PFCL

PETEFINNIGAN.COM LIMITED

Oracle Database Password Security

An Appreciation

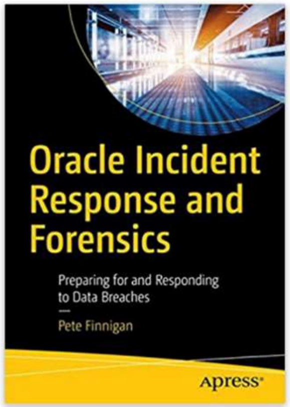
Legal Notice

Data Security in ERP and Other Large Business Systems

Published by
PeteFinnigan.com Limited
Tower Court
3 Oakdale Road
York
England, YO30 4XL

Limit of Liability / Disclaimer of warranty. This information contained in this course and this material is distributed on an "as-is" basis without warranty. Whilst every precaution has been taken in the preparation of this material, neither the author nor the publisher shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions or guidance contained within this course.

TradeMarks. Many of the designations used by manufacturers and resellers to distinguish their products are claimed as trademarks. Linux is a trademark of Linus Torvalds, Oracle is a trademark of Oracle Corporation. All other trademarks are the property of their respective owners. All other product names or services identified throughout the course material are used in an editorial fashion only and for the benefit of such companies with no intention of infringement of the trademark. No such use, or the use of any trade name, is intended to convey endorsement or other affiliation with this course.



Oracle Incident Response and Forensics

Preparing for and Responding to Data Breaches

—

Pete Finnigan

Apress

Pete Finnigan – Background, Who Am I?

- Oracle Security specialist and researcher
- CEO and founder of PeteFinnigan.com Limited in February 2003
- Writer of the longest running Oracle security blog
- Author of the Oracle Security step-by-step guide and "Oracle Expert Practices", "Oracle Incident Response and Forensics" books
- Oracle ACE for security
- Member of the OakTable
- Speaker at various conferences
  - UKOUG, PSOUG, BlackHat, more..
- Published many times, see
  - http://www.petefinnigan.com for links
- Influenced industry standards
  - And governments

Agenda

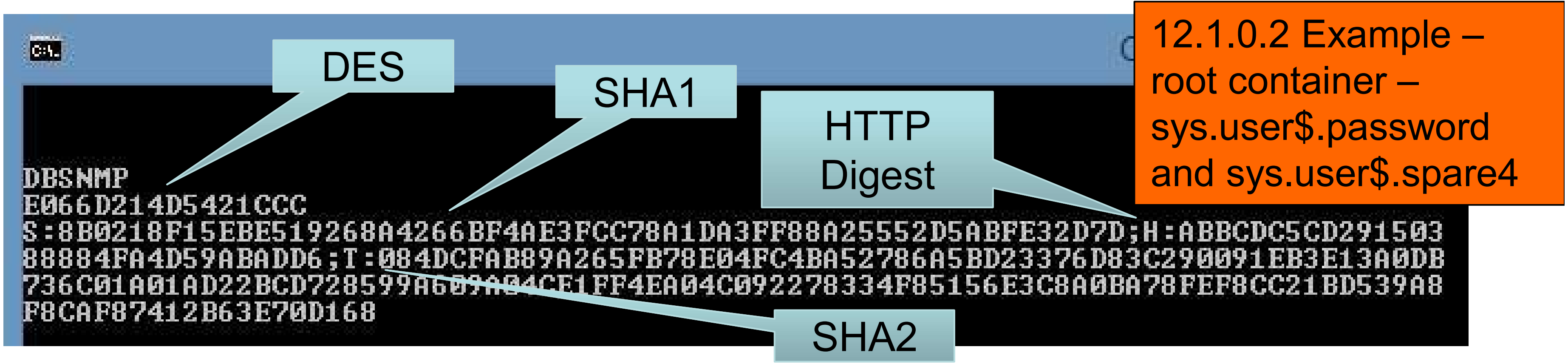
- Define the problem
- The password algorithms used
- Cracking passwords
- Security of passwords
- Password design
- Profile design
- Password safes

The Problem Space – High Level

- Attacking a database needs either:
  - A direct database connection (**We are focusing on this one!**)
  - OR Exploiting an application via SQL Injection or similar
- For a direct database connection we need:
  - A direct “pipe” to the database – Open or controlled routing
  - Network details – IP/host, port, SID/Service Name
  - A username and password
  - **Often: we can locate almost all of the above except the password – (tnsnames, guess usernames…..)**
  - BUT; often (in real sites/systems) we can also **guess or find passwords**

The Password Algorithms in The Database

- Starting in Oracle 6 there is one core password algorithm (DES)
- Starting in 11gR1 there are 2 core algorithms (DES, SHA1)
- Starting in 12.1.0.1 there are 3 core database algorithms (DES, SHA1, HTTP Digest)
- Starting in 12.1.0.2 there are 4 core password algorithms (DES, SHA1, SHA512, HTTP Digest)
- Along the way we have also had others such as ftp via XDB
- Unless you control it all algorithms exist – more shortly on this



```
DBSNMP
E066D214D5421CCC
S:8B0218F15EBE519268A4266BF4AE3FCC78A1DA3FF88A25552D5ABFE32D7D;H:ABBCDC5CD291503
88884FA4D59ABADD6;T:084DCFAB89A265FB78E04FC4BA52786A5BD23376D83C290091EB3E13A0DB
736C01A01AD22BCD728599A607A04CE1FF4EA04C092278334F85156E3C8A0BA78FEF8CC21BD539A8
F8CAF87412B63E70D168
```

12.1.0.2 Example – root container – sys.user$.password and sys.user$.spare4

Strengths / weaknesses – No real attack except brute force but key is too short now

DES

- Used from Oracle 6 through Oracle 10gR2
  - Actually still enabled in 11gR1 to 12.1.0.2
- Designed by Bob Baldwin – designer of NT and VMS algorithms - https://groups.google.com/forum/#!msg/comp.databases.oracle/F0uSWBy9e_Q/7bZ_I3pVroMJ - posted to usenet in 1993
  - **Note that the details posted are not 100% correct**
- Algorithm:
  - Concatenate user|password => Unicode the string => encrypt with DES using key 0x0123456789abcdef => encrypt first block => xor next block with result => take the last IV as a new KEY and repeat
  - The password hash generated is then not reversible

SHA1

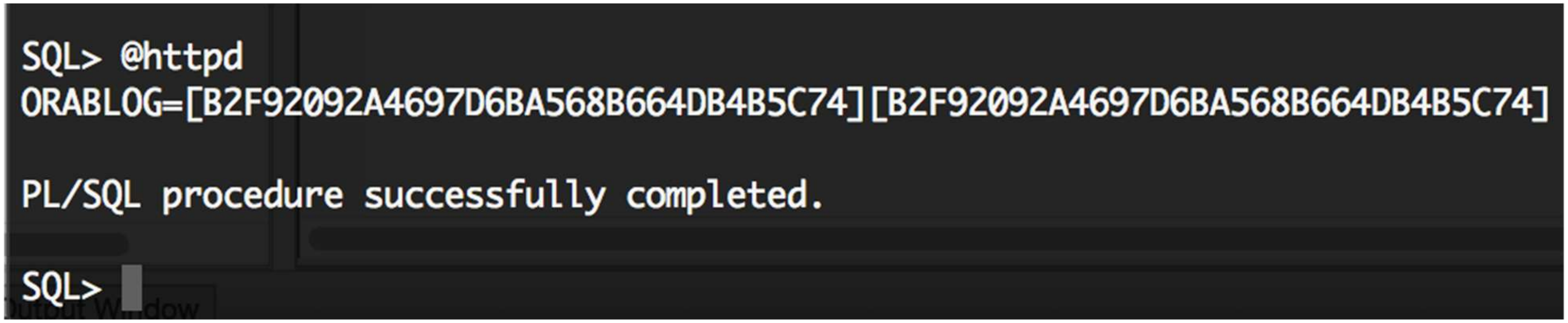
- Used in 11gR1 through 11.2.0.4
  - Actually still available in 12.1.0.2
- Added case sensitive passwords to the database for first time
- As a result longer key space by default
- Password only is hashed, not username and password (in DES the username is the salt)
- Salt is generated by the database on password create/change
  - Salt is passed by SQL*Net to the client
  - Salt is stored in SYS.USER$.SPARE4
  - Salt is to prevent same hash generated for same password
- Fast algorithm – not good for avoiding cracking..☹
- SHA1 is broken - https://www.schneier.com/blog/archives/2005/02/sha1_broken.html

SHA2

- Only added since 12.1.0.2 – SHA2 also added to DBMS_CRYPTO
- Hinted at in 12.1.0.1 – see comments in code in .bsq file for user$ table creation for instance
- Password hash stored as T: in SYS.USER$.SPARE4 column
- Combination of SHA2 – (SHA512) and PBKDF2 algorithms
  - PBKDF2 is done in the client
  - SHA2 is completed in the server
  - As with SHA1 the password hash and salt are stored in SYS.USER$.SPARE4
- Strengths / Weaknesses
  - Much slower to crack due to PBKDF2 part so much better than SHA1 or DES for slowing cracking
  - Documented as demo already on-line back in June; known longer

HTTP Digest

- Added in 12.1.0.1 to all database accounts
- Strange addition; SHA2 added as much stronger algorithm but HTTP Digest added just before
- MD5 is of course a predecessor to SHA and SHA1 and must faster to execute than SHA2
- Same hash always generated for same password
- Can crack the password in PL/SQL:

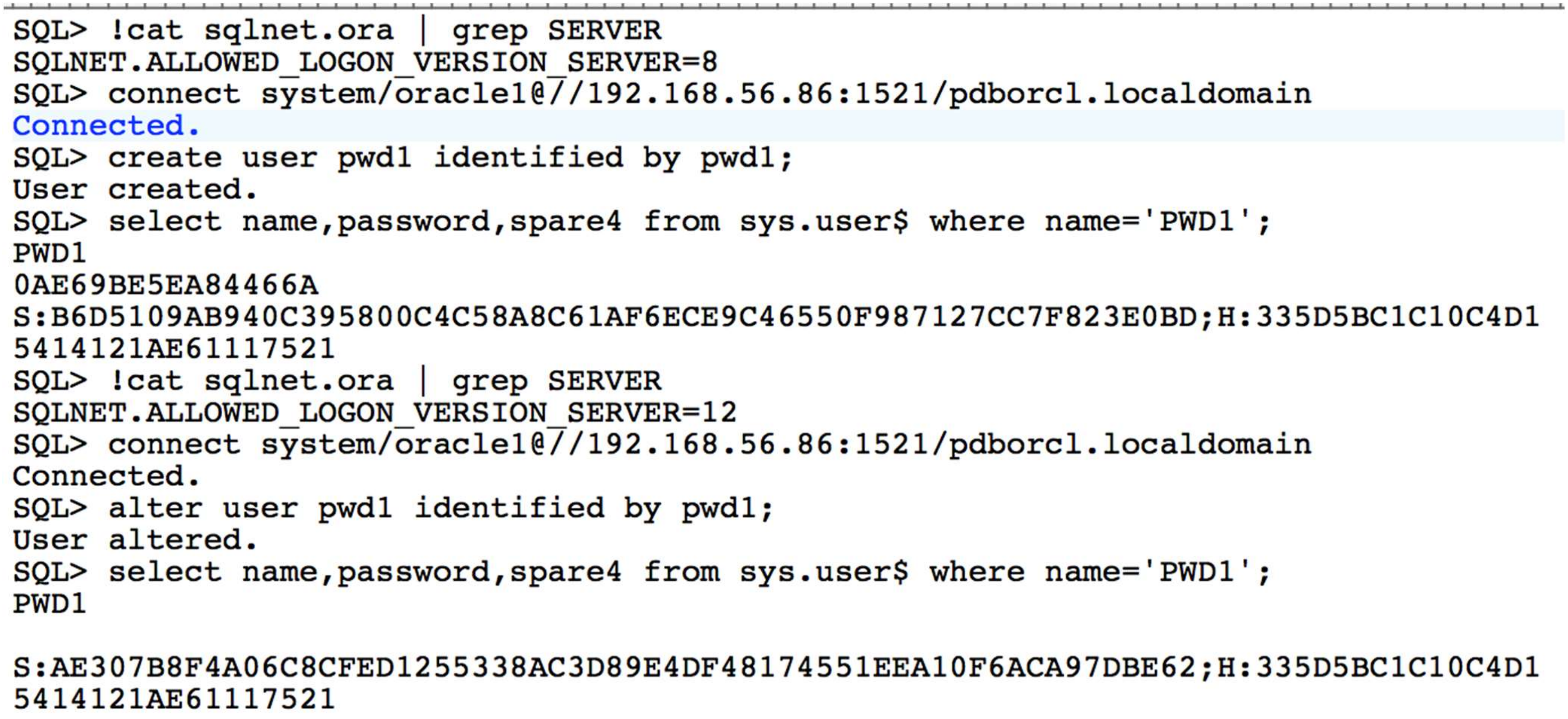


```
SQL> @httpd
ORABLOG=[B2F92092A4697D6BA568B664DB4B5C74][B2F92092A4697D6BA568B664DB4B5C74]

PL/SQL procedure successfully completed.

SQL> 
```

Disable Weaker Algorithms



```
SQL> !cat sqlnet.ora | grep SERVER
SQLNET.ALLOWED_LOGON_VERSION_SERVER=8
SQL> connect system/oracle1@//192.168.56.86:1521/pdborcl.localdomain
Connected.
SQL> create user pwd1 identified by pwd1;
User created.
SQL> select name,password,spare4 from sys.user$ where name='PWD1';
PWD1
0AE69BE5EA84466A
S:B6D5109AB940C395800C4C58A8C61AF6ECE9C46550F987127CC7F823E0BD;H:335D5BC1C10C4D1
5414121AE61117521
SQL> !cat sqlnet.ora | grep SERVER
SQLNET.ALLOWED_LOGON_VERSION_SERVER=12
SQL> connect system/oracle1@//192.168.56.86:1521/pdborcl.localdomain
Connected.
SQL> alter user pwd1 identified by pwd1;
User altered.
SQL> select name,password,spare4 from sys.user$ where name='PWD1';
PWD1

S:AE307B8F4A06C8CFED1255338AC3D89E4DF48174551EEA10F6ACA97DBE62;H:335D5BC1C10C4D1
5414121AE61117521
```

Demo PL/SQL based cracker on 12c

Cracking Passwords

- Why do we need to crack passwords
  - We need to test the strength of our passwords but unless they are weak we cannot fully do this
  - A password cracker will take too long to test a 12 or more character password
  - If we have 15 character passwords we cannot prove this without access to government level hardware and budget
  - We must assume others can crack our passwords so we must make some efforts to test our own
- Cracking types and more
  - Connect brute force
  - Default passwords
  - Dictionary attacks
  - Brute force
  - Permutes
  - Top 500, 1,000, 10,000 passwords
  - Dictionary languages – Switzerland!
- A simple PL/SQL based cracker can give a good overview of current password security

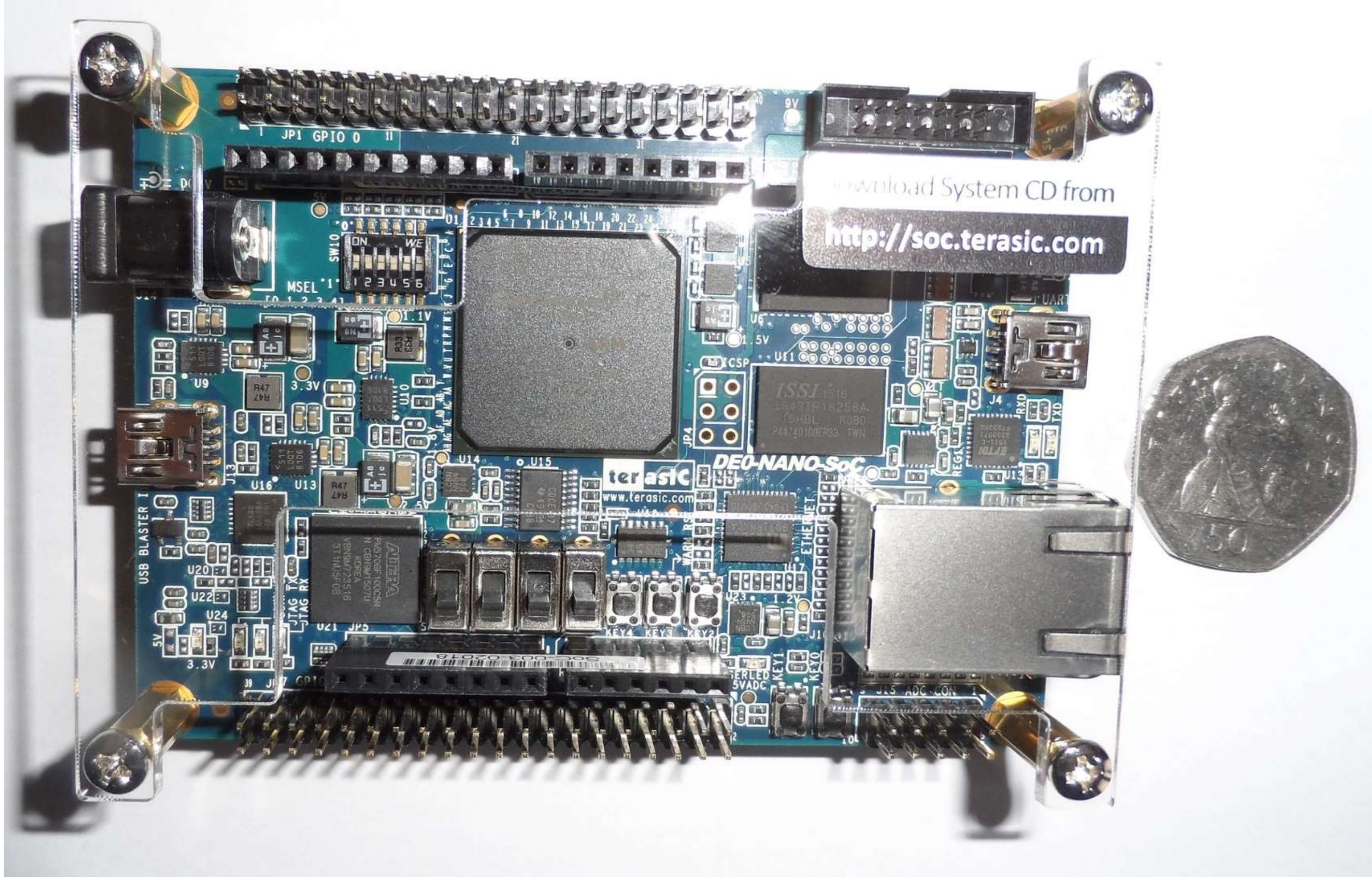
Cracking More…

- C based crackers – run faster than PL/SQL so can test more passwords
  - Orabf – 0rm
  - Woraauthbf – Laszlo Toth
  - Checkpwd – Alex Kornbrust
  - Many more such as JTR
- GPU crackers
  - - http://marcellmajor.com/frame_cudadbcracker.html from Marcell Major - 200 Million hashes a second
  - IGHASGPU - 790 million hashes a second SHA1 cracker - >52 character space would null the speed increase compared to DES
  - These can also be used on a limited character set – so SHA1 and 26 characters
- Online crackers exists for some algorithms – such as md5 and DES so can be used for single hashes
- Dennis Yurichev cracker on next slide was on-line BUT NO LONGER

Cracking – Hardware Crackers

- Hardware crackers – ASIC, FPGA, GPU (Really Software?)
- http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/ - Not Oracle but good example of GPU hardware cluster – **350 Billion guesses a second!**
  - SHA512 only 364K / sec on same hardware (massively slower)
  - Approx cost 25 * £135 + 5 * £600 = £6,375 (my guess on price) + time / dev!!
- http://yurichev.com/ops_FPGA.html - 65 - 85 Million hashes a second – just built and set live, no analysis, no serious tuning
  - Because its an FPGA it can be duplicated on other FPGA hardware
  - How far can someone go with reasonable costs
  - Denis used Stratix II 60k LE; Available: XESS Xula 24K (DIP40 pkg) or De0-Nano-Soc 40K (dual core, 900mhz Arm Cortex A9 running Linux, gig Ethernet) – (Possibly 77K LE if Arm Removed) – all for £67 (chip only £95; hmmm) - £6375/67 = 95 * 60 = 5700 * 1.4M = 7.9 Billion Hashes a second – for the same small money – would get more cards for same price – possibly 1B/hps with tuning for approx £7-800 GBP
- GPU is better value ? we are comparing Windows NT and DES
- ASIC probably would be better for speed, not cost? – would need more work but custom design should always be faster

Deo-Nano-Soc FPGA

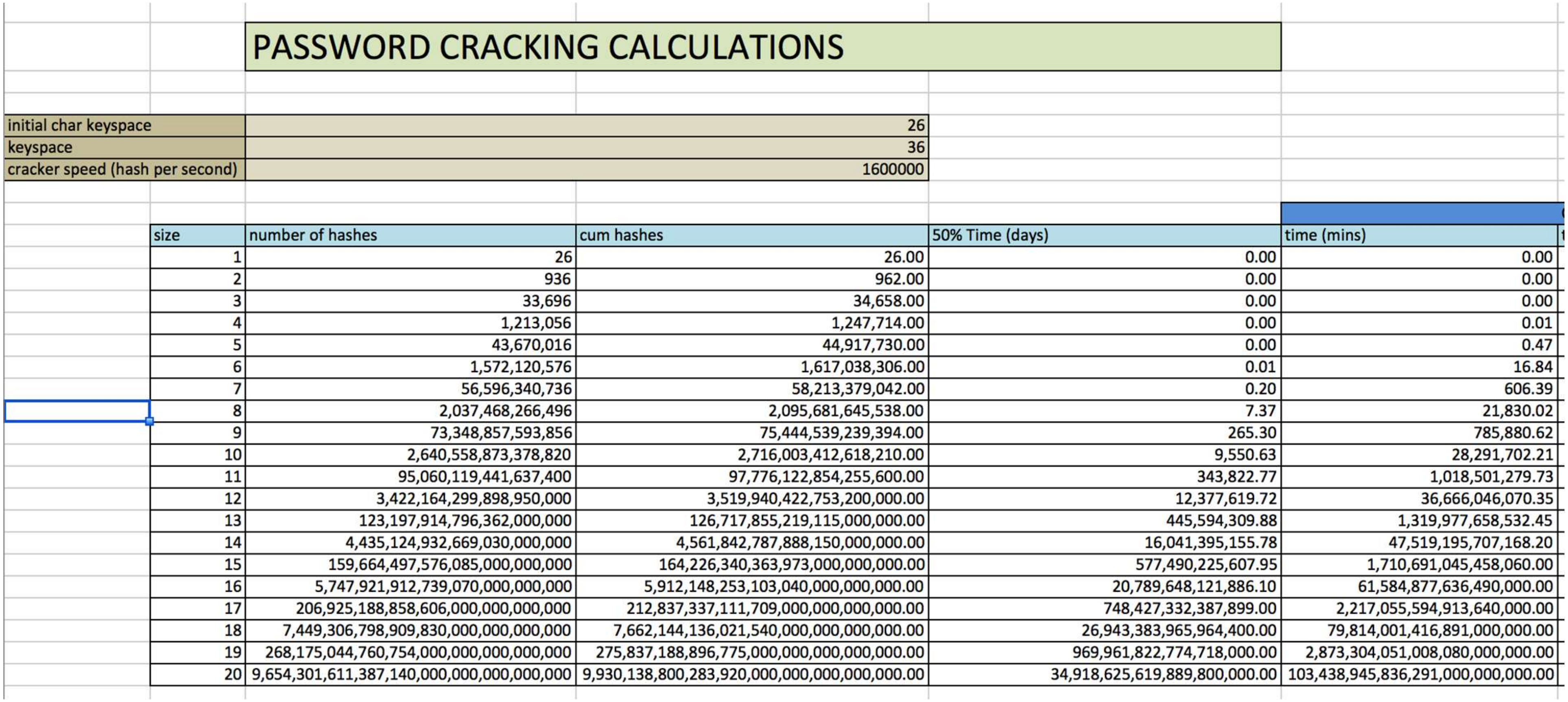


Password Design

- Password design must be scientific
  - We cannot simply set a length based on no other factors such as lifetime and complexity of the password
  - If we must set a length then we have to design a lifetime and complexity rules
- We also must consider users ability to bypass the rules
- We must ensure that the business does not bypass the rules for some passwords – (often schemas and DBA)
- We must understand how someone could find, steal, crack, subvert passwords and use that knowledge to design strong passwords

Demonstrate different cracker speeds and also keyspaces

Password Cracking Calculations



PASSWORD CRACKING CALCULATIONS

initial char keyspace	26
keyspace	36
cracker speed (hash per second)	1600000

size	number of hashes	cum hashes	50% Time (days)	time (mins)
1	26	26.00	0.00	0.00
2	936	962.00	0.00	0.00
3	33,696	34,658.00	0.00	0.00
4	1,213,056	1,247,714.00	0.00	0.01
5	43,670,016	44,917,730.00	0.00	0.47
6	1,572,120,576	1,617,038,306.00	0.01	16.84
7	56,596,340,736	58,213,379,042.00	0.20	606.39
8	2,037,468,266,496	2,095,681,645,538.00	7.37	21,830.02
9	73,348,857,593,856	75,444,539,239,394.00	265.30	785,880.62
10	2,640,558,873,378,820	2,716,003,412,618,210.00	9,550.63	28,291,702.21
11	95,060,119,441,637,400	97,776,122,854,255,600.00	343,822.77	1,018,501,279.73
12	3,422,164,299,898,950,000	3,519,940,422,753,200,000.00	12,377,619.72	36,666,046,070.35
13	123,197,914,796,362,000,000	126,717,855,219,115,000,000.00	445,594,309.88	1,319,977,658,532.45
14	4,435,124,932,669,030,000,000	4,561,842,787,888,150,000,000.00	16,041,395,155.78	47,519,195,707,168.20
15	159,664,497,576,085,000,000,000	164,226,340,363,973,000,000,000.00	577,490,225,607.95	1,710,691,045,458,060.00
16	5,747,921,912,739,070,000,000,000	5,912,148,253,103,040,000,000,000.00	20,789,648,121,886.10	61,584,877,636,490,000.00
17	206,925,188,858,606,000,000,000,000	212,837,337,111,709,000,000,000,000.00	748,427,332,387,899.00	2,217,055,594,913,640,000.00
18	7,449,306,798,909,830,000,000,000,000	7,662,144,136,021,540,000,000,000,000.00	26,943,383,965,964,400.00	79,814,001,416,891,000,000.00
19	268,175,044,760,754,000,000,000,000,000	275,837,188,896,775,000,000,000,000,000.00	969,961,822,774,718,000.00	2,873,304,051,008,080,000,000.00
20	9,654,301,611,387,140,000,000,000,000,000	9,930,138,800,283,920,000,000,000,000,000.00	34,918,625,619,889,800,000.00	103,438,945,836,291,000,000,000.00

Crackers Can Affect Password Choice

```
SQL> create user aaaa identified by aaaaa;
User created.
SQL> create user zzzz identified by zzzzz;
User created.
SQL> select name,password from sys.user$ where name in ('AAAA','ZZZZ');
NAME                           PASSWORD
------------------------------ ------------------------------
AAAA                           00F5652AE69FE700
ZZZZ                           7AAED8BB9D1B19F3
C:\>woraauthbf.exe -p aaaa.lis -t hash -m 5 -c alpha
...
Password found: AAAA:AAAAA:A:A
Elapsed time: 1s
Checked passwords: 874317
Password / Second: 874317
C:\>woraauthbf.exe -p zzzz.lis -t hash -m 5 -c alpha
...
Password found: ZZZZ:ZZZZZ:A:A
Elapsed time: 7s
Checked passwords: 12359760
Password / Second: 1765680
```

Profile Design

- Resource fields (e.g. sessions_per_user) need resource_limit to be turned on
- Fields reuse_time and reuse_max should not be used
  - In combination they do not work as you imagine
  - Better to never allow passwords to be re-used
- The field grace_time is confusing and artificially extends the life time
- The life time must be designed in combination with complexity
- Complexity function must exist to enforce the password
- Lock time must be designed based on use of the account
- Ensure global parameters – case, failed logins also match design

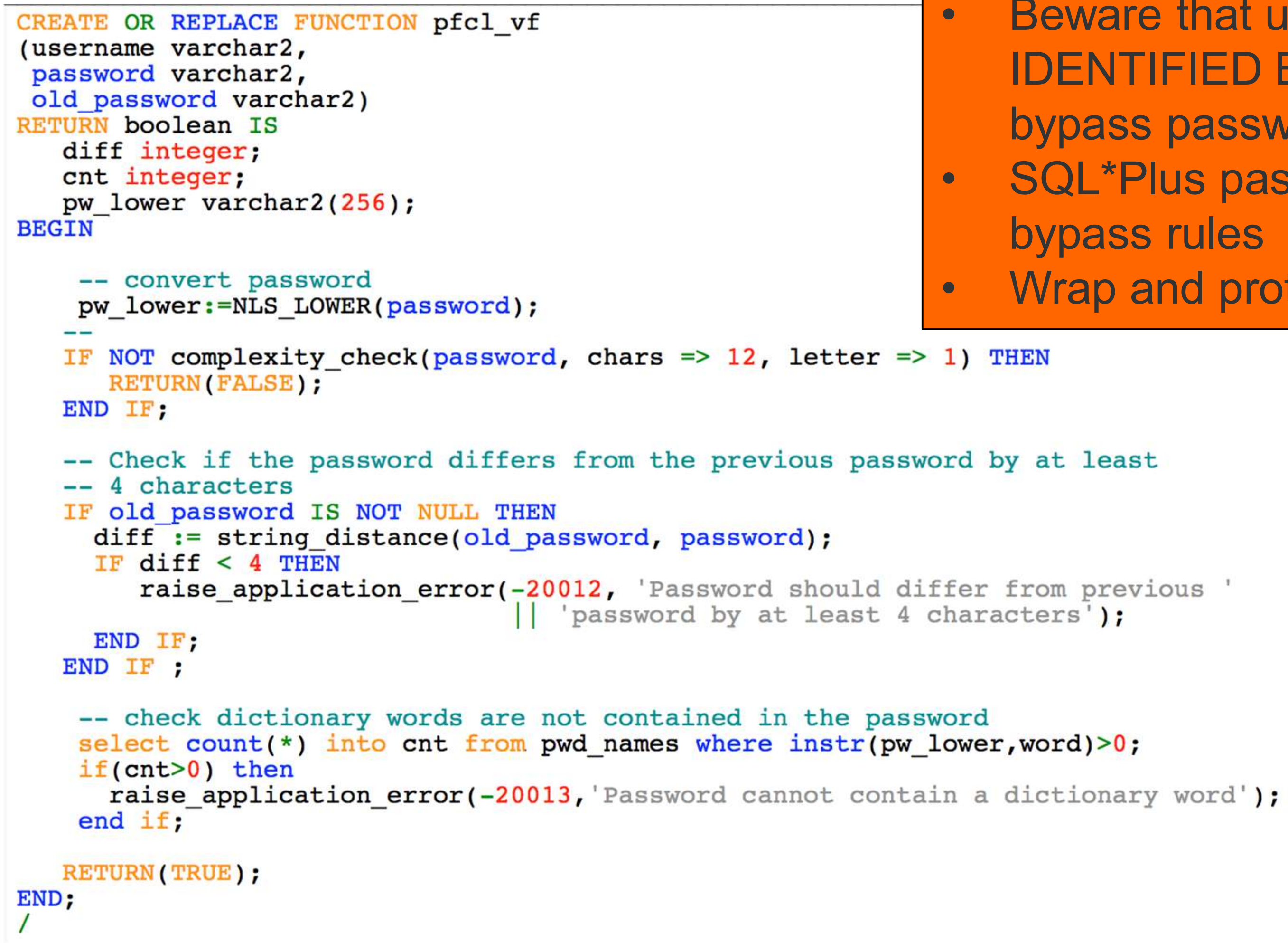
These are my examples, design your own..☺

Profile Design (2)

	Schema	Built-in	Admin	Power	Default
Failed Login	1	1	3	5	1
Reuse Time	INF	INF	INF	INF	INF
Sessions	1	1	3	2	1
Lock Time	10 Days	10 Days	0.5 Days	1 Day	10 Days
Max Reuse	Never	Never	Never	Never	Never
Grace Time	0	0	1 Day	3 Days	0
LifeTime	Calc	Calc	Calc	Calc	Calc

Password Verify Function

- Use the 12c Core functions in utlpwdmg.sql
- Write your own simple “frame”
- Adding a verify function forces use of “replace” syntax
- Beware that use of ALTER USER IDENTIFIED BY VALUES can bypass password verification
- SQL*Plus password also can bypass rules
- Wrap and protect function



```
CREATE OR REPLACE FUNCTION pfcl_vf
(username varchar2,
 password varchar2,
 old_password varchar2)
RETURN boolean IS
   diff integer;
   cnt integer;
   pw_lower varchar2(256);
BEGIN

    -- convert password
    pw_lower:=NLS_LOWER(password);
   --
   IF NOT complexity_check(password, chars => 12, letter => 1) THEN
      RETURN(FALSE);
   END IF;

   -- Check if the password differs from the previous password by at least
   -- 4 characters
   IF old_password IS NOT NULL THEN
     diff := string_distance(old_password, password);
     IF diff < 4 THEN
        raise_application_error(-20012, 'Password should differ from previous '
                                || 'password by at least 4 characters');
     END IF;
   END IF ;

    -- check dictionary words are not contained in the password
    select count(*) into cnt from pwd_names where instr(pw_lower,word)>0;
    if(cnt>0) then
      raise_application_error(-20013,'Password cannot contain a dictionary word');
    end if;

   RETURN(TRUE);
END;
/
```

PFCL
PETEFINNIGAN.COM LIMITED

Conclusions

- Design strong passwords
- Ensure hashes cannot be read
- Ensure strong passwords are properly enforced
- Ensure everyone is involved – e.g. no gaps

Questions?

Any Final Questions?

PFCL
PETEFINNIGAN.COM LIMITED

Oracle Database Password Security

An Appreciation