SQL Injection Isn't Dead

Smuggling Queries at the Protocol Level

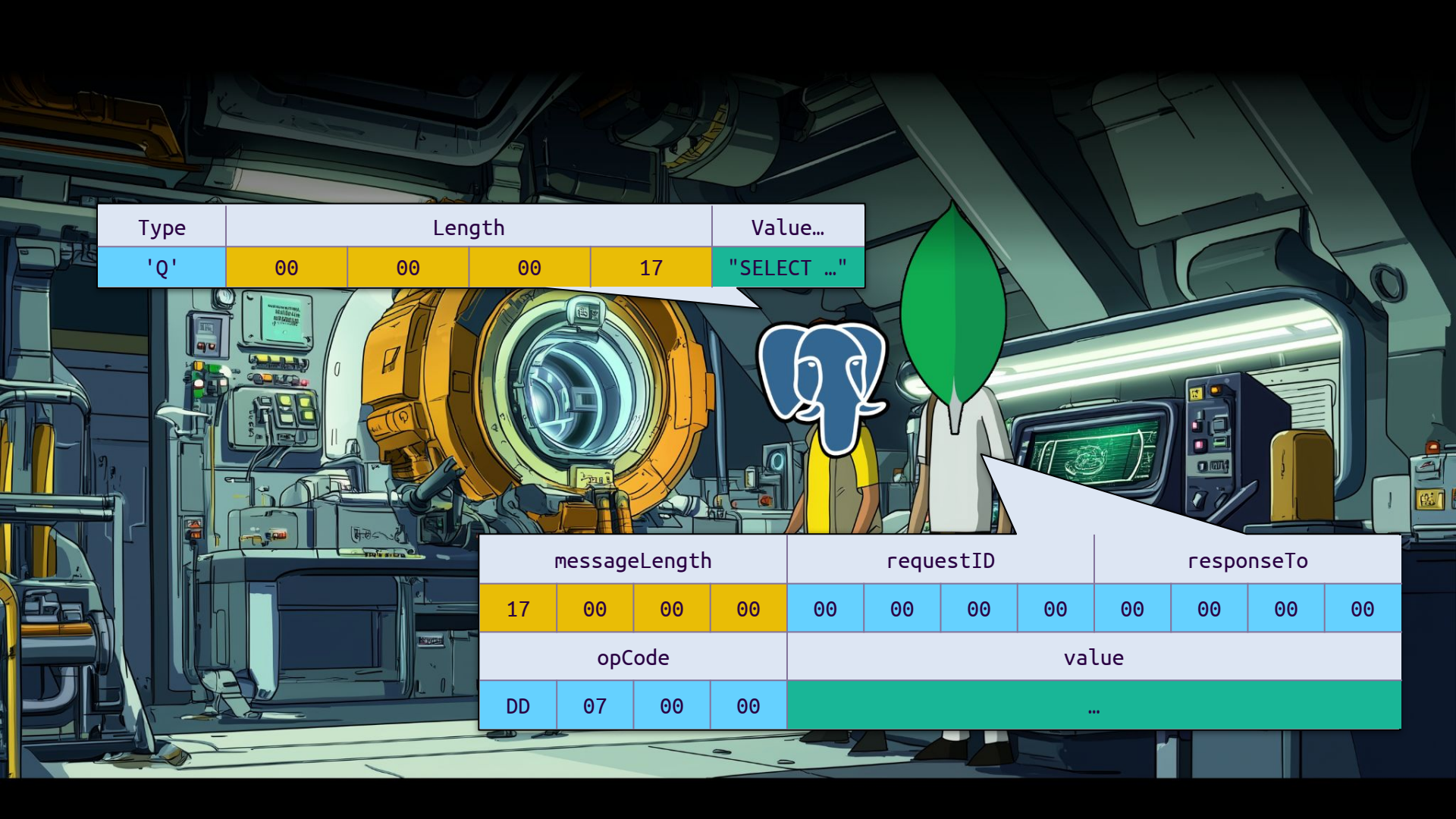
Paul Gerste – Hack.lu 2024 – October 24, 2024

sonar

SQL INJECTION LOWER DECKS

sonar


HGETALL user:1
SELECT * FROM users WHERE id=1
db.users.find({
id: 1,
})
MySQL


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

```
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)
    // ...
}
```


sonar

SELECT * FROM speakers

- Paul Gerste
  - Vulnerability Researcher at Sonar
- I love to break (web) stuff
- I love to play and organize CTFs with FluxFingers
  - Hack.lu CTF challenges are still up!

sonar

Outline

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

sonar

The Idea

Request smuggling, but for binary protocols

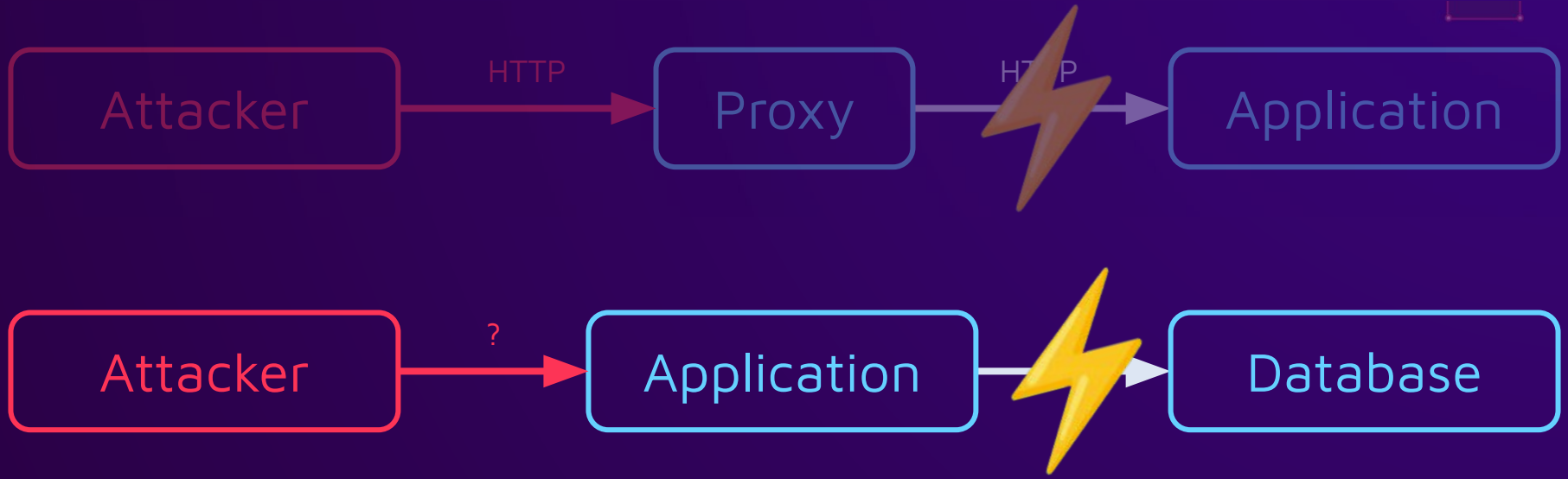
Request smuggling...

HTTP Desync Attacks: Request Smuggling Reborn

James Kettle
Director of Research
@albinowax

sonar

... but for binary protocols



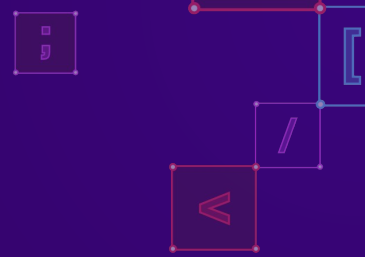
sonar

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

sonar

Attacking Database Wire Protocols



sonar

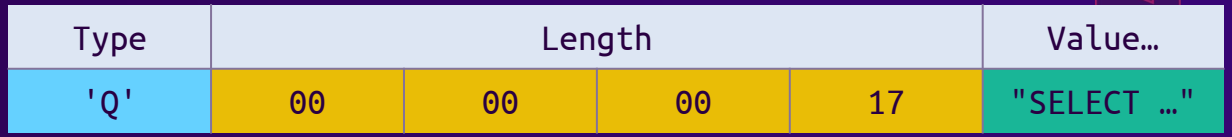
High-Level Protocol Comparison

- PostgreSQL
- MySQL
- MongoDB

sonar

High-Level Protocol Comparison

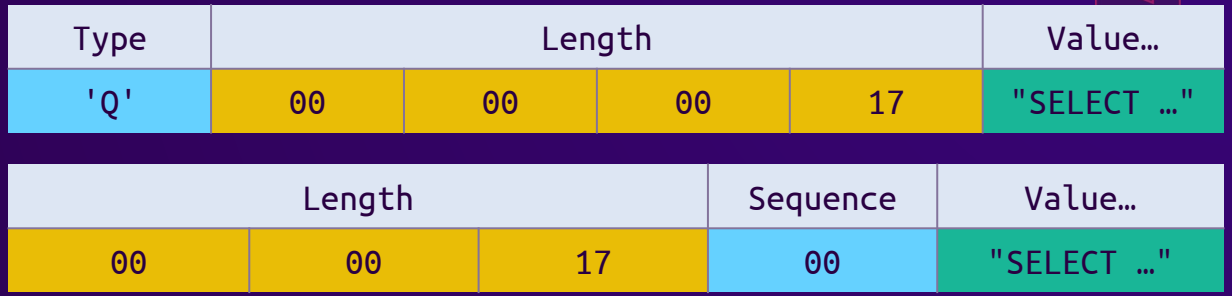
- **PostgreSQL**
- MySQL
- MongoDB



sonar

High-Level Protocol Comparison

- PostgreSQL
- **MySQL**
- MongoDB



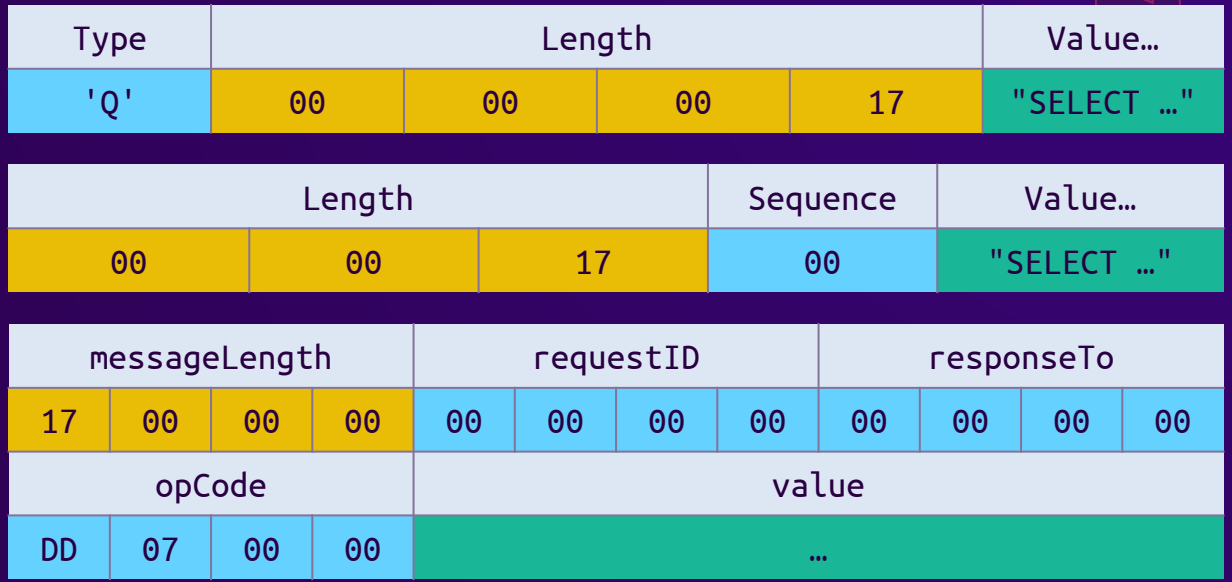
Type	Length				Value…
'Q'	00	00	00	17	"SELECT …"

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

sonar

High-Level Protocol Comparison

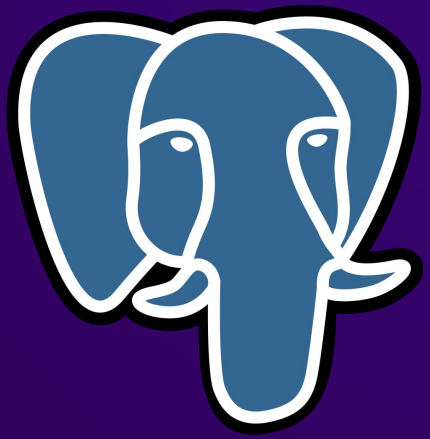
- PostgreSQL
- MySQL
- **MongoDB**



sonar

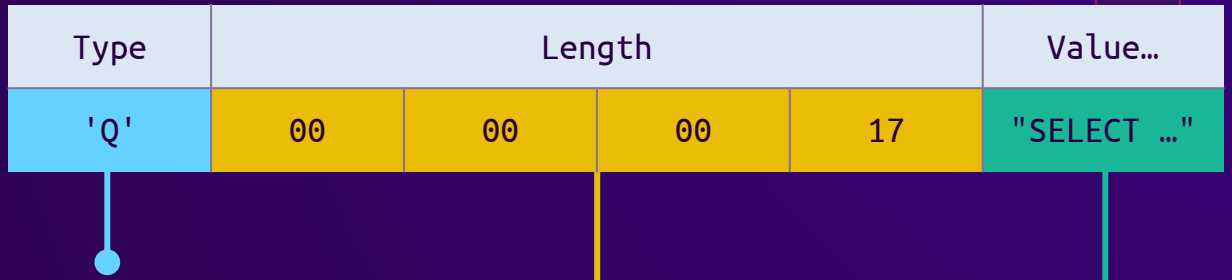
Case Study:

PostgreSQL



sonar

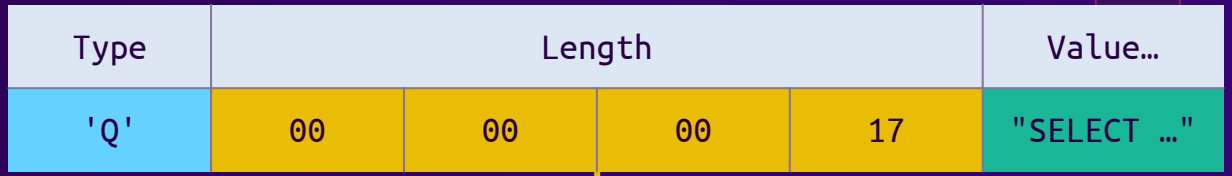
PostgreSQL Wire Protocol



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

sonar

PostgreSQL Wire Protocol

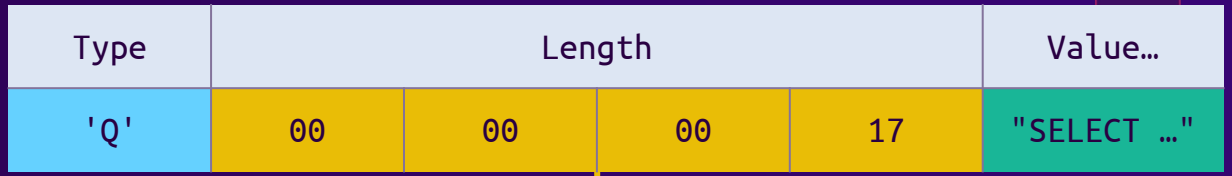


- Type: 1-byte identifier
- Length: 4-byte integer
- Value

Max value: $2^{32}-1$

sonar

PostgreSQL Wire Protocol



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

Max value: $2^{32}-1$

sonar

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
}
```


sonar

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
}
```

Write message type

sonar

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

sonar

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
}
```

Build the rest

sonar

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
}
```

Write size

sonar

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

sonar

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
}
```

Buffer length (int)

sonar

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`

sonar

Message Size Overflow

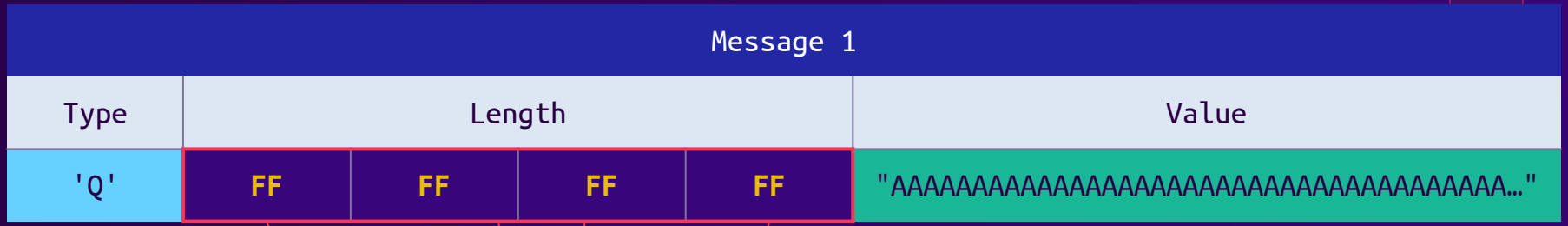
Size: 8 = 0x00000008

4 bytes length + 4 bytes data

Payload: "A" * 4

sonar

Message Size Overflow



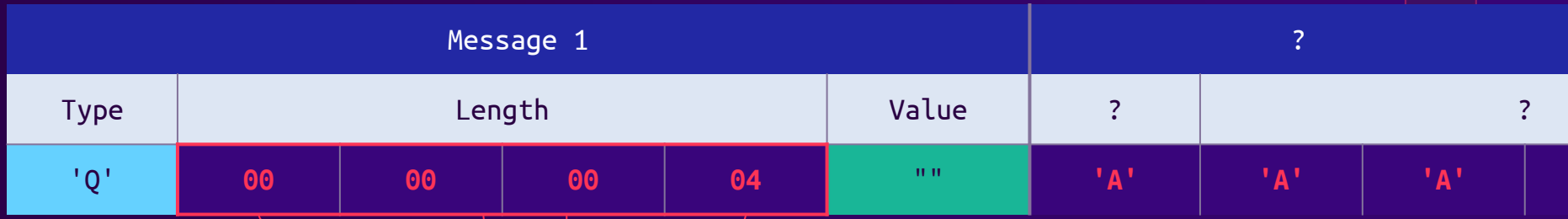
Size: 2^{32}-1 = 0xFFFFFFFF

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

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

sonar

Message Size Overflow



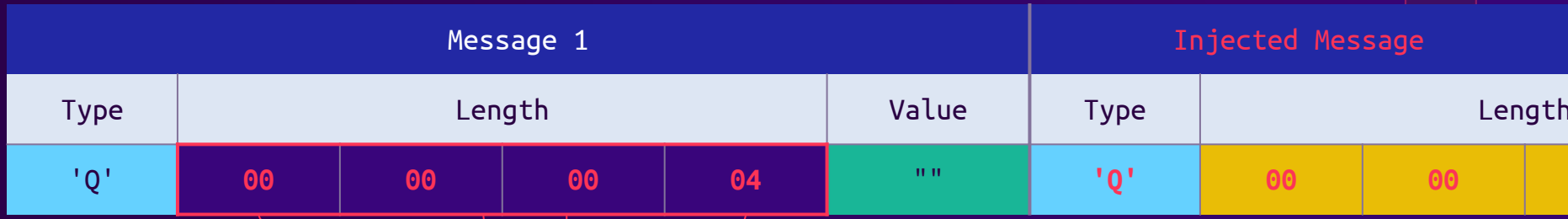
Size: $2^{32}+4$ = 0x100000004

4 bytes length + 2^{32} bytes data

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

sonar

Message Size Overflow



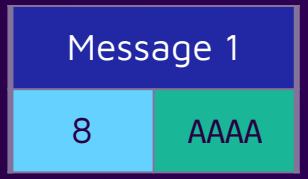
Size: $2^{32}+4$ = 0x100000004

4 bytes length + 2^{32} bytes data

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

sonar

Message Size Overflow - Zoomed Out



Message Size Overflow - Zoomed Out

Message 1 ...

2^{32}-1 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

... Message 1 ...

...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

... Message 1 ...

...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

... Message 1

...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Message Size Overflow - Zoomed Out

Message 1 ...

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

... Message 1 ...

Application

...AAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

... Message 1 ...

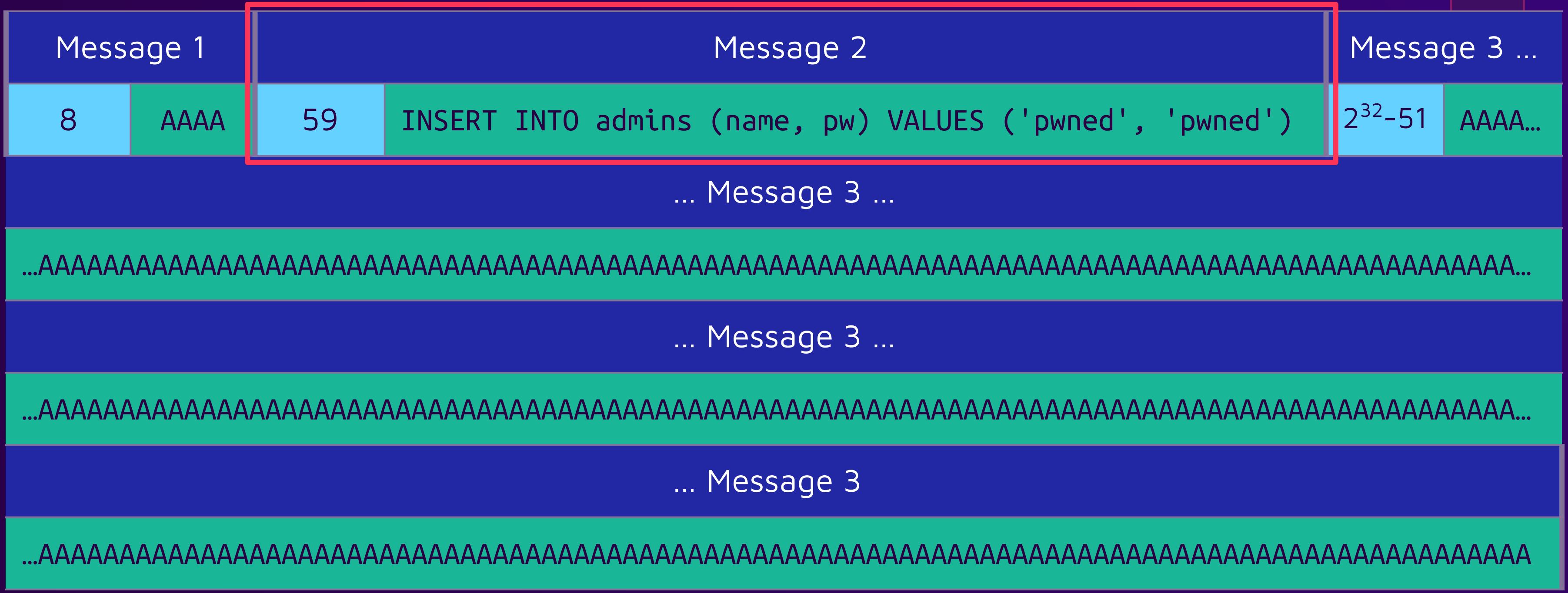
...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

... Message 1

...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Message Size Overflow - Zoomed Out

Message Size Overflow - Zoomed Out



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)
```

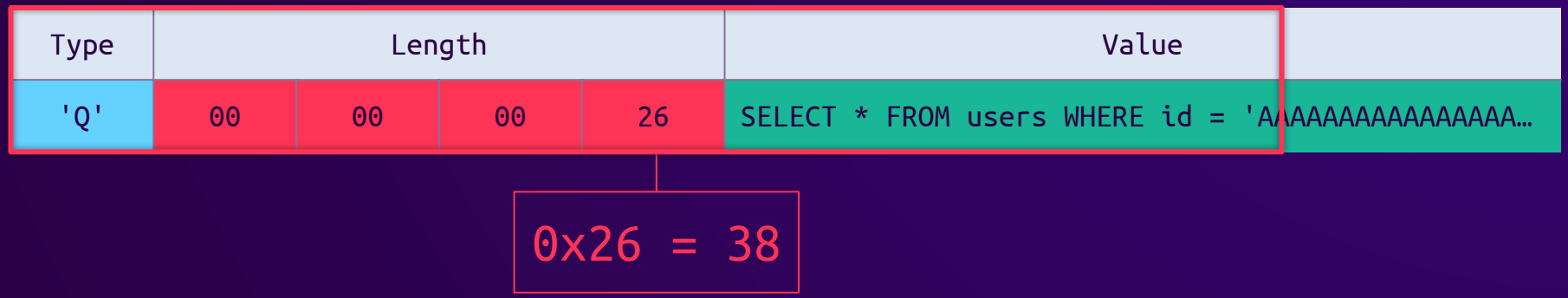
Type	Length				Value
'Q'	00	00	00	2e	SELECT * FROM users WHERE id = '5831bfeb'\x00

sonar

How does it look in the real world?

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

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

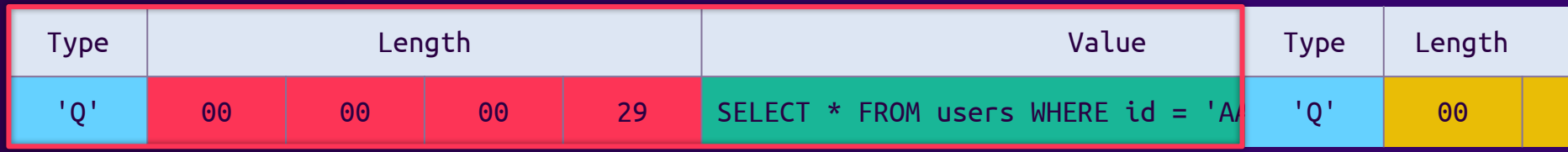


sonar

How does it look in the real world?

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

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



How to know this offset?

sonar

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?

sonar

Crafting a Payload: Magic Pattern

```
00000000:  5100 5100 5100 5100 5100 5100 5100 5100   Q.Q.Q.Q.Q.Q.Q.Q.
00000010:  5100 5100 5100 5100 5100 5100 5100 5100   Q.Q.Q.Q.Q.Q.Q.Q.
```

…

Crafting a Payload: Magic Pattern

```
00000000:  5100 5100 5100 5100 5100 5100 5100 5100   Q.Q.Q.Q.Q.Q.Q.Q.
00000010:  5100 5100 5100 5100 5100 5100 5100 5100   Q.Q.Q.Q.Q.Q.Q.Q.
```

…

sonar

Crafting a Payload: Magic Pattern

	T	Length	Value	
00000000:	51	00 51 00	51 00 51 00 51 00 51 00	Q.Q.Q.Q.Q.Q.Q.Q.
00000010:	51	00 51 00	51 00 51 00 51 00 51 00	Q.Q.Q.Q.Q.Q.Q.Q.
…	Q	0x510051	…	
	✅	✅	✅	

sonar

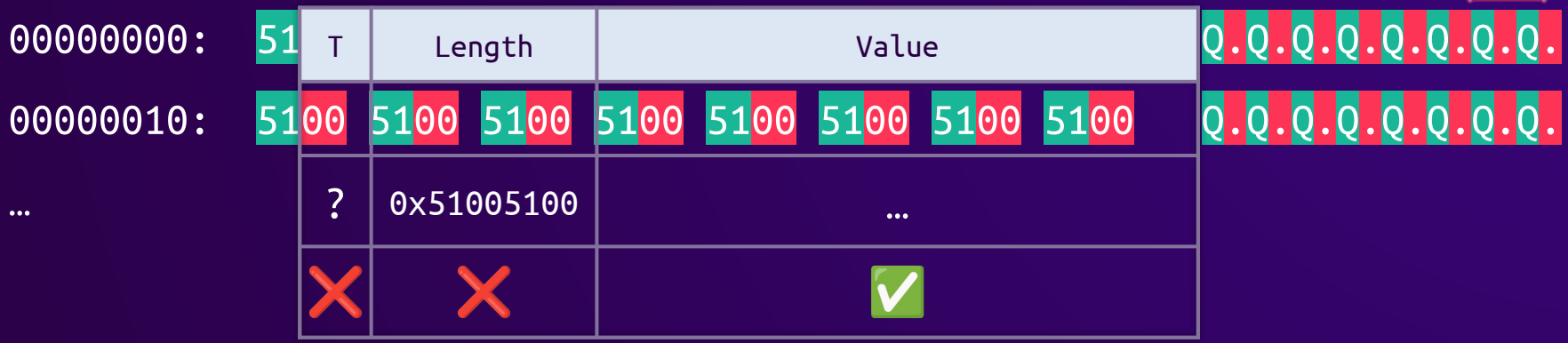
Crafting a Payload: Magic Pattern

```
00000000:  5100 5100 5100 5100 5100 5100 5100 5100  Q.Q.Q.Q.Q.Q.Q.Q.
00000010:  5100 5100 5100 5100 5100 5100 5100 5100  Q.Q.Q.Q.Q.Q.Q.Q.
```

…

sonar

Crafting a Payload: Magic Pattern



sonar

Crafting a Payload: Magic Pattern

```
00000000:  5100 5100 5100 5100 5100 5100 5100 5100   Q.Q.Q.Q.Q.Q.Q.Q.
00000010:  5100 5100 5100 5100 5100 5100 5100 5100   Q.Q.Q.Q.Q.Q.Q.Q.
```

…

- Success after ≤2 attempts!
  - 50% chance of success
  - Attack is repeatable, just change the offset

sonar

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	✅	❌	-

sonar

Exploitable Applications

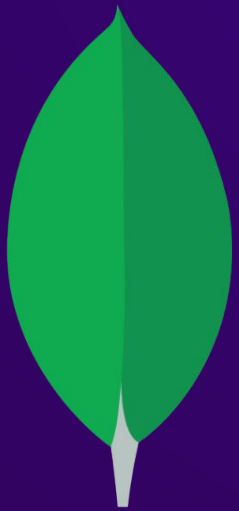
Demo: Harbor

- 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



sonar

MongoDB Wire Protocol

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

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

sonar

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?;
    // ...
}
```


sonar

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?;
    // ...
}
```

Get content bytes

sonar

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?;
    // ...
}
```

Calculate message size (usize)

sonar

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

sonar

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

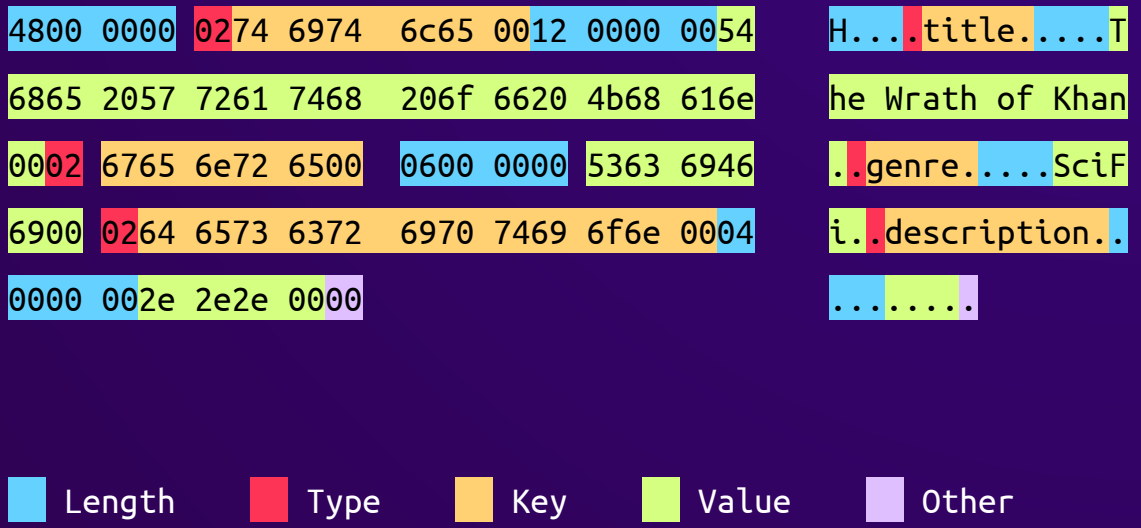
- Avoid bad bytes
  - Payload must be valid UTF-8
- Problem:
  - Message type (`dd 07`) is already invalid
  - Size fields can become invalid
- Solution:
  - Use metadata to create those bytes!

Crafting a Payload

Query:

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

BSON Document:



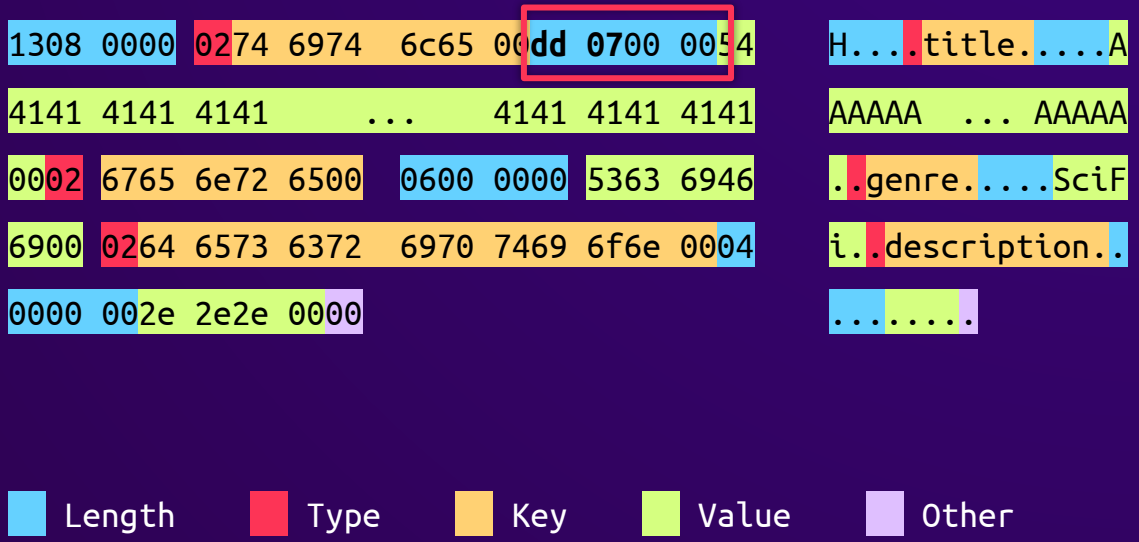
sonar

Crafting a Payload

Query:

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

BSON Document:



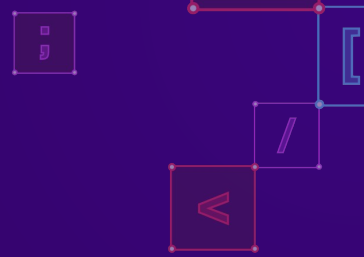
sonar

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



sonar

Constraints

How Web Apps Handle Large Payloads

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

sonar

How Web Apps Handle Large Payloads

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

sonar

How Web Apps Handle Large Payloads

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

- No default limits
- Disabled limits
  - Harbor

sonar

How Web Apps Handle Large Payloads

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

- Some enforce size limits

  before decompression
  - Nginx
  - Fastify

sonar

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

sonar

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	$> 2^{32}$	$> 2^{32}$

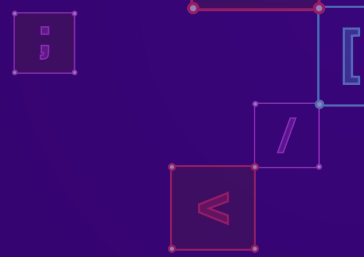
Only considering 64-bit versions.

* 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



sonar

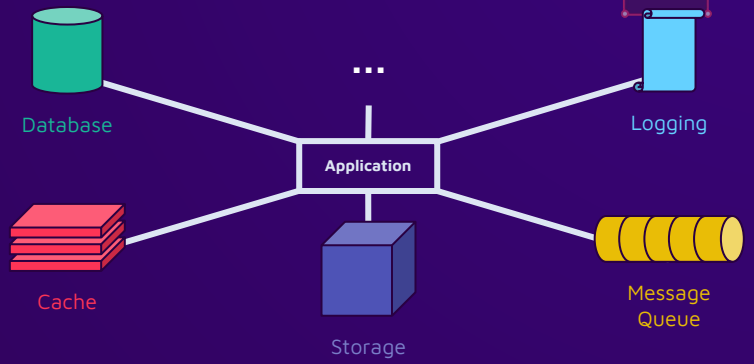
Safety First: No DoS Please!

Do not send large payloads to third-party systems!

sonar

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



sonar

Getting Started

https://archive.fluxfingers.net/2024/challenges/18.html

Play a hands-on challenge!

"FLX-Lock" from Hack.lu CTF 2024

```
        +----------------------------------------+      +~~~+~~~+~~~+
  ,----/  ~ FLX-Lock Keypad :: Debug Console ~  /       | 1 | 2 | 3 |
 /     +----------------------------------------+       +~~~+~~~+~~~+
+                                                       | 4 | 5 | 6 |
|       +----------------------------------------+      +~~~+~~~+~~~+
| ,----/  CMDs: //  status  /  unlock  /  quit  /       | 7 | 8 | 9 |
|/     +-------------------------------------+          +~~~+~~~+~~~+
+                                                       | * | 0 | # |
|                                                       +~~~+~~~+~~~+
+--> status
+--[ door is locked ]
|
+--> unlock
+--[ disabled in production build! ]
|
```


sonar

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!

sonar

Thank you!


@Sonar_Research


@SonarResearch@infosec.exchange


https://sonarsource.com


@pspaul95


@pspaul@infosec.exchange

sonar