Akamai Web Application Firewall Bypass Journey: Exploiting “Google BigQuery” SQL Injection Vulnerability

Akamai Web Application Firewall Bypass Journey: Exploiting “Google BigQuery” SQL Injection Vulnerability - Duc Nguyen The, hackemall.live.

- Published: date not stated
- Original: <https://hackemall.live/index.php/2020/03/31/akamai-web-application-firewall-bypass-journey-exploiting-google-bigquery-sql-injection-vulnerability/>
- Preserved from: https://hackemall.live/index.php/2020/03/31/akamai-web-application-firewall-bypass-journey-exploiting-google-bigquery-sql-injection-vulnerability/ (stored) on 2026-08-11
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Akamai Web Application Firewall Bypass Journey: Exploiting “Google BigQuery” SQL Injection Vulnerability | Hack 'Em All

akamai, Bug bounty, WAF bypass, Web, Writeup

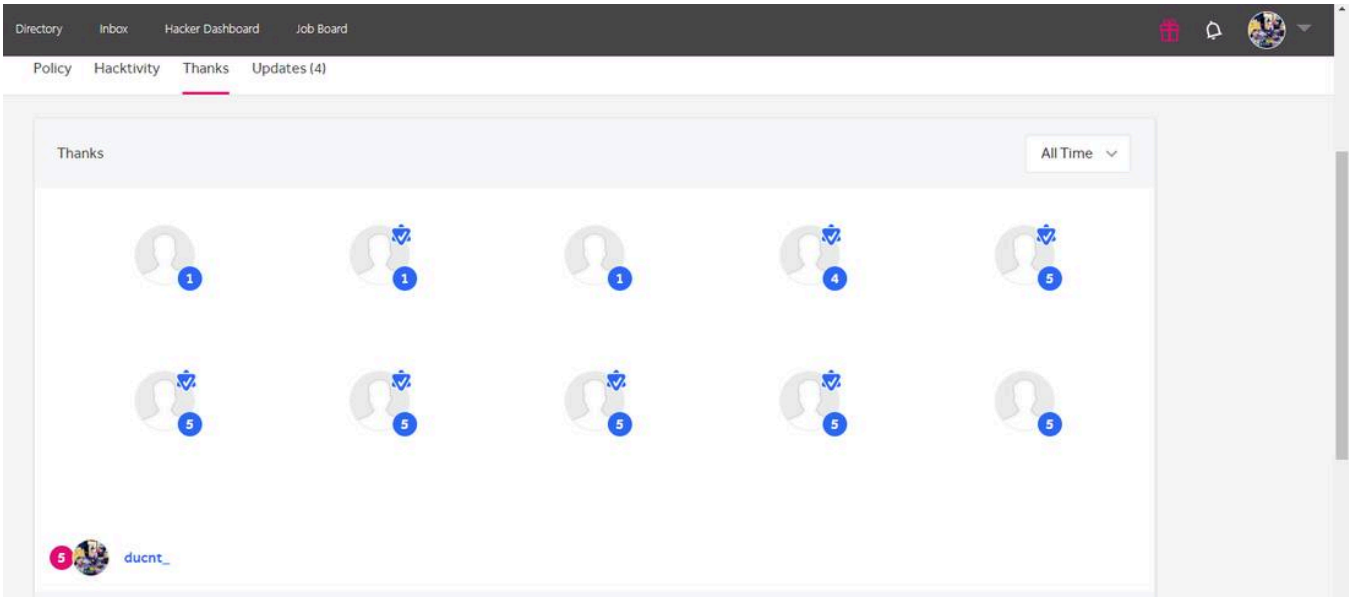
Akamai Web Application Firewall Bypass Journey: Exploiting “Google BigQuery” SQL Injection Vulnerability

By Duc Nguyen The, March 31, 2020

Hi guys, long time no write. As some previous articles in my blog only focus on CTF writeups, so in this time and maybe the next time, I want to write another topic about my research also doing bug bounty hunter. So as the topic name above, in this time I will write about my experience when bypass the popular web application firewall (WAF) of akamai technologies company aka. Kona WAF and exploit a SQL injection vulnerability.

0x01: The Stumble Upon.

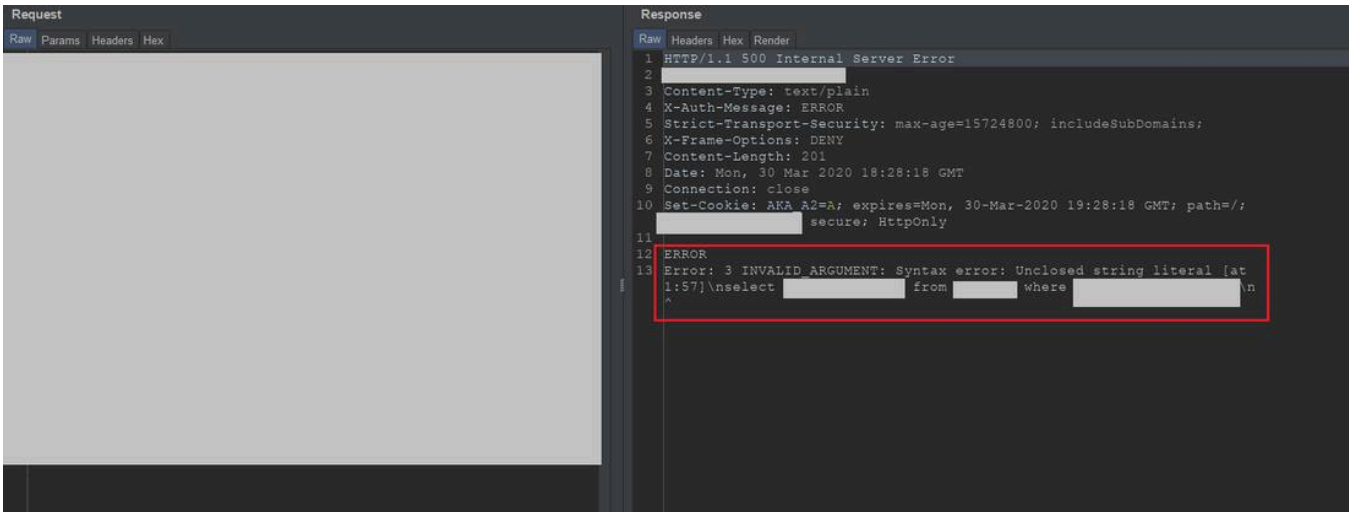
Last weekend, I was invited to a private program on hackerone, and yes for the private info as usually, I will call that program is: **0x1337.space** program. Actually, I quickly navigate to the scope section also the thanks page for looking the basic info. The program has a large scope: ***. 0x1337.space** and launched for a long time ago. At the thanks page:



Thanks page of program. So ***clear verify*** *much wow :doge:*

So many "big" bros with average reputation from 2000 . When you was invited to a private program, you think "oh my time, I will find more bug, more bounty, that ez than public program ". You should rethink about that.

And yes as usually, when wating the report from medusa-v1.0 (my bug bounty hunter tool), I poked around the main domain (homepage). The page opened with so much function, endpoint ...etc... After working around with the burpsuite requests / responses, I noticed that there is a problem at a POST parameter. So in my article, I will call it with the name is: **"location"**. Basically, the server don't sanitize user input in the location parameter, so the exception appear as below image.



Exception

The crafted request I used simply:

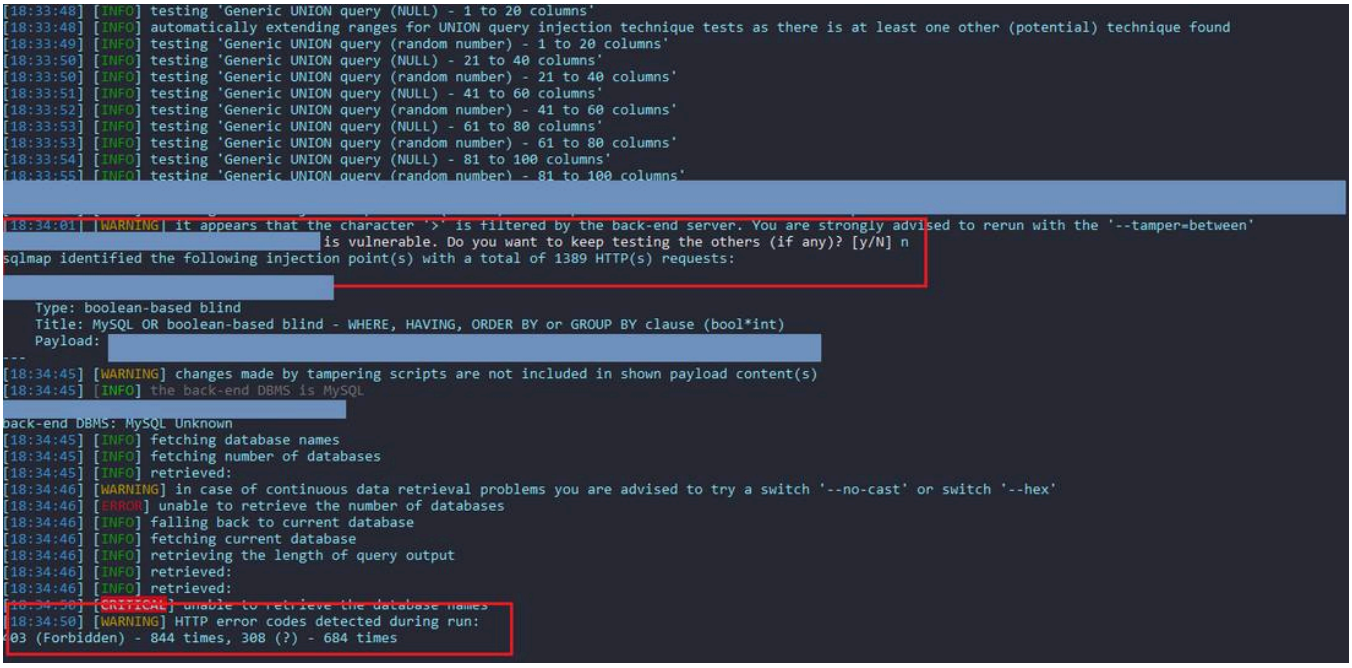
```
POST /vulnerable-endpoint/ HTTP/1.1
Host: 0x1337.space
...

**location=abcd1337"**
```

As you can see at above image, the server response SQL query exception with detail table name also the columns name. The info similar that:

```
**select AAA,BBB,CCC,DDD from EEE where location=\"abcd1337\"\"\n **
```

My friend named me with the nickname ducnt `SQL` and with all my experiences, I totally definitely this is a SQL injection vulnerability . So, I quickly setup an automatic exploit with this parameter, made a coffee cup, draft a SQL injection report to the vendor, waiting the information extracted from automation tool and fill in, submit the report and get the bounty. Ez life, ez money.

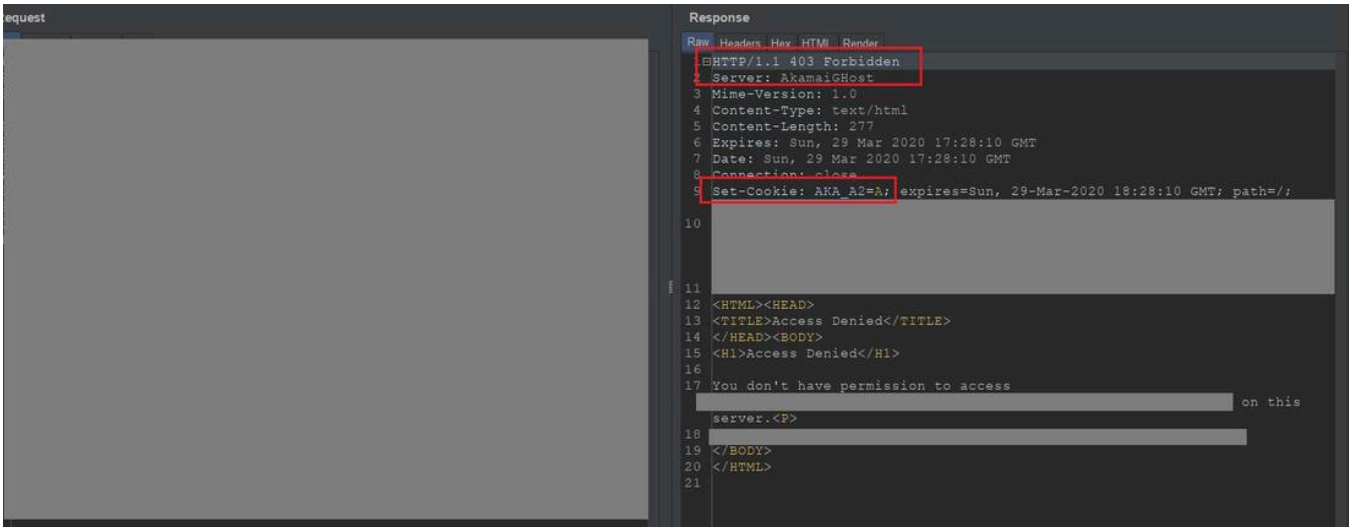
But but ...



WAFWTF … Not easy like that, after digging more about the 403 status code when exploit SQL injection vulnerability that prevent extract the info from the database. The payload I used is:

```
**location=abcd1337" union select 1-- **
```

and I stumble upon with:



WAFWTF …

So, I think, that is the reason why a vendor with a large scope launched for a long time ago also tested with so many “big” bros but this vulnerability still exist. However, as usually: challenges accepted.

0x02: Kona WAF Bypass && Exploit A Blind SQL Injection Vulnerability.

Detect the Kona WAF's behavior.

First of all, I tried every bypass method was public on the internet for bypass Kona WAF but no one success. Below are some payload I tried.

```
**' and sleep/**f**/(10) LIKE '3--
-abc)/**/OR/**/MID(CURRENT_USER,1,1)/**/LIKE/**/'a'/**/--**
```

I spent 1 day and try everything to find the way for bypass WAF but got nothing.

In the next day, when I revisited it and manual exploit SQL injection vulnerability in a filter / block / WAF ... context, I try to detect which keyword, function was blocked also the behavior of them. From this I will find the way that can bypass, suitable with the context and find the payload that can extract the information from database. To sum up the WAF's behavior:

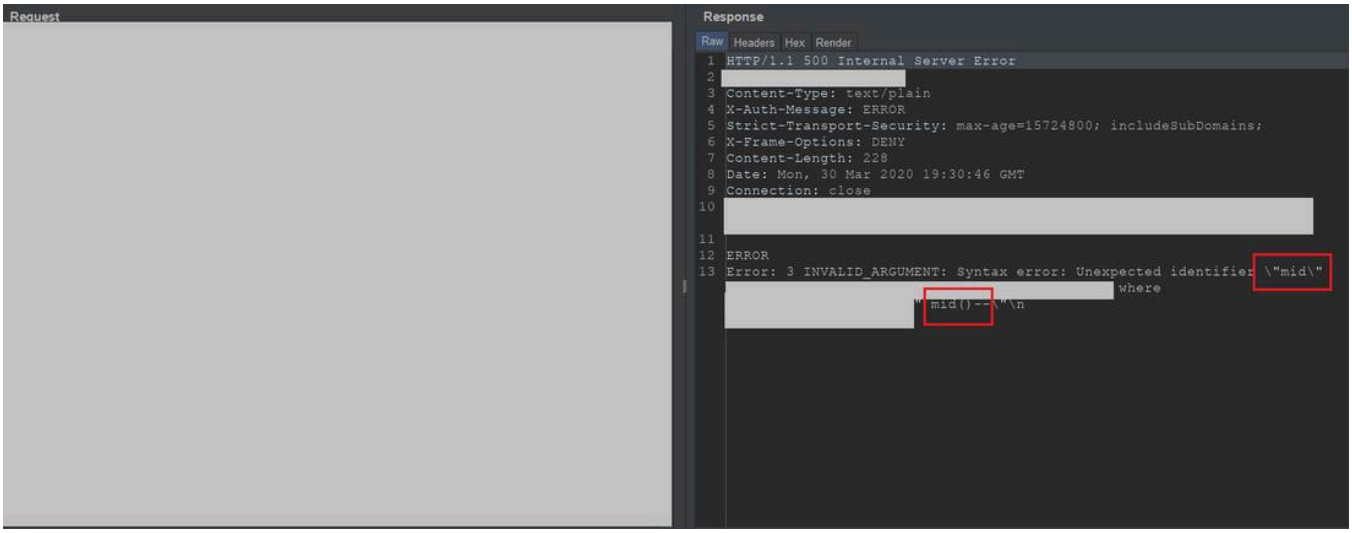
Block keyword: **union**.

Block some function:** sleep(), count()** ...

Block some global variable: **@@version** ...

Block the combine of the query: **select “ducnt” is okay**, but *select "ducnt" from* was blocked.

No **mid()**, **ascii()** function !?!, that really weird with a normal database like SQL or MySQL.



Unidentified function error

When google bigquery meet the Kona WAF.

So with the info I collected above, I reviewed again the exception info.

> “INVALID_ARGUMENT: Syntax error: Unexpected identifier”

Sound familiar, from this info and **“500 Internal Server Error”** status code from server also the info of the WAF’s behavior above. I going to a conclusion that the server using google bigquery, not MySQL also SQL Server. For more information about the exception, you can find at: **https://cloud.google.com/spanner/docs/reference/rest/v1/Code**

**Kona WAF bypass: Exploiting a “Google BigQuery” blind sql injection vulnerability. **

0: Preparation.

As I mentioned above, the info of database exception similar that:

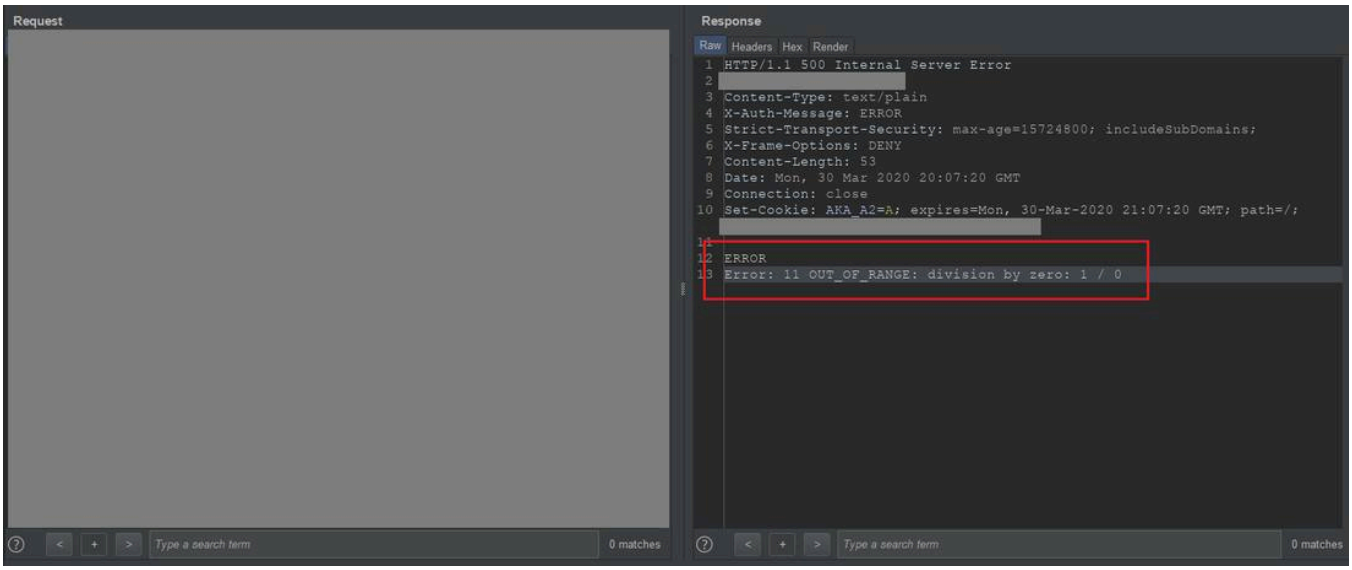
> **select AAA,BBB,CCC,DDD from EEE where location=\"abcd1337\"\"\n **

So in below exploit, I will try to extract the value of the **DDD **column from **EEE** table.

The silver bullet :

Request:

POST /vulnerable-endpoint/ HTTP/1.1
Host: 0x1337.space
...
location=ducnt1337" OR if(1/(length((select('a')))-1)=1,true,false) or "a"="b--



- Division by zero for the win*

And yes, Déjà vu.

Déjà vu

I was write up a CTF web challenge by abuse this division by zero error in 2017, you can check it out: **http://www.ducnt.net/2017/09/dzut-co-hon-phong-cach-ctf-web.html**

1: The concept.

First of all, I abuse the **length function **of google big query also the **division by zero **for detect the **true / false** (blind SQL injection) signature. Below is the concept. The payload I used to find the length of a string is:

```
**location=ducnt1337" OR if(1/(length((select("ducnt")))-5)=1,true,false) or "a"="b--**

In the google bigquery will be:
**select AAA,BBB,CCC,DDD from EEE where location="ducnt1337" OR if(1/(length((select("ducnt")))-5)=1,true,false) or '
```

Payload explain: If the length of the string **ducnt **(length = 5) minus 5, the result will be **1/0** and lead to division by zero error exception from database. Change the value after the minus math operation if you want to blind the length of value in column. So base on the concept of this, I can extract the value of the columns from database.

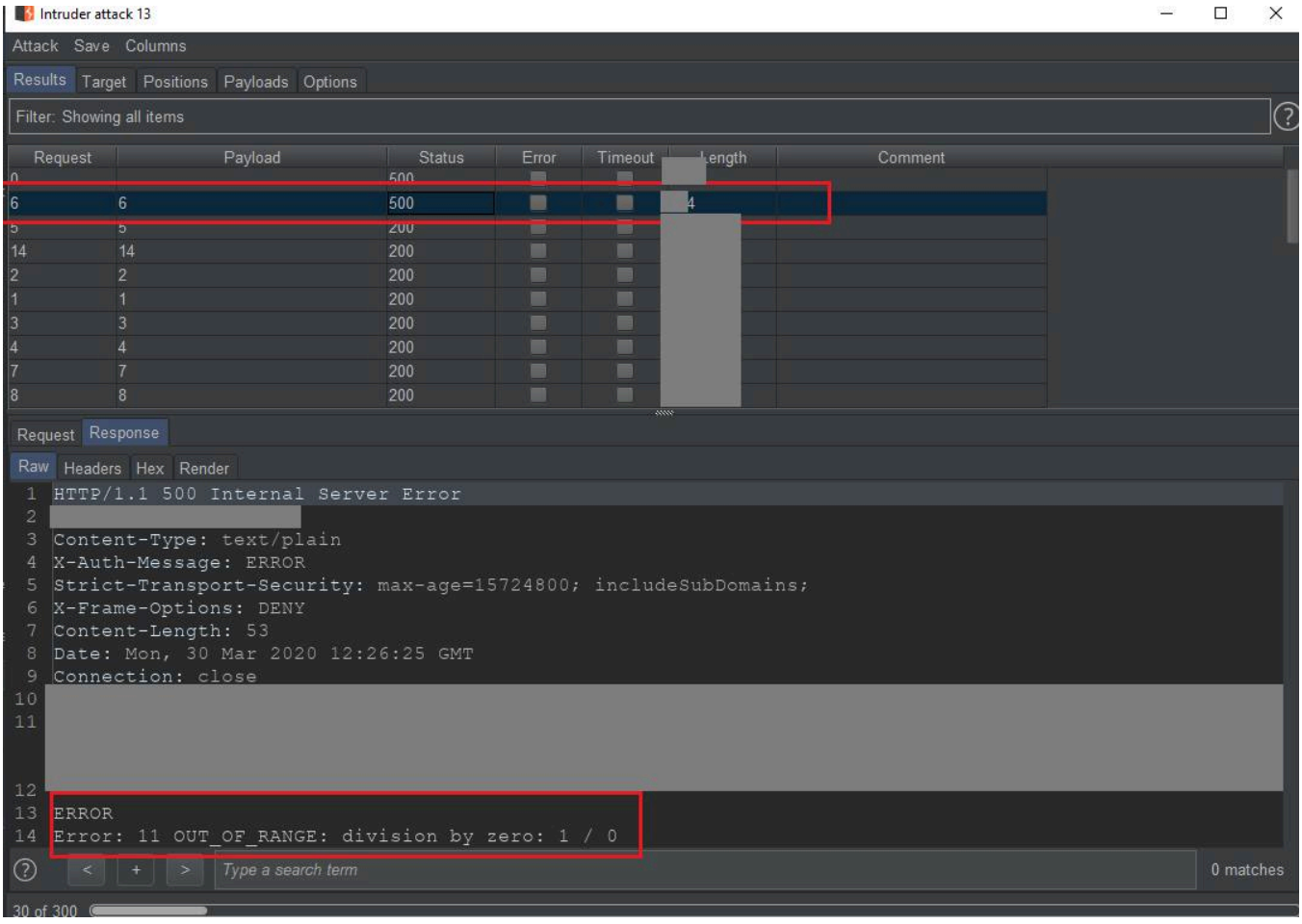
2: Exploit.

In below exploit, I try to extract the value of the **DDD **column from **EEE** table in database. So first of all, I will try to find the length of the value from **DDD **column (at **limit 1 position**).

a) Blind the length of the value from **DDD **column at the limit 1 position. The payload:

```
**location=ducnt1337" OR if(1/(length((select/**/DDD/**/limit/**/1))-$*BLIND_LENGTH_HERE*$)=1,true,false) or "a"="

In the google bigquery will be:
**select AAA,BBB,CCC,DDD from EEE where location="ducnt1337" OR if(1/(length((select/**/DDD/**/limit/**/1))-$*BLI
```

As you can see, if the length of the value from **DDD *column at the limit 1 position equal with the value *$BLIND_LENGTH_HERE$**, division by zero error exception will appear.



Length of the value was blinded equal 6

So after the exploit, going to a conclusion that the length of the value from **DDD** column at limit 1 position **equal 6**.

b) Blind the value from **DDD *column at the limit 1 position. To extract the value from the **DDD **column. I use **STRPOS **function of the google bigquery (REF: *https://cloud.google.com/bigquery/docs/reference/standard-sql/string_functions**) (ft) with division by zero error exception from the (1) and can extract the value from **DDD **column. The payload:

```
**location=ducnt" OR if(1/(STRPOS((select/**/DDD/**/limit/**/1),"$*BLIND_STRING_HERE*$"))=1,true,false) or "a"="b--
```

Let me explain the payload: In the $***BLIND_STRING_HERE***$ will be a single string you want to blind. The **STRPOS **function *will return the 1-based index of the first occurrence of substring inside string, returns 0 if substring is not found.*

For example. If the string ***a*** exists in the value from the **DDD **column, the value return from **STRPOS **function will not **equal 0**. So the content of the page will response (not the division by zero signature).

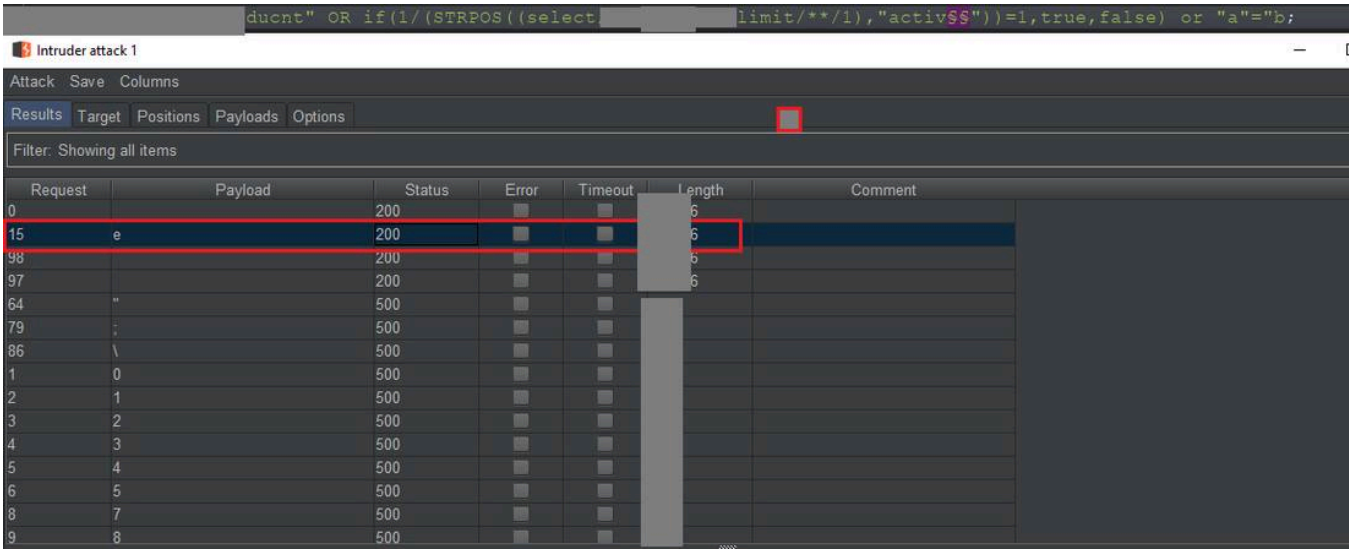
The next step is determined the string ***a*** was blinded locate at which position from the value. Simply add the minus math operation outside the query. The payload:

```
**location=ducnt" OR if(1/(STRPOS((select/**/DDD/**/limit/**/1),"a")-$*BLIND_THE_POSITION_HERE*$)=1,true,false) o
```

Basically, the string ***a*** from the value of the **DDD **column start at **1st position***, *result of the* ***STRPOS **minus math operation will **equal **with the $****BLIND_THE_POSITION_HERE*****$ if the division by zero signature appear. If not, the string** *a* **not locate at **1st position** from the value of the **DDD **column.

So in this context, I extracted the value of the first position from the **DDD **value is **a* **character. Repeat the process to blind all of the value. (**remember the length = 6**)

c) PoC: After the exploit, I can extract the value from **DDD **column at the limit 1 position and the value was: **active**



Bypass Kona WAF and extract info from database via SQL injection vulnerability successfully.

0x03: Response from the vendor and my thought.

From the vendor: Vendor confirmed the vulnerability also the WAF bypass exploit and bounty in the same day, the patch was released into the next day. Kudos to them for a working hard and

very quickly response, I really appreciate it.

[image: image]

Vendor's response

Finally, kudos to my master with very strong techniques advised: **g4mm4** [image: image] for helping me in the journey.

Take away:

Red team: Don't give up, keep digging and you will find something.

Blue team: Notice about every weirdly exception also the HTTP error status code. Everything has its reason.

WAF seller / consultant / farmer: ¯_( )_/¯

Thanks for reading and as usually, sorry for my bad engrisk * . *(TOEIC 900) ~Cheers, ducnt**

References:

- https://cloud.google.com/bigquery/docs/reference/standard-sql/string_functions
- https://cloud.google.com/spanner/docs/reference/rest/v1/Code
- https://twitter.com/gerben_javado/status/940220078947291137
- https://security.stackexchange.com/questions/201653/sql-injection-bypass-against-konaakmai-waf
- https://github.com/EdOverflow/bugbounty-cheatsheet/blob/master/cheatsheets/sqli.md
- http://www.ducnt.net/2017/09/dzut-co-hon-phong-cach-ctf-web.html