SQL Injection Isn't Dead

Smuggling Queries at the Protocol Level

Paul Gerste – RuhrSec 2025 – February 21, 2025

sonar

SQL INJECTION

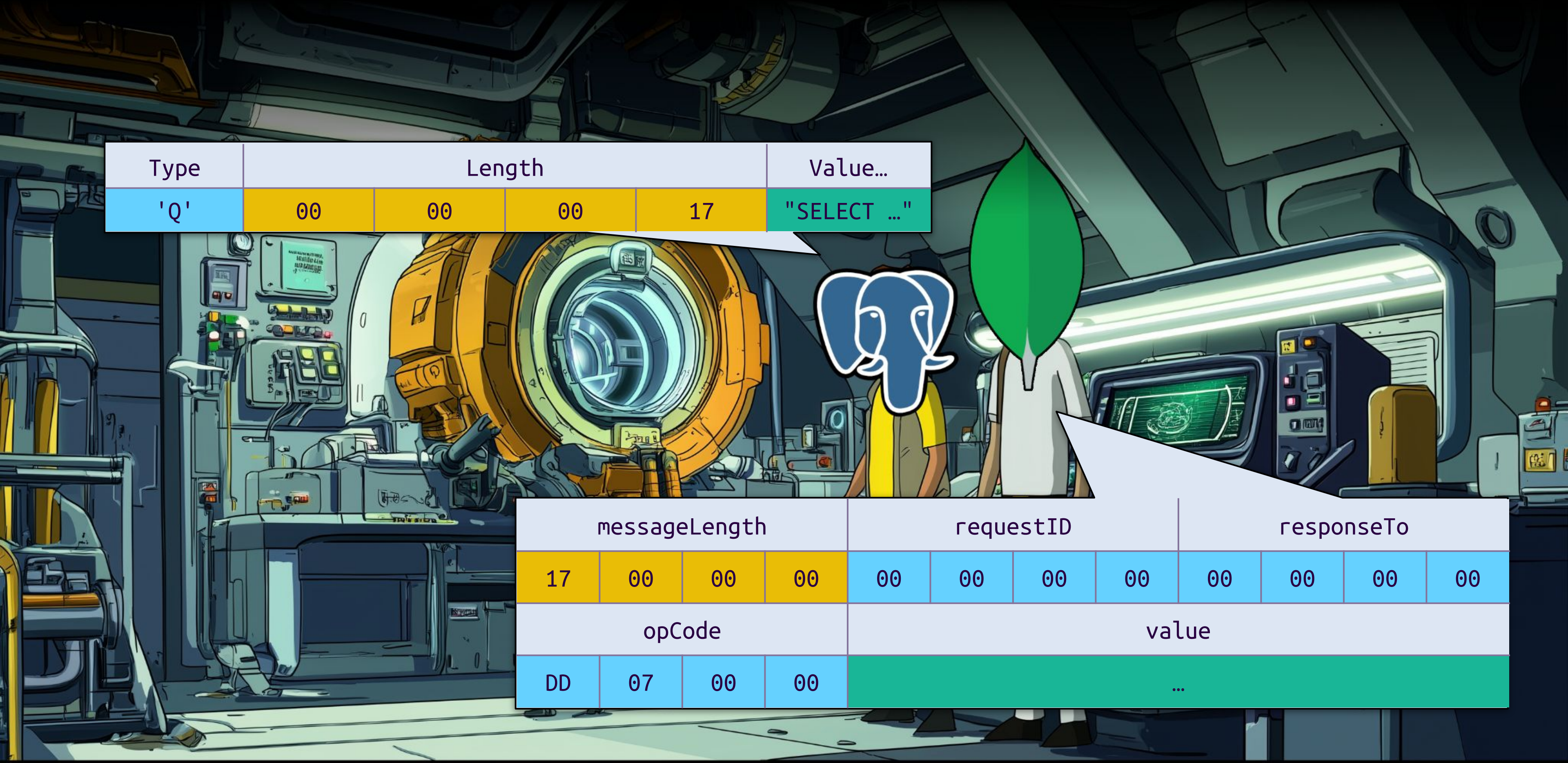
LOWER DECKS

sonar

```
HGETALL user:1
```

```
db.users.find({
  id: 1,
})
```

```
SELECT * FROM users WHERE id=1
```



Type	Length				Value…
'Q'	00	00	00	17	"SELECT …"

messageLength				requestID				responseTo			
17	00	00	00	00	00	00	00	00	00	00	00
opCode				value							
DD	07	00	00	…							

Teaser

```go
func getUser(w http.ResponseWriter, req *http.Request) (user User) {
    body, _ := io.ReadAll(req.Body)
    id := string(body)
    db.QueryRow("SELECT * FROM users WHERE id=$1", id).Scan(&user)
    // ...
}
```

SELECT * FROM speakers

- Paul Gerste / pspaul
  - Vulnerability Researcher at Sonar
- I love to break (web) stuff
- I love to play and organize CTFs with FluxFingers

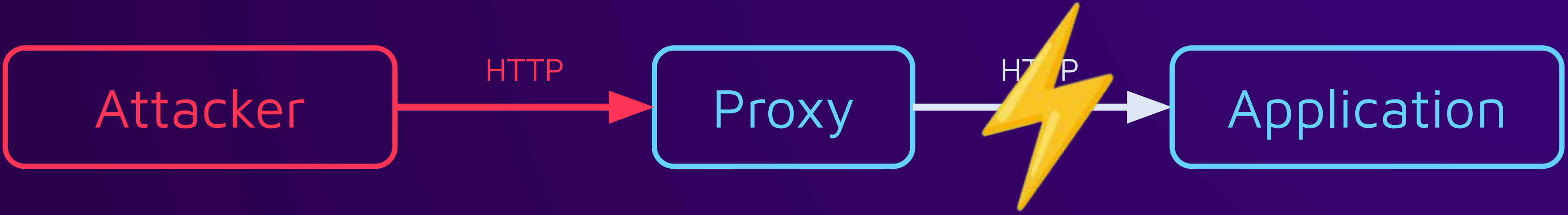
Outline

- The Idea
- Attacking Database Wire Protocols
  - PostgreSQL
  - MongoDB
- Real-World Applicability
- Future Research
- Takeaways

The Idea

Request smuggling, but for binary protocols

Request smuggling…



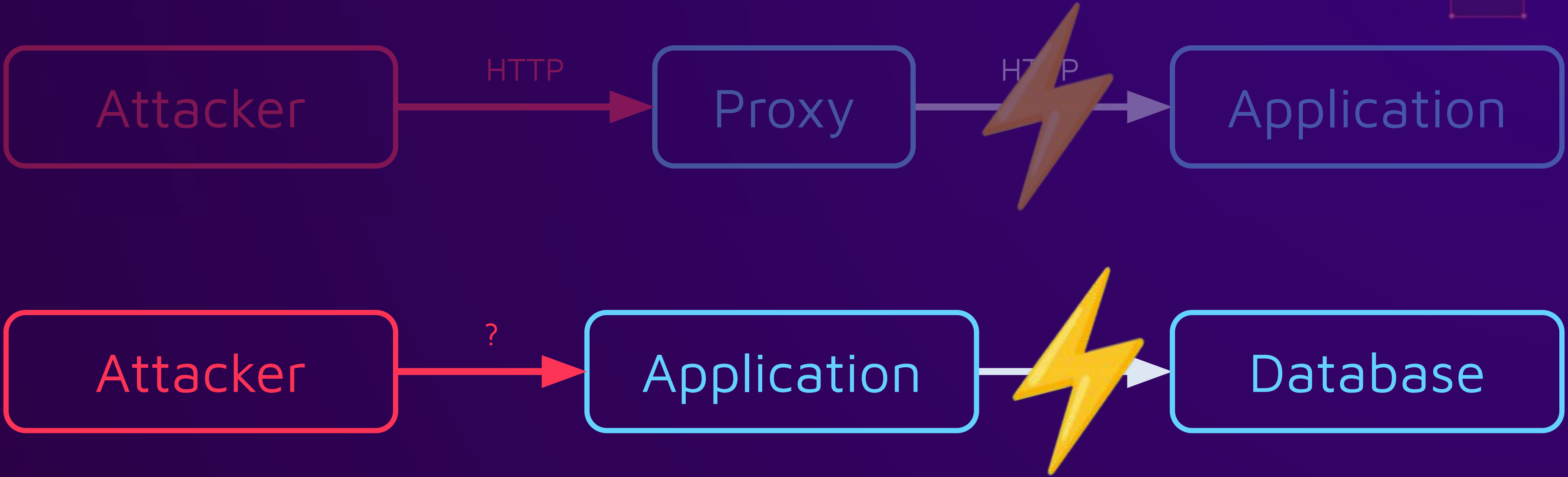
HTTP Desync Attacks: Request Smuggling Reborn

James Kettle

Director of Research

@albinowax

… but for binary protocols



©2024, SonarSource S.A, Switzerland.

Why **Database** Wire Protocols?

- Applicability
  - Almost every web app has a database
- Severity
  - Interesting data (e.g., PII)
  - Relevant data (e.g., for authentication)
- Exploitability
  - Most queries contain some user input

Attacking Database Wire Protocols

High-Level Protocol Comparison

- PostgreSQL
- MySQL
- MongoDB

High-Level Protocol Comparison

- **PostgreSQL**
- MySQL
- MongoDB

Type	Length				Value…
'Q'	00	00	00	17	"SELECT …"

High-Level Protocol Comparison

- PostgreSQL
- **MySQL**
- MongoDB

Type	Length				Value…
'Q'	00	00	00	17	"SELECT …"

Length			Sequence	Value…
00	00	17	00	"SELECT …"

High-Level Protocol Comparison

- PostgreSQL

Type	Length				Value…
'Q'	00	00	00	17	"SELECT …"

- MySQL

Length			Sequence	Value…
00	00	17	00	"SELECT …"

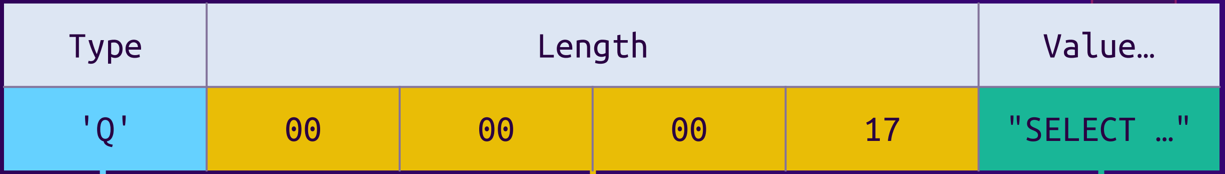
- **MongoDB**

messageLength				requestID				responseTo			
17	00	00	00	00	00	00	00	00	00	00	00
opCode				value							
DD	07	00	00	…							

Case Study:

PostgreSQL

PostgreSQL Wire Protocol



Type	Length				Value…
'Q'	00	00	00	17	"SELECT …"

- Type: 1-byte identifier
- Length: 4-byte integer
- Value

PostgreSQL Wire Protocol

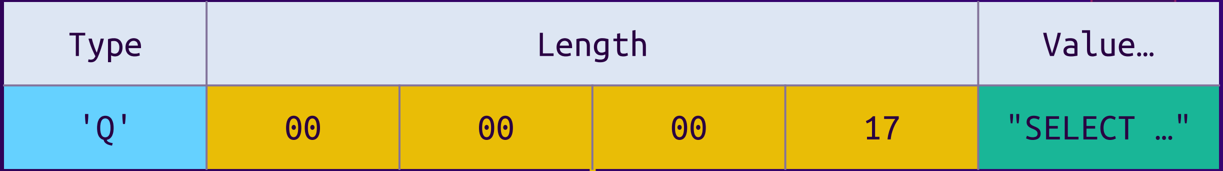
Type	Length				Value…
'Q'	00	00	00	17	"SELECT …"

- Type: 1-byte identifier
- Length: 4-byte integer
- Value

Max value: 2^{32}-1

sonar

PostgreSQL Wire Protocol



Type	Length				Value…
'Q'	00	00	00	17	"SELECT …"

- Type: 1-byte identifier
- Length: 4-byte integer
- Value

Max value: 2^{32}-1

The Bug: pgx

```go
func (src *Bind) Encode(dst []byte) []byte {
   dst = append(dst, 'B')
   sp := len(dst)
   // …
   pgio.SetInt32(dst[sp:], int32(len(dst[sp:])))
   return dst
}
```

The Bug: pgx

```go
func (src *Bind) Encode(dst []byte) []byte {
   dst = append(dst, 'B')
   sp := len(dst)
   // …
   pgio.SetInt32(dst[sp:], int32(len(dst[sp:])))
   return dst
}
```

Write message type

The Bug: pgx

```
func (src *Bind) Encode(dst []byte) []byte {
    dst = append(dst, 'B')
    sp := len(dst)
    // …
    pgio.SetInt32(dst[sp:], int32(len(dst[sp:])))
    return dst
}
```

Save size offset

The Bug: pgx

```go
func (src *Bind) Encode(dst []byte) []byte {
    dst = append(dst, 'B')
    sp := len(dst)
    // …
    pgio.SetInt32(dst[sp:], int32(len(dst[sp:])))
    return dst
}
```

Build the rest

The Bug: pgx

```go
func (src *Bind) Encode(dst []byte) []byte {
    dst = append(dst, 'B')
    sp := len(dst)
    // …
    pgio.SetInt32(dst[sp:], int32(len(dst[sp:])))
    return dst
}
```

Write size

The Bug: pgx

```
func (src *Bind) Encode(dst []byte) []byte {
    dst = append(dst, 'B')
    sp := len(dst)
    // …
    pgio.SetInt32(dst[sp:], int32(len(dst[sp:])))
    return dst
}
```

The message buffer

The Bug: pgx

```go
func (src *Bind) Encode(dst []byte) []byte {
    dst = append(dst, 'B')
    sp := len(dst)
    // …
    pgio.SetInt32(dst[sp:], int32(len(dst[sp:])))
    return dst
}
```

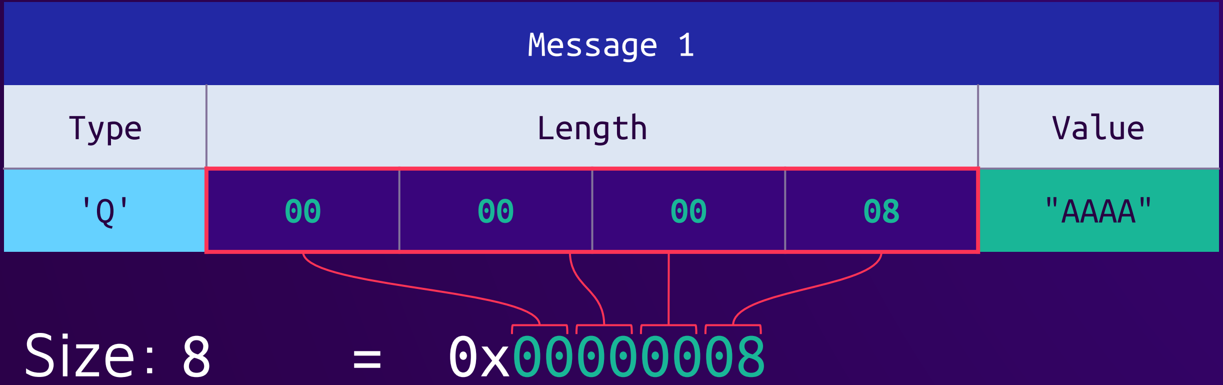
Buffer length (int)

The Bug: pgx

```
func (src *Bind) Encode(dst []byte) []byte {
    dst = append(dst, 'B')
    sp := len(dst)
    // …
    pgio.SetInt32(dst[sp:], int32(len(dst[sp:])))
    return dst
}
```

Truncate to int32

Message Size Overflow



Message 1					
Type	Length				Value
'Q'	00	00	00	08	"AAAA"

Size: 8 = 0x00000008

4 bytes length + 4 bytes data

Payload: "A" * 4

Message Size Overflow

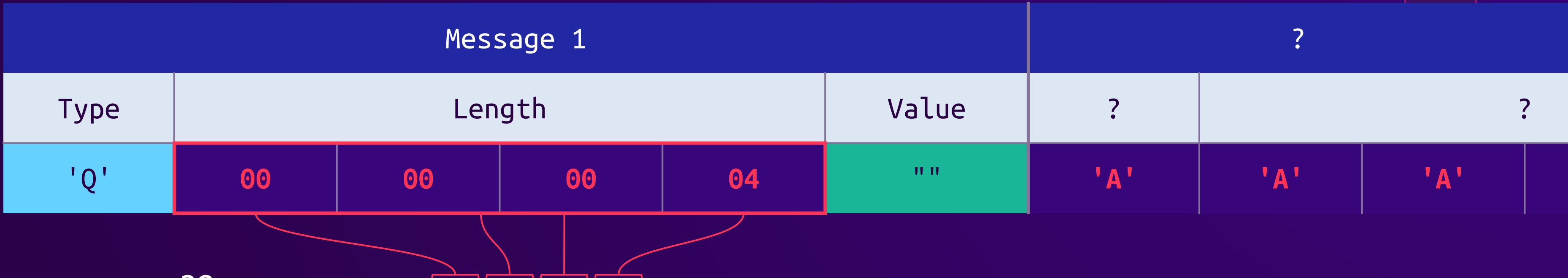
<table>
<tr><th colspan="6">Message 1</th></tr>
<tr><td>Type</td><td colspan="4">Length</td><td>Value</td></tr>
<tr><td>'Q'</td><td>FF</td><td>FF</td><td>FF</td><td>FF</td><td>"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA…"</td></tr>
</table>

Size: $2^{32}-1$ = 0xFFFFFFFF

4 bytes length + $2^{32}-5$ bytes data

Payload: "A" * (2**32 - 5)

Message Size Overflow



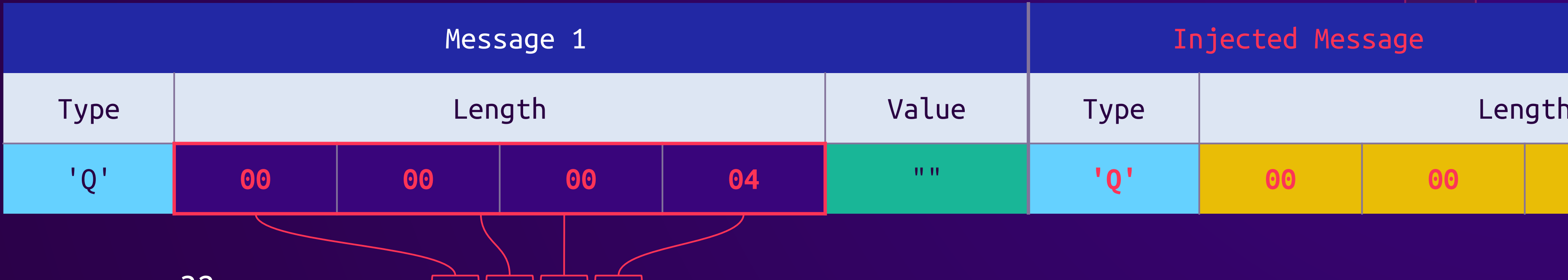
Message 1						?			
Type	Length				Value	?	?		
'Q'	00	00	00	04	""	'A'	'A'	'A'	

Size: $2^{32}+4$ = 0x100000004

4 bytes length + 2^{32} bytes data

Payload: "A" * (2**32)

Message Size Overflow



Message 1						Injected Message		
Type	Length				Value	Type	Length	
'Q'	00	00	00	04	""	'Q'	00	00

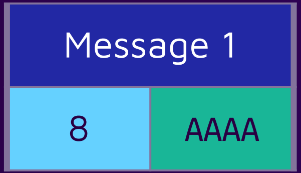
Size: $2^{32}+4$ = 0x100000004

4 bytes length + 2^{32} bytes data

Payload: fakeMsg + "A" * (2**32 - len(fakeMsg))

©2024, SonarSource S.A, Switzerland.

Message Size Overflow - Zoomed Out

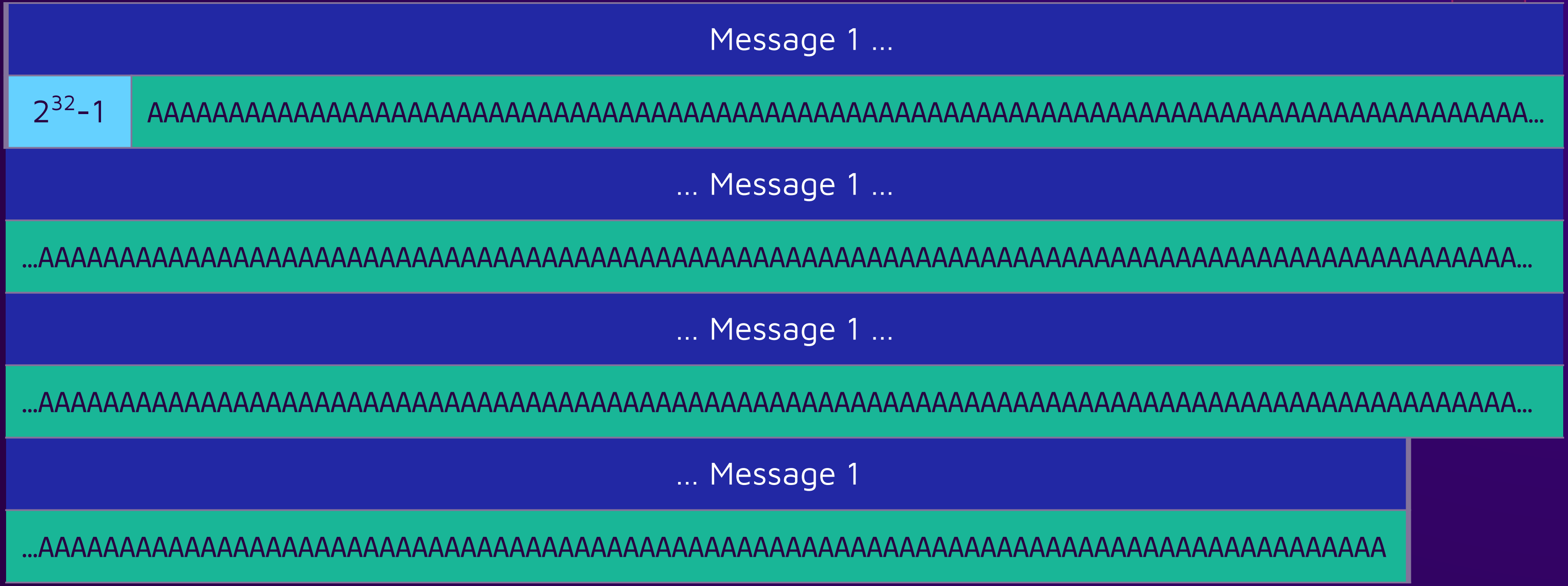


Message 1

8

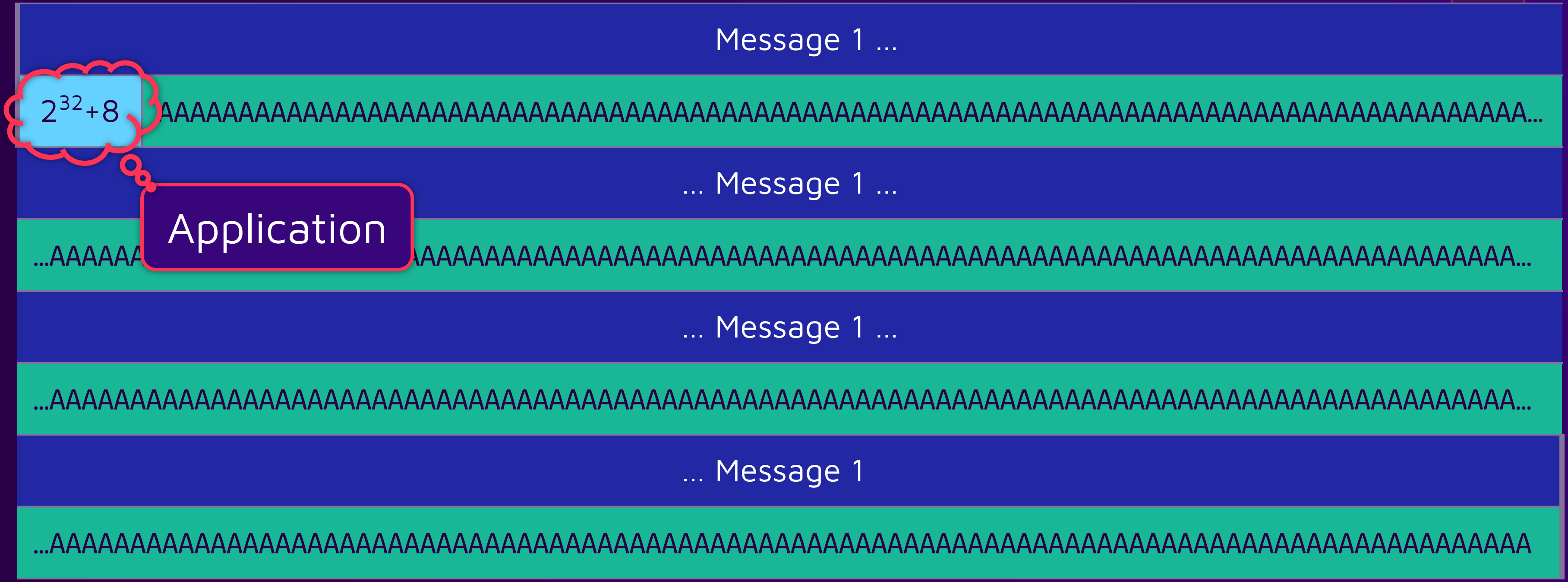
AAAA

Message Size Overflow - Zoomed Out



Message 1 ...
2^{32}-1
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...
... Message 1 ...
...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...
... Message 1 ...
...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...
... Message 1
...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Message Size Overflow - Zoomed Out



Message 1 ...

$2^{32}+8$ AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

... Message 1 ...

...AAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

Application

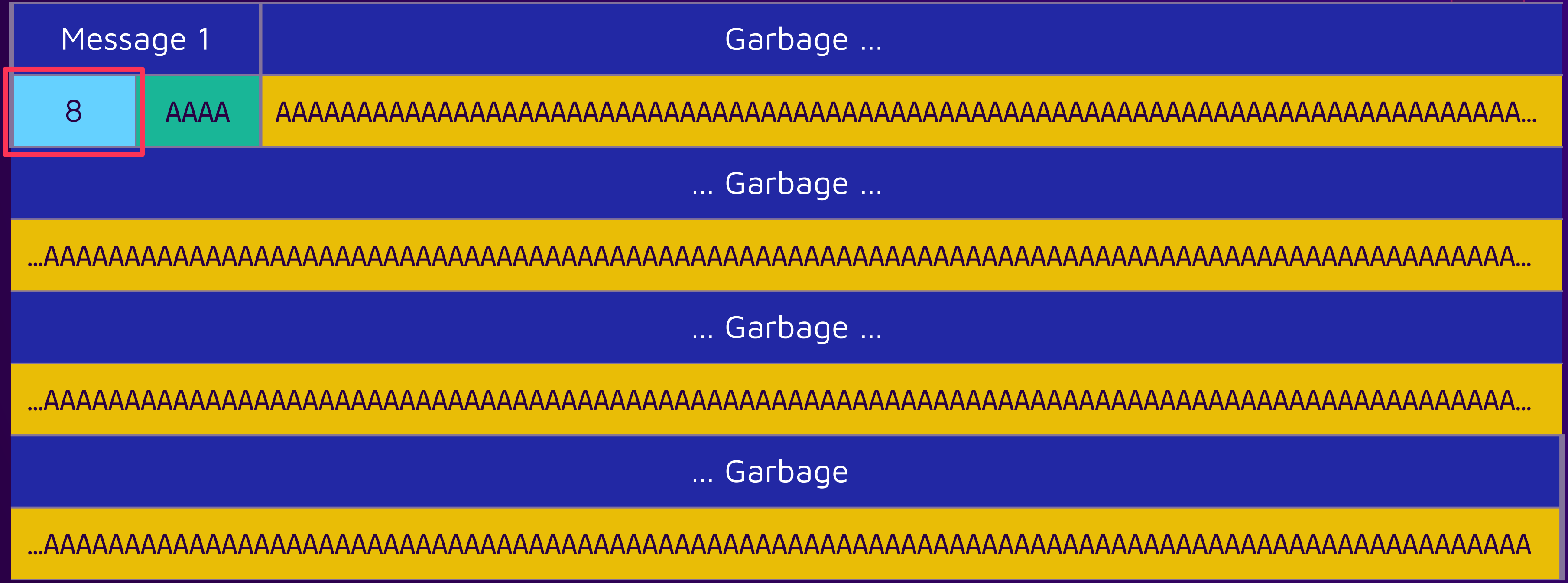
... Message 1 ...

...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

... Message 1

...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Message Size Overflow - Zoomed Out



Message 1

Garbage ...

8

AAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

... Garbage ...

...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

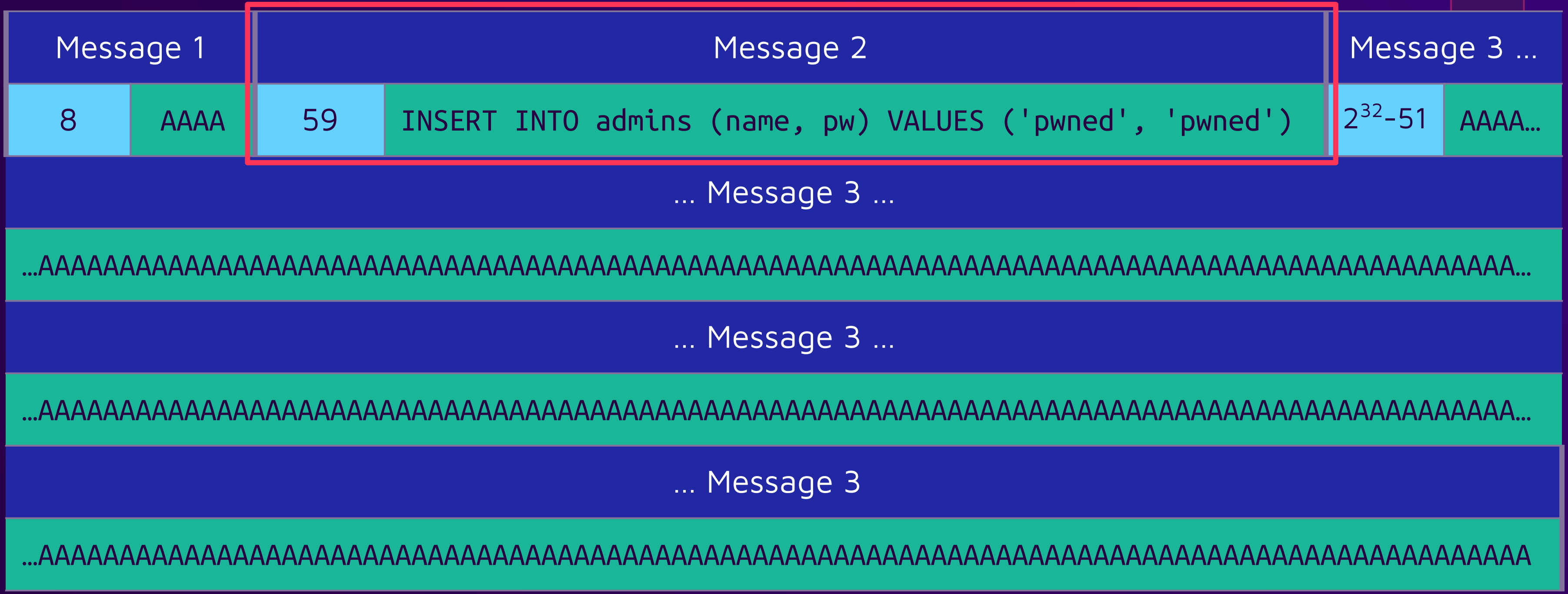
... Garbage ...

...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

... Garbage

...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Message Size Overflow - Zoomed Out



Message 1

8 AAAA

Message 2

59 INSERT INTO admins (name, pw) VALUES ('pwned', 'pwned')

Message 3 ...

2^{32}-51 AAAA...

... Message 3 ...

...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

... Message 3 ...

...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

... Message 3

...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Impact

- Inject entire SQL statements
  - Not limited to UNION, subqueries, etc.
  - Like stacked queries
- Read/write/delete all data in the DB
- Direct exfiltration is inconvenient
  - Application only processes the first DB response

How does it look in the real world?

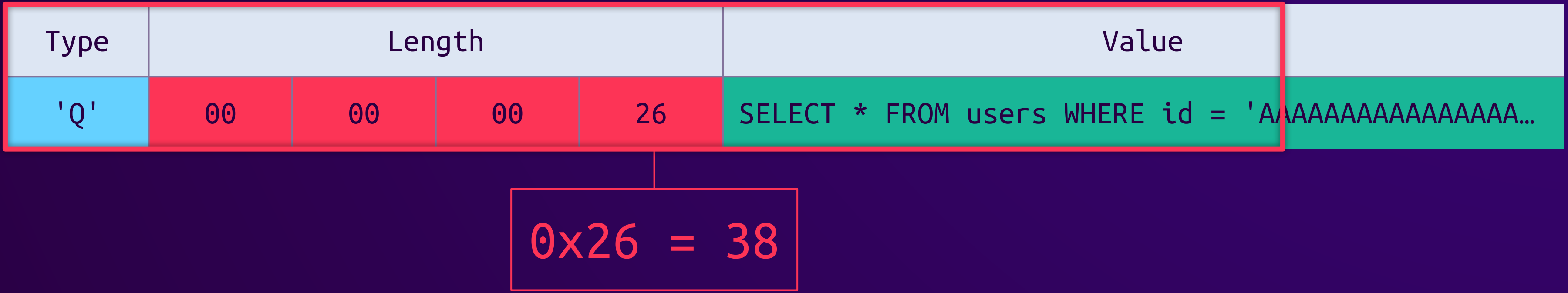
How does it look in the real world?

```
id := "5831bfeb"

conn.QueryRow("SELECT * FROM users WHERE id = $1", id)
```

Type	Length				Value
'Q'	00	00	00	2e	SELECT * FROM users WHERE id = '5831bfeb'\x00

How does it look in the real world?

```
id := strings.Repeat("A", 1<<32)

conn.QueryRow("SELECT * FROM users WHERE id = $1", id)
```

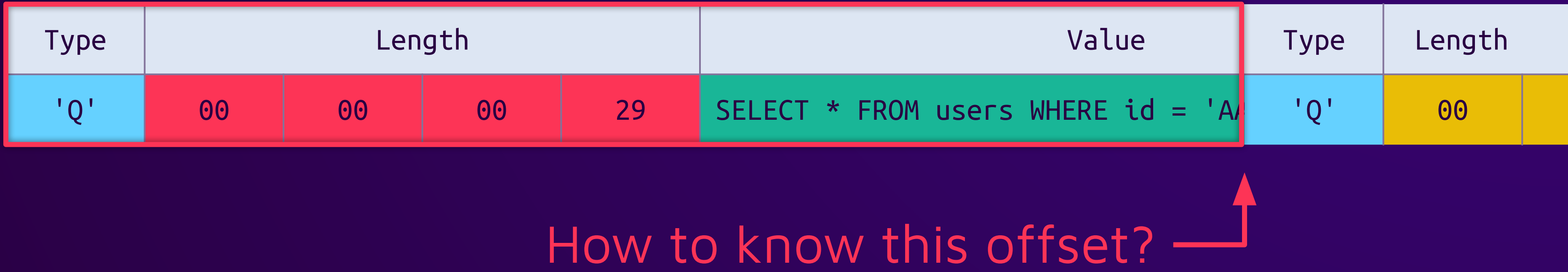


Type	Length				Value
'Q'	00	00	00	26	SELECT * FROM users WHERE id = 'AAAAAAAAAAAAAAAA…

0x26 = 38

How does it look in the real world?

```
id := strings.Repeat("A", 1<<32)

conn.QueryRow("SELECT * FROM users WHERE id = $1", id)
```



Type	Length				Value	Type	Length
'Q'	00	00	00	29	SELECT * FROM users WHERE id = 'A	'Q'	00

How to know this offset?

Crafting a Payload

- Offset depends on the query
  - Where is the injection point?
  - How long is the query?
- Calculate the offset when query is known
- What if it's not?

Crafting a Payload

- Naïve solution: Try all the offsets!
  - Need to send 4GB for each try
  - Takes time, creates noise
  - Risk of DoS
- Can we make it more reliable?

Crafting a Payload: NOP Sled

- Idea: NOP sled
  - Use a lot of small messages
  - Hit start of a message → success
  - Hit something else → connection closed

Crafting a Payload: NOP Sled

Smallest possible message

Type	Length			
'Q'	00	00	00	04

Crafting a Payload: NOP Sled

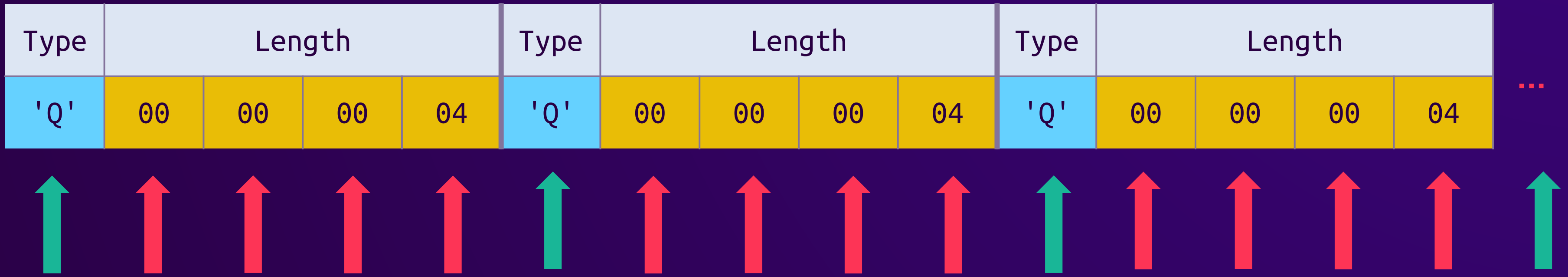
Type	Length				Type	Length				Type	Length			
'Q'	00	00	00	04	'Q'	00	00	00	04	'Q'	00	00	00	04

...

...

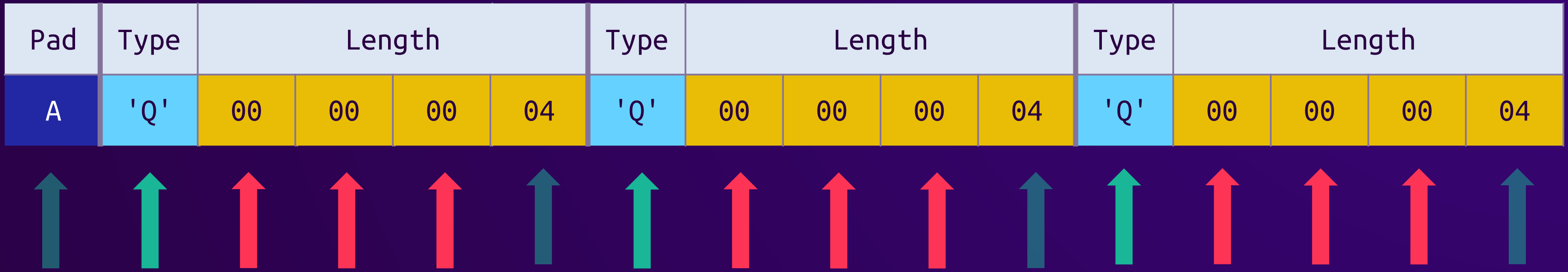
Type	Length				Type	Length				Value
'Q'	00	00	00	04	'Q'	00	00	00	3B	INSERT INTO admins VALUES …

Crafting a Payload: NOP Sled



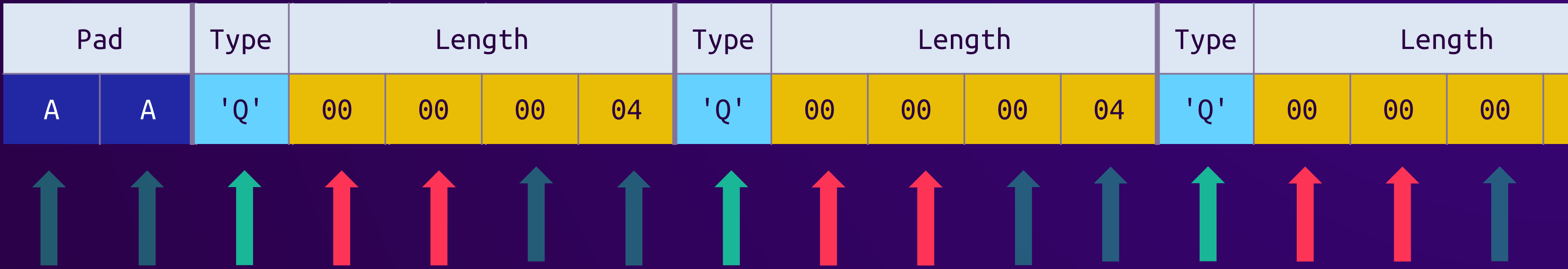
Type	Length				Type	Length				Type	Length				
'Q'	00	00	00	04	'Q'	00	00	00	04	'Q'	00	00	00	04	...

Crafting a Payload: NOP Sled



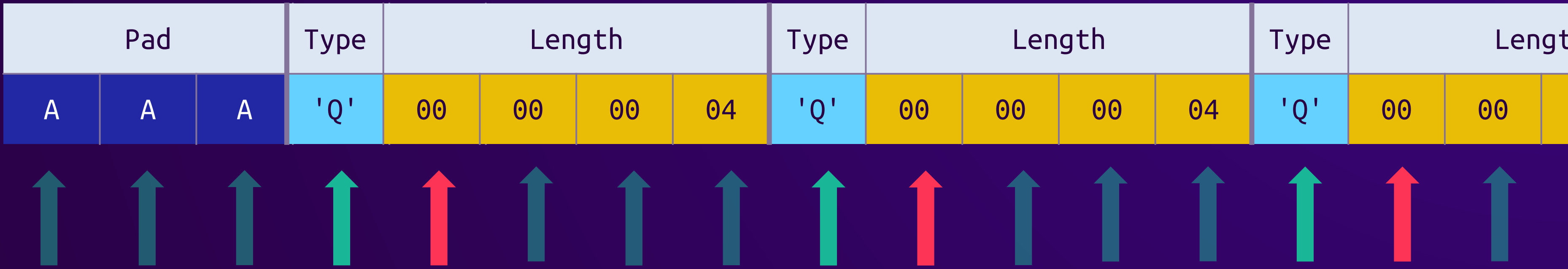
Pad	Type	Length				Type	Length				Type	Length			
A	'Q'	00	00	00	04	'Q'	00	00	00	04	'Q'	00	00	00	04

Crafting a Payload: NOP Sled



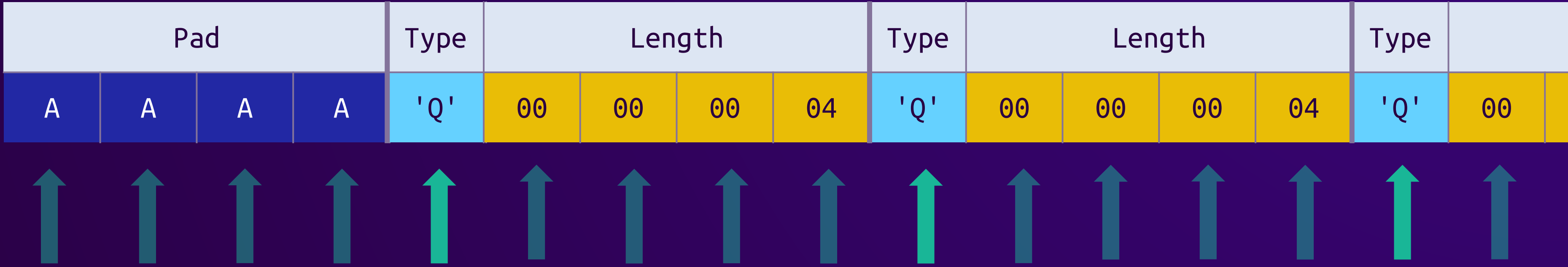
Pad		Type	Length				Type	Length				Type	Length		
A	A	'Q'	00	00	00	04	'Q'	00	00	00	04	'Q'	00	00	00

Crafting a Payload: NOP Sled



Pad			Type	Length				Type	Length				Type	Lengt	
A	A	A	'Q'	00	00	00	04	'Q'	00	00	00	04	'Q'	00	00

Crafting a Payload: NOP Sled



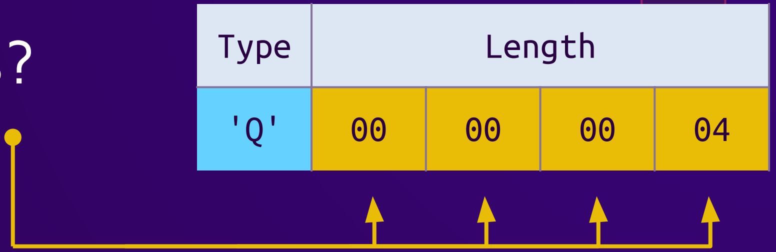
Pad				Type	Length				Type	Length				Type	
A	A	A	A	'Q'	00	00	00	04	'Q'	00	00	00	04	'Q'	00

Crafting a Payload: NOP Sled

- Success after ≤5 attempts!
  - 20% chance of success
  - Attack is repeatable, just change the offset
- Still have to send 5 x 4 GB in the worst case
  - Can we make it even better?

Crafting a Payload: Trampolines

- Can length bytes be valid types?



Type	Length			
'Q'	00	00	00	04

Crafting a Payload: Trampolines

- Can length bytes be valid types?
  - Trampolines!

Type	Length			
'Q'	'Q'	'Q'	'Q'	'Q'
	?	?	?	?

Crafting a Payload: Trampolines

- Can length bytes be valid types?
  - Trampolines!

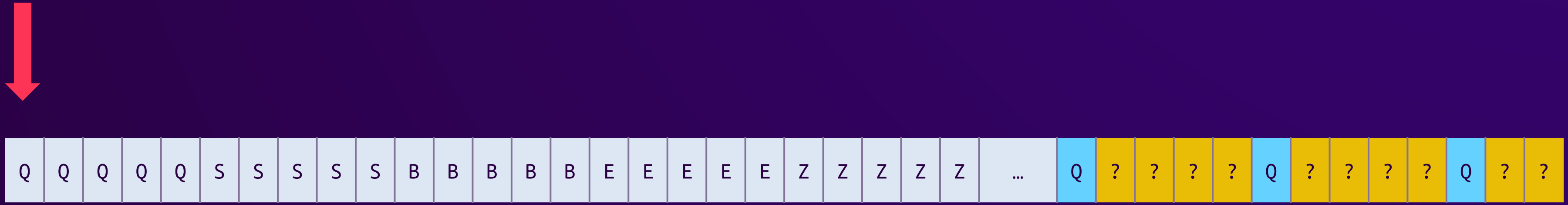
Type	Length			
'Q'	51	51	51	51
	?	?	?	?

Q	Q	Q	Q	Q	S	S	S	S	S	B	B	B	B	B	E	E	E	E	E	Z	Z	Z	Z	Z	…	Q	?	?	?	?	Q	?	?	?	?	Q	?	?

Crafting a Payload: Trampolines

- Can length bytes be valid types?
  - Trampolines!

Type	Length			
'Q'	51	51	51	51
	?	?	?	?



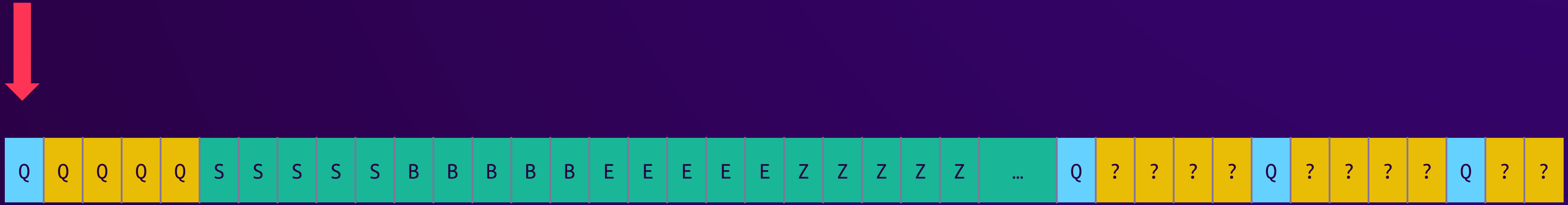
Q	Q	Q	Q	Q	S	S	S	S	S	B	B	B	B	B	E	E	E	E	E	Z	Z	Z	Z	Z	…	Q	?	?	?	?	Q	?	?	?	?	Q	?	?

Crafting a Payload: Trampolines

- Can length bytes be valid types?
  - Trampolines!

Type	Length			
'Q'	51	51	51	51

? ? ? ?

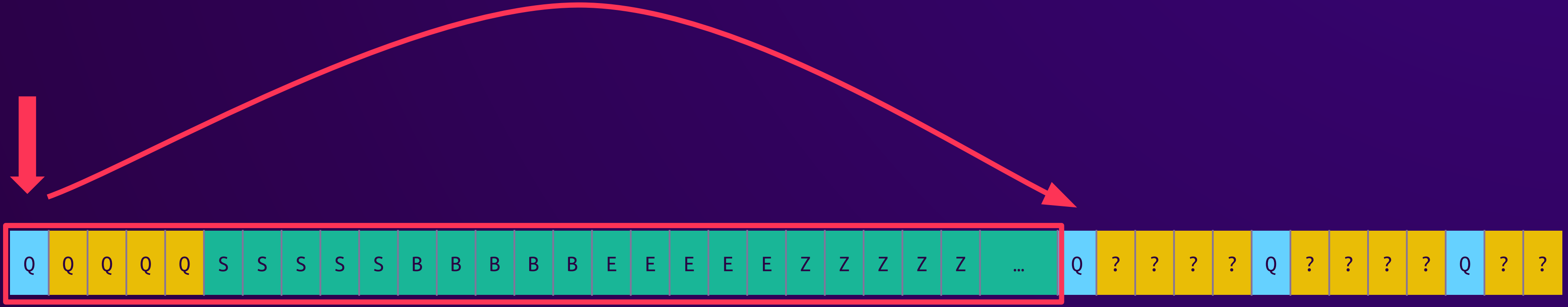


Q	Q	Q	Q	Q	S	S	S	S	S	B	B	B	B	B	E	E	E	E	E	Z	Z	Z	Z	Z	…	Q	?	?	?	?	Q	?	?	?	?	Q	?	?

Crafting a Payload: Trampolines

- Can length bytes be valid types?
  - Trampolines!

Type	Length			
'Q'	51	51	51	51
	?	?	?	?

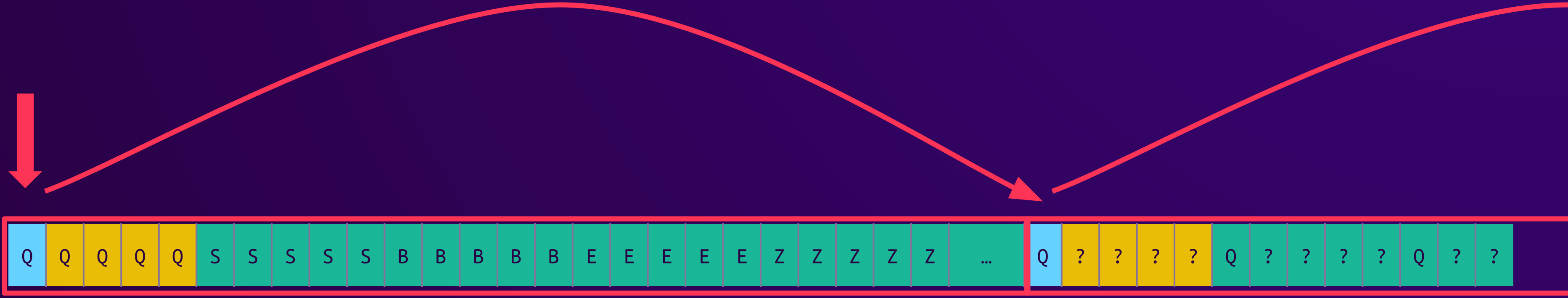


Q	Q	Q	Q	Q	S	S	S	S	S	B	B	B	B	B	E	E	E	E	E	Z	Z	Z	Z	Z	…	Q	?	?	?	?	Q	?	?	?	?	Q	?	?

Crafting a Payload: Trampolines

- Can length bytes be valid types?
    - Trampolines!

Type	Length			
'Q'	51	51	51	51
	?	?	?	?



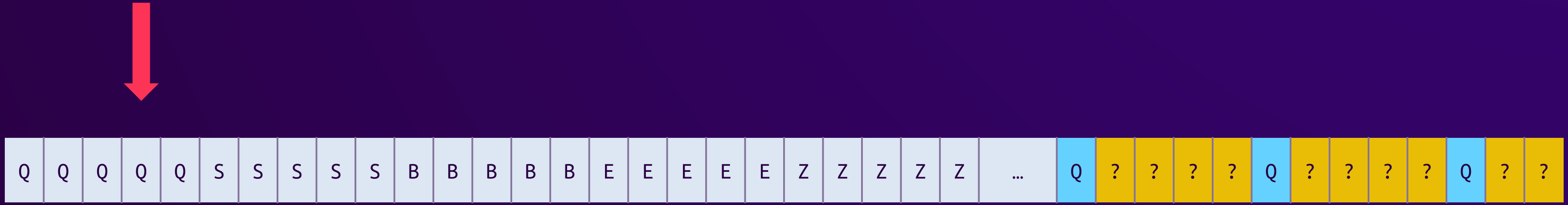
Q	Q	Q	Q	Q	S	S	S	S	S	B	B	B	B	B	E	E	E	E	E	Z	Z	Z	Z	Z	…	Q	?	?	?	?	Q	?	?	?	?	Q	?	?

Crafting a Payload: Trampolines

- Can length bytes be valid types?
    - Trampolines!

Type	Length			
'Q'	51	51	51	51

? ? ? ?

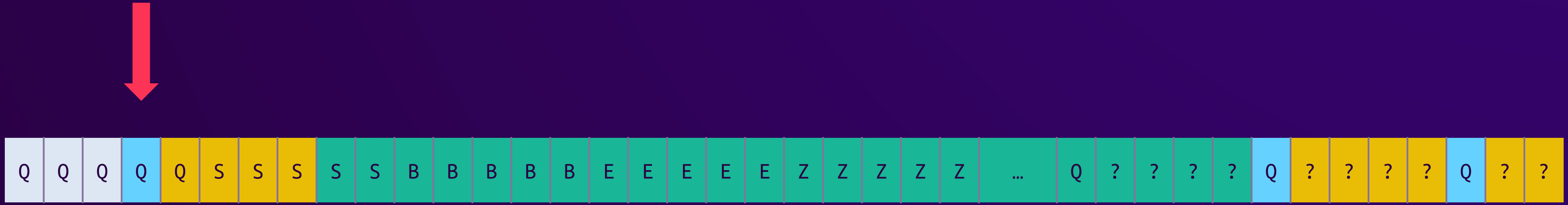


Q	Q	Q	Q	Q	S	S	S	S	S	B	B	B	B	B	E	E	E	E	E	Z	Z	Z	Z	Z	…	Q	?	?	?	?	Q	?	?	?	?	Q	?	?

Crafting a Payload: Trampolines

- Can length bytes be valid types?
  - Trampolines!

Type	Length			
'Q'	51	51	51	51
	?	?	?	?

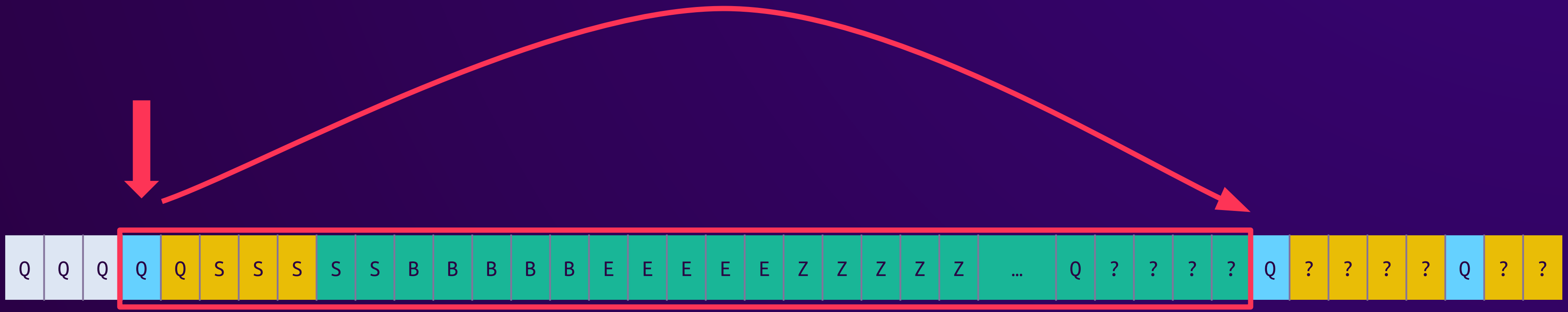


Q	Q	Q	Q	Q	S	S	S	S	S	B	B	B	B	B	E	E	E	E	E	Z	Z	Z	Z	Z	…	Q	?	?	?	?	Q	?	?	?	?	Q	?	?

Crafting a Payload: Trampolines

- Can length bytes be valid types?
    - Trampolines!

Type	Length			
'Q'	51	51	51	51
	?	?	?	?

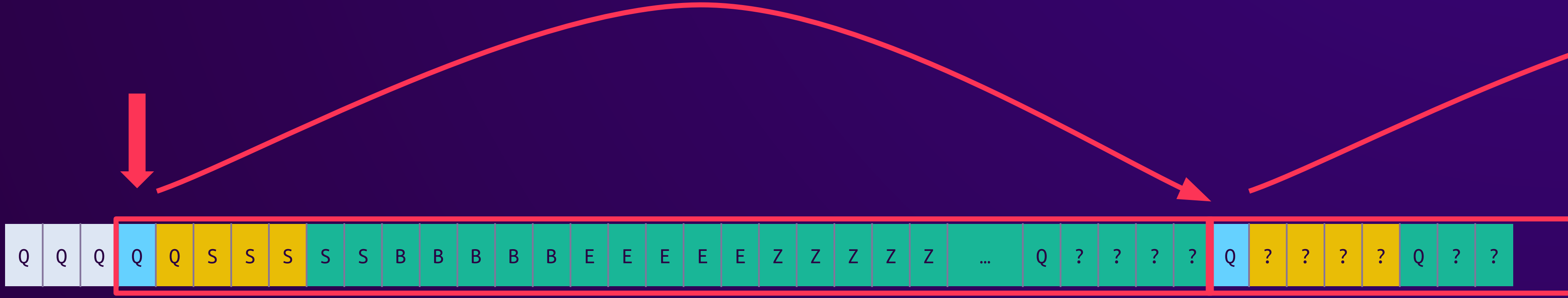


Q	Q	Q	Q	Q	S	S	S	S	S	B	B	B	B	B	E	E	E	E	E	Z	Z	Z	Z	Z	…	Q	?	?	?	?	Q	?	?	?	?	Q	?	?		

Crafting a Payload: Trampolines

- Can length bytes be valid types?
    - Trampolines!

Type	Length			
'Q'	51	51	51	51
	?	?	?	?



Q	Q	Q	Q	Q	S	S	S	S	S	B	B	B	B	B	E	E	E	E	E	Z	Z	Z	Z	Z	…	Q	?	?	?	?	Q	?	?	?	?	Q	?	?	

Crafting a Payload: Trampolines

- Can length bytes be valid types?
  - Trampolines!

Type	Length			
'Q'	51	51	51	51
	?	?	?	?

Crafting a Payload: Trampolines

- Can length bytes be valid types?
  - Trampolines!
- Max. logical size: `0x3fffffff`
  - First size byte cannot be > `0x3f`

Type	Length			
'Q'	51	51	51	51
✅	❌	✅	✅	✅

Crafting a Payload: Trampolines

- Can length bytes be valid types?
  - Trampolines!
- Max. logical size: `0x3fffffff`
  - First size byte cannot be > `0x3f`

Type	Length			
3f	3f	3f	3f	3f
?	?	?	?	?

Crafting a Payload: Trampolines

- Can length bytes be valid types?
  - Trampolines!
- Max. logical size: `0x3fffffff`
  - First size byte cannot be > `0x3f`
- No valid message type ≤ `0x3f`

Type	Length			
3f	3f	3f	3f	3f
❌	✅	✅	✅	✅

Crafting a Payload: Trampolines

- Can length bytes be valid types?
  - Trampolines!
- Max. logical size: `0x3fffffff`
  - First size byte cannot be > `0x3f`
- No valid message type ≤ `0x3f`
- Solution: alternating pattern

Type	Length			
'Q'	00	'Q'	00	'Q'
✅	❌	✅	❌	✅

Crafting a Payload: Trampolines

- Every **2nd** byte is a valid type
  - Hit a valid type byte → success
  - Hit other bytes → connection closed
- Success after ≤2 attempts!
  - 50% chance of success
  - Attack is repeatable, just change the offset

Type	Length			
'Q'	00	'Q'	00	'Q'
✅	❌	✅	❌	✅

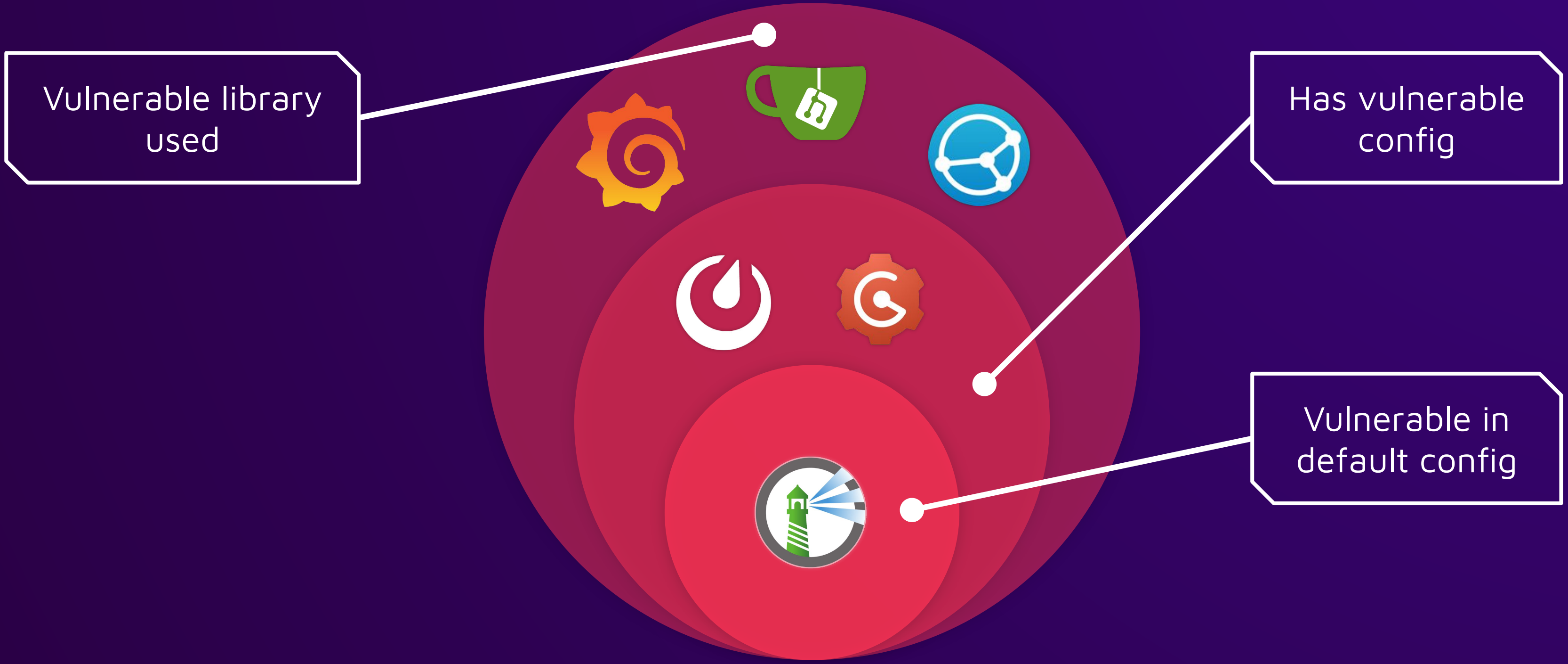
Vulnerable Libraries

Language	Library	Vulnerable?	Exploitable?	Fixed Versions
Go	pgx	✅	✅	4.18.2, 5.5.4
	pg	✅	✅	none
	pgdriver	✅	✅	none
	pq	✅	✅	none
C#/.NET	Npgsql	✅	✅	4.0.14, 4.1.13, 5.0.18, 6.0.11, 7.0.7, 8.0.3
Java	pgjdbc	❌	❌	-
	pgjdbc-ng	✅	❌	-
	r2dbc-postgresql	✅	❌	-
JS/TS	pg	✅	❌	-
	pg-promise	❌	❌	-
	pogi	✅	❌	-
	postgres	✅	❌	-
	@vercel/postgres	✅	❌	-

Disclosure Timeline

- Sent advisories in February 2024
- pgx fixed in March
- Npgsql fixed in May
- pg and pgdriver maintainer initially responded but then stopped
- pq maintainers never responded to issue/PR

Exploitable Applications



Vulnerable library used

Has vulnerable config

Vulnerable in default config

Demo: Harbor

- Container registry
  - CNCF Graduate project
  - Part of VMware Tanzu Kubernetes
- Default configuration was vulnerable
- No authentication required
- Fixed in 2.11.0 by updating pgx [1]

[1] https://github.com/goharbor/harbor/pull/20139

Case Study:

MongoDB

MongoDB Wire Protocol

<table>
<tr><td colspan="4">messageLength</td><td colspan="4">requestID</td><td colspan="4">responseTo</td></tr>
<tr><td>17</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td><td>00</td></tr>
<tr><td colspan="4">opCode</td><td colspan="8">value</td></tr>
<tr><td>DD</td><td>07</td><td>00</td><td>00</td><td colspan="8">…</td></tr>
</table>

- 4-byte length field
- Queries are BSON documents
  - Hierarchical objects
  - Serialized to TLV sections

The Bug: mongodb

```rust
async fn write_to<T: AsyncWrite + Send + Unpin>(&self, mut writer: T) -> Result<()> {
    let sections = self.get_sections_bytes();
    let total_length = Header::LENGTH
        + std::mem::size_of::<u32>()
        + sections.len()
        + /* ... */;
    let header = Header {
        length: total_length as i32,
        // ...
    };
    header.write_to(&mut writer).await?;
    writer.write_u32_le(self.flags.bits()).await?;
    writer.write_all(&sections).await?;
    // ...
}
```

The Bug: mongodb

```
async fn write_to<T: AsyncWrite + Send + Unpin>(&self, mut writer: T) -> Result<()> {
    let sections = self.get_sections_bytes();
    let total_length = Header::LENGTH
        + std::mem::size_of::<u32>()
        + sections.len()
        + /* ... */;
    let header = Header {
        length: total_length as i32,
        // ...
    };
    header.write_to(&mut writer).await?;
    writer.write_u32_le(self.flags.bits()).await?;
    writer.write_all(&sections).await?;
    // ...
}
```

Get content bytes

The Bug: mongodb

```
async fn write_to<T: AsyncWrite + Send + Unpin>(&self, mut writer: T) -> Result<()> {
   let sections = self.get_sections_bytes();
   let total_length = Header::LENGTH
       + std::mem::size_of::<u32>()
       + sections.len()
       + /* ... */;
   let header = Header {
       length: total_length as i32,
       // ...
   };
   header.write_to(&mut writer).await?;
   writer.write_u32_le(self.flags.bits()).await?;
   writer.write_all(&sections).await?;
   // ...
}
```

Calculate message size (usize)

The Bug: mongodb

```rust
async fn write_to<T: AsyncWrite + Send + Unpin>(&self, mut writer: T) -> Result<()> {
    let sections = self.get_sections_bytes();
    let total_length = Header::LENGTH
        + std::mem::size_of::<u32>()
        + sections.len()
        + /* ... */;
    let header = Header {
        length: total_length as i32,
        // ...
    };
    header.write_to(&mut writer).await?;
    writer.write_u32_le(self.flags.bits()).await?;
    writer.write_all(&sections).await?;
    // ...
}
```

Truncate to i32

Crafting a Payload

- Avoid bad bytes
  - Payload must be valid UTF-8
- Problem:
  - Message type (`dd 07`) is already invalid
  - Size fields can become invalid

Crafting a Payload

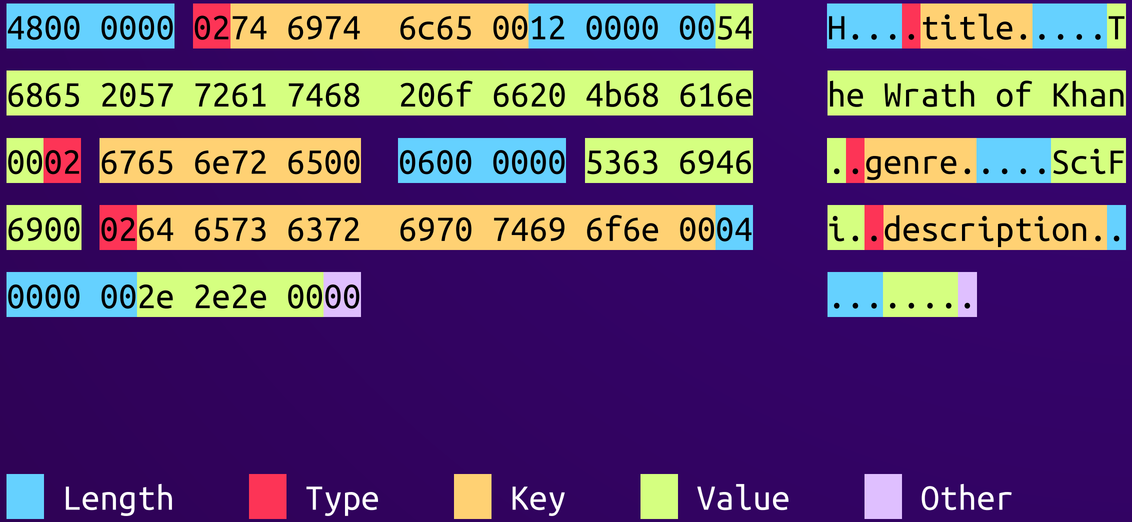
- Avoid bad bytes
  - Payload must be valid UTF-8
- Problem:
  - Message type (`dd 07`) is already invalid
  - Size fields can become invalid
- Solution:
  - Use metadata to create those bytes!

Crafting a Payload

Query:

```
{
  title: "The Wrath of Khan",
  genre: "SciFi",
  description: "...",
}
```

BSON Document:



```
4800 0000 0274 6974  6c65 0012 0000 0054    H....title.....T
6865 2057 7261 7468  206f 6620 4b68 616e    he Wrath of Khan
0002 6765 6e72 6500  0600 0000 5363 6946    ..genre.....SciF
6900 0264 6573 6372  6970 7469 6f6e 0004    i..description..
0000 002e 2e2e 0000                         ........
```

🟦 Length 🟥 Type 🟧 Key 🟩 Value 🟪 Other

Crafting a Payload

Query:

```
{
  title: "A" * (0x7dd - 1),
  genre: "SciFi",
  description: "...",
}
```

BSON Document:

```
1308 0000 0274 6974  6c65 00dd 0700 0054     H....title.....A
4141 4141 4141   ...   4141 4141 4141        AAAAA  ...  AAAAA
0002 6765 6e72 6500  0600 0000 5363 6946     ..genre.....SciF
6900 0264 6573 6372  6970 7469 6f6e 0004     i..description..
0000 002e 2e2e 0000                          ........
```

🟦 Length 🟥 Type 🟨 Key 🟩 Value 🟪 Other

Vulnerable Libraries

Language	Library	Vulnerable?	Exploitable?	Fixed Version
Rust	mongodb	✅	✅	2.8.2
Python	pymongo	❌	❌	-
Go	mongo	❌	❌	-
Java	mongo-java-driver	❌	❌	-
JavaScript	mongodb	❌	❌	-

- Sent advisory in February 2024
- mongodb fixed in March

Real-World Applicability

Constraints

How Web Apps Handle Large Payloads

- Aren't apps limiting input sizes?
- Common protections:
  - Default body size limits
  - Maximum JSON/form decode sizes
  - Size-limiting reverse proxies
  - … and more

How Web Apps Handle Large Payloads

- Potential bypasses
  - Unprotected endpoints
  - Compression
  - WebSockets
  - Server-side creation

How Web Apps Handle Large Payloads

- Potential bypasses
  - **Unprotected endpoints**
  - Compression
  - WebSockets
  - Server-side creation

- No default limits
- Disabled limits
  - Harbor

How Web Apps Handle Large Payloads

- Potential bypasses
  - Unprotected endpoints
  - **Compression**
  - WebSockets
  - Server-side creation

- Some enforce size limits **before** decompression
  - Nginx
  - Fastify

How Web Apps Handle Large Payloads

- Potential bypasses
  - Unprotected endpoints
  - Compression
  - **WebSockets**
  - Server-side creation

- Large message size
- Compression support
- Many filters don't apply

sonar

How Web Apps Handle Large Payloads

- Potential bypasses
  - Unprotected endpoints
  - Compression
  - WebSockets
  - **Server-side creation**

- Create strings on the server
  - SSRF, templates, …
- Depends on business logic

Language Comparison

- Silent integer overflows?
- How big can strings/buffers be?

Language Comparison: Integer Overflows

Language	Silent Addition Overflow?	Silent Serialization Overflow?
Go	Yes	N/A *
Java	Yes	N/A *
C#	Yes	N/A *
JS	No	Depends on impl.
Python	No	No
Rust	In release builds	N/A *

* Type system prevents overflows. Devs have to check for overflows, which leads to bugs

Language Comparison: Large Payloads

Language	Max. String Size	Max. Buffer Size
Go	$> 2^{32}$	$> 2^{32}$
Java	$2^{31}-1$	$2^{31}-1$
C#	$2^{31}-1$	$> 2^{32}$
JS	$2^{29}-24$ *	$> 2^{32}$ *
Python	$> 2^{32}$	$> 2^{32}$
Rust	$> 2^{32}$	$> 2^{32}$

Only considering 64-bit versions.

* Depends on the implementation

Real-World Applicability

- Can I send large payloads?
  - A lot of times, yes!
- Can integers silently overflow/truncate?
  - In many languages, yes!
- Can I exploit real-world apps with this?
  - Absolutely!

Future Research

Safety First: No DoS Please!

⚠️

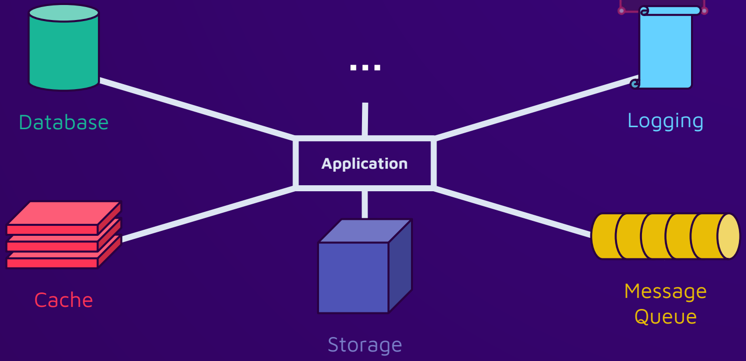
Do not send large payloads to third-party systems!

Non-Invasive Detection

- White-box tests are harmless
  - Just set up your own test environment
- How to test this black-box?
  - Sending large payloads risks DoS
- More research and tools needed!
  - Can we safely detect vulnerable libraries?
  - Build tools to test this safely

Research More!

- More protocols
  - Other databases
  - Caches, message queues, …
- Find more desync techniques
  - What about delimiters?
- More "large payload" methods
  - New ways to bypass limits
  - Generic server-side creation techniques



Research More!

- All this was about **4-byte** length fields
- What about 2-byte fields?
  - Much easier to exploit (65KB vs. 4GB)

Example - MQTT

```go
func encodeBytes(field []byte) []byte {
    fieldLength := make([]byte, 2)
    binary.BigEndian.PutUint16(fieldLength, uint16(len(field)))
    return append(fieldLength, field...)
}
```

- "Overflow" from topic into payload

Example - RADIUS

```
let size = RADIUS_PACKET_HEADER_LENGTH as u16 + encoded_avp.len() as u16;
if size as usize > MAX_PACKET_LENGTH {
   return Err("packet is too large".to_owned());
}
```

- Inject RADIUS packets
- Potential to spoof an Access-Accept response

Conclusion

Takeaways

- Integer overflows are still relevant in memory-safe languages
- Sending large amounts of data is feasible
- SQL injection isn't dead
  - If you can't hack it, just go a level deeper!

Thank you!

@Sonar_Research

@SonarResearch@infosec.exchange

https://sonarsource.com

@pspaul95

@pspaul@infosec.exchange

@pspaul95.bsky.social

sonar