DBA (ADM = NO) which is granted to =>
                                  User => SYS (ADM = YES)
                                  User => WKSYS (ADM = NO)
                                  User => SYSTEM (ADM = YES)
                           User => SYS (ADM = YES)
                    Role => IMP_FULL_DATABASE (ADM = NO) which is granted to =>
                           Role => DBA (ADM = NO) which is granted to =>
                                  User => SYS (ADM = YES)
                                  User => WKSYS (ADM = NO)
                                  User => SYSTEM (ADM = YES)
                           User => SYS (ADM = YES)

PL/SQL procedure successfully completed.


For updates please visit http://www.petefinnigan.com/tools.htm
```

Administrators should also identify any users who are granted the SELECT ANY DICTIONARY privilege, as well as the SELECT ANY TABLE privilege. Note that the SELECT ANY TABLE privilege is only applicable if the o7_dictionary_accessibilty parameter is set to TRUE in the init.ora file.

Audit SELECT statements on the DBA_USERS view

Organizations may also wish to audit the DBA_USERS view that can be used to obtain password hashes for local database accounts. Once auditing is configured on the database, a

DBA or a user with the AUDIT ANY privilege can enable auditing on this view, as shown below:

```
SQL> audit SELECT on dba_users;

Audit succeeded.
```

Note that an attacker can refer to the SYS.USER$ base table directly obtaining password hashes without referencing the DBA_USERS view, evading the auditing operation. Attempting to audit the SYS.USER$ table directly generates an error:

```
SQL> audit select on sys.user$;
audit select on sys.user$
                    *
ERROR at line 1:
ORA-00701: object necessary for warmstarting database cannot be altered
```

The authors are not aware of a workaround for this issue.

Encrypt TNS traffic

Organizations should also consider using encryption options to protect the confidentiality of TNS traffic that may disclose sensitive information including password hashes. The ability to encrypt TNS traffic is included in the Oracle Advanced Security (OAS) option package for Oracle Enterprise Edition. Alternatively, organizations who are running Oracle Standard or Lite Edition, or cannot purchase the OAS software can use SSL tunneling features with tools such as OpenSSH (www.openssh.org) or stunnel (www.stunnel.org).

Enforce a Minimum Password Length

Database administrators can utilize the Oracle user profile and password verification function mechanism to ensure users select a password length that makes it improbable for an attacker to succeed when mounting a brute-force attack. An appropriate password length depends on the amount of resources available to the attacker that an organization wishes to defend against. Assuming an attacker has access to optimized DES-cracking hardware, an organization may need to enforce 12-character passwords and a password expiration duration of 60 days to mitigate a brute-force attack against the password hash.

A password verification procedure is a simple PL/SQL function that exists in the SYS schema, accepting the username, desired password and old password as input parameters that returns TRUE or an error depending on the characteristics of the password. A simple example of a password verification function is shown below:

```
SQL> CONNECT sys/change_on_install AS SYSDBA;
Connected.
SQL> CREATE OR REPLACE FUNCTION sys.password_verify
  2      (username varchar2, password varchar2, old_password varchar2)
  3      RETURN boolean IS
  4
  5  BEGIN
  6
  7      IF length(password) < 12 THEN
  8          raise_application_error(-20000, 'Password less than 12 characters');
  9      END IF;
 10
 11      RETURN(TRUE);
 12  END;
 13  /
```

```
Function created.
```

This simple function will raise an error when the input password length is less than 12 characters, otherwise it will return TRUE which permits the use of the desired password. Note that this function does not check for password complexity requirements, nor does it verify the password is not based on a dictionary word. A robust password verification script should include these checks to mitigate the risk of users selecting weak passwords.

After creating the password verification function, it must be associated with a profile. The example below demonstrates the use of the verification function after altering the default user profile:

```
SQL> ALTER PROFILE DEFAULT LIMIT PASSWORD_VERIFY_FUNCTION password_verify;

SQL> connect scott/tiger
Connected.
SQL> ALTER USER scott IDENTIFIED BY "abcdefghijk" REPLACE "tiger";
ALTER USER scott IDENTIFIED BY "abcdefghijk" REPLACE "tiger"
*
ERROR at line 1:
ORA-28003: password verification for the specified password failed
ORA-20000: Password less than 12 characters

SQL> ALTER USER scott IDENTIFIED BY "abcdefghijkl" REPLACE "tiger";

User altered.
```

5 Conclusion

In this paper we examine the mechanism used in Oracle databases for protecting users' passwords. We review the algorithm used for generating password hashes and describe an implementation of a simple password recovery tool. Previously limited to commercial tools, knowledge of the hashing algorithm accommodates the development of open-source auditing tools that DBA's can leverage to increase the security of their database. By auditing user password selection, an administrator can identify weak passwords and force users to change their passwords before they can be exploited by an attacker.

The current Oracle password mechanism presents a number of weaknesses, making it straightforward for an attacker to recover a user's plaintext password from the hashed value. Although there are a number of countermeasures that can be taken to protect users passwords, such as protecting the password table and enforcing complexity rules for passwords, the authors encourage Oracle customers to communicate their desire for a stronger password hashing mechanism through the appropriate channels.

References

[1] Bob Baldwin. (1993, July 9). "Oracle Password Encryption Algorithm?", Usenet Newsgroup comp.databases.oracle, URL: http://groups-beta.google.com/group/comp.databases.oracle/msg/83ae557a977fb6ed?hl=en

[2] Erik Pace Birkholz. "Special Ops, Host and Network Security for Microsoft UNIX, and Oracle". Rockland, MA. Syngress Press, 2003.

[3] Pete Finnigan. (2005, May). "PeteFinnigan.com Tools, who_can_access script", URL: http://www.petefinnigan.com/who_can_access.sql.

[4] Robert Morris and Ken Thompson. Password Security: A Case History. Communications of the ACM, 22(11):594 597, November 1979.

[5] Philippe Oechslin. (2003). "Making a Faster Cryptanalytic Time-Memory Trade-Off", URL: http://lasecwww.epfl.ch/pub/lasec/doc/Oech03.pdf.

[6] Oracle Default Password List. (2005, May 19). URL: http://www.petefinnigan.com/default/default_password_list.htm

[7] RSA Laboratories. PKCS#5: Password-Based Cryptography Standard. Version 2.0 (March 1999). http://www.rsasecurity.com/rsalabs/node.asp?id=2127

[8] Zhu Shuanglei. (2005, April 19). "RainbowCrack", URL: http://www.antsight.com/zsl/rainbowcrack/.