

The CRIME attack



HTTPS:// Secure HTTP

HTTPS provides:

- Confidentiality (Encryption),
- Integrity (Message Authentication Code),
- Authenticity (Certificates)

CRIME decrypts HTTPS traffic to steal cookies and hijack sessions.



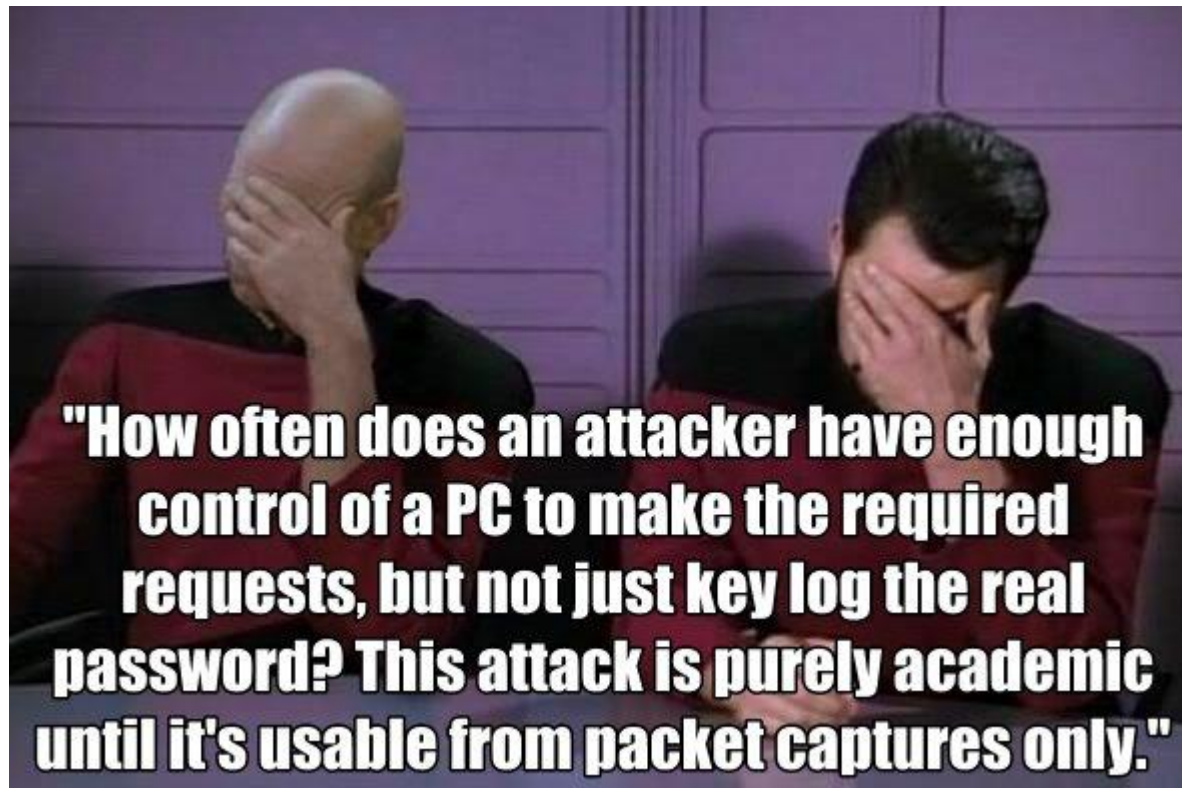
How can you become a victim of CRIME?

- 1st requirement: the attacker can sniff your network traffic.
 - You share a (W)LAN.
 - He's [hacked](#) your home router.
 - He's your network admin, ISP or government.

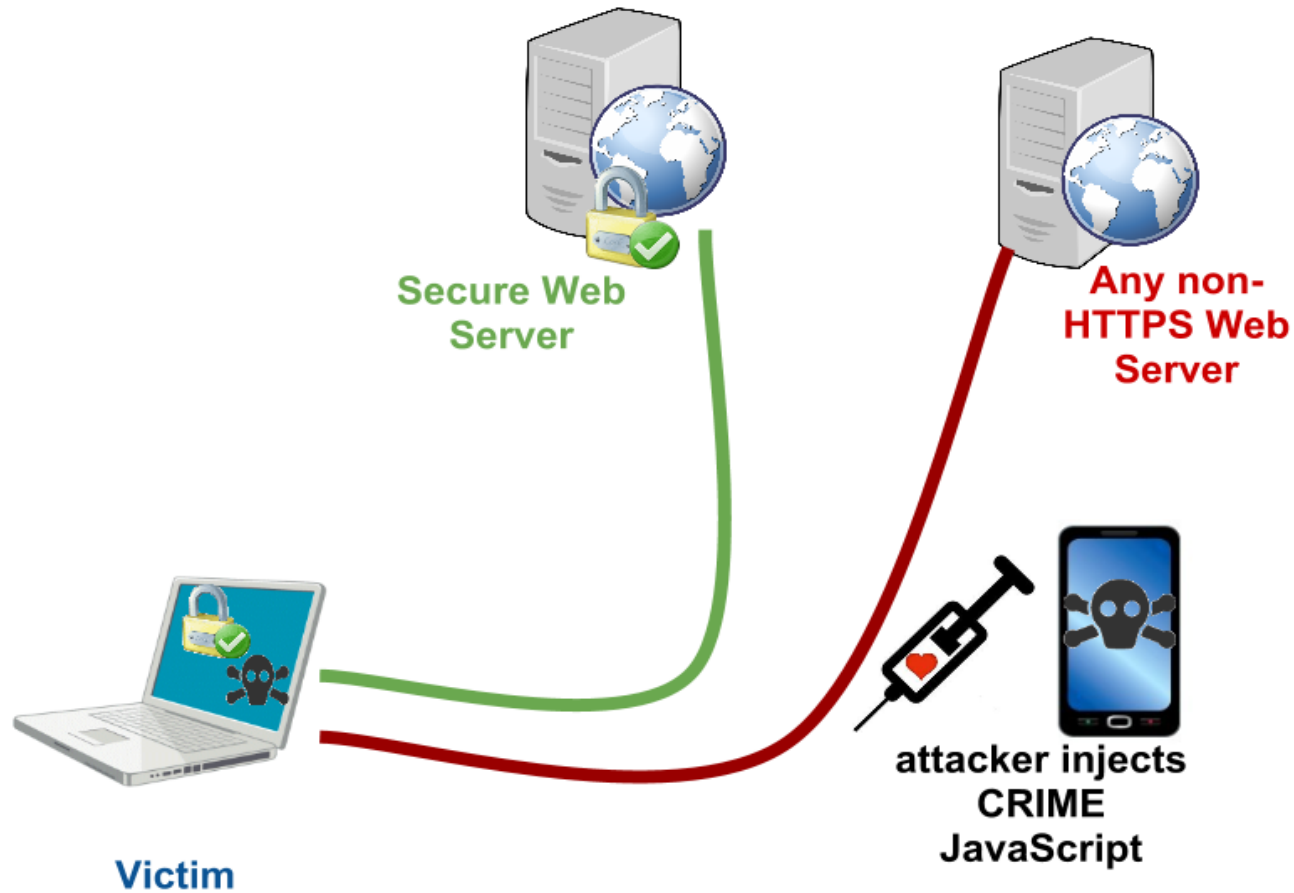


How can you become a victim of CRIME?

- 2nd requirement: you visit evil.com.
 - You click on a link.
 - Or you surf a non-HTTPS site.



CRIME injection



C in CRIME is compression

- Transmit or store the same amount of data in fewer bits.
- When you see compression in Internet protocols, it's probably DEFLATE.
- zlib and gzip are the two most popular DEFLATE wrappers.

Compression is everywhere

- TLS layer compression.
- Application layer compression
 - SPDY header compression,
 - HTTP response gzip compression,
 - Not so sure if exploitable: SSH, PPTP, OpenVPN, XMPP, IMAP, SMTP, etc.
- We will discuss TLS compression, SPDY and HTTP gzip.

DEFLATE

- Lossless compression reducing bits by removing redundancy.
- Best way to learn: RFC 1951 and puff.c.
- DEFLATE consists of two sub algorithms:
 - a. LZ77, and
 - b. Huffman coding.

DEFLATE: LZ77

- Google is so googley -> Google is so g(-13, 5)y
- It scans input, looks for repeated strings and replaces them with back-references to last occurrence as (distance, length).
- Most important parameter: window size.
 - How far does it go back to search for repetition?
 - Also called dictionary size.

DEFLATE: Huffman coding

- Replace common bytes with shorter codes.
- Build a table that maps each byte with a unique code.
 - Dynamic table: built based on the input, codes can be as short as 1 or 2 bits.
 - Fixed table: specified in the RFC, longer codes (7-9 bits), good for English or short input.



achievement unlocked

finally understand how compression works after all these years

R in CRIME is ratio

- How much redundancy the message has.
- More redundancy -> better compression ratio -> smaller request length.
- `len(compress(input + secret))`
 - input is attacker-controlled.
 - If it has some redundancy with secret, length will be smaller.
 - Idea: change input and measure length to guess secret.

I in CRIME is info-leak

- SSL/TLS doesn't hide request/response length.

The image shows a Wireshark packet capture of an SSL/TLS session. The top section displays a list of packets, with packet 83 selected. The details pane below shows the structure of the selected packet, which is a TLSv1 record containing application data. The data is encrypted, and the length of the record is 754 bytes. The bottom status bar indicates the length of the SSL record data is 754 bytes.

No.	Time	Source	Destination	Protocol	Length	Info
80	9.891491	199.59.150.39	192.168.0.172	TCP	54	59994 > https [ACK] Seq=35103
81	9.964145	199.59.150.39	192.168.0.172	TLSv1	754	Application Data
82	9.964217	192.168.0.172	199.59.150.39	TCP	54	59994 > https [ACK] Seq=2981
83	9.969836	199.59.150.39	192.168.0.172	TLSv1	754	Application Data
84	9.969870	192.168.0.172	199.59.150.39	TCP	54	59994 > https [ACK] Seq=2981
85	9.970168	199.59.150.39	192.168.0.172	TLSv1	754	Application Data
86	9.970183	192.168.0.172	199.59.150.39	TCP	54	59994 > https [ACK] Seq=2981
87	9.970519	199.59.150.39	192.168.0.172	TLSv1	754	Application Data

Transmission Control Protocol, Src Port: https (443), Dst Port: 59994 (59994), Seq: 35586, Ack: 2981, Len: 759

Secure Socket Layer

TLSv1 Record Layer: Application Data Protocol: http

Content Type: Application Data (23)

Version: TLS 1.0 (0x0301)

Length: 754

Encrypted Application Data: C67B0275849307B5A0B6E97B998341B6BA375E08123C830B...

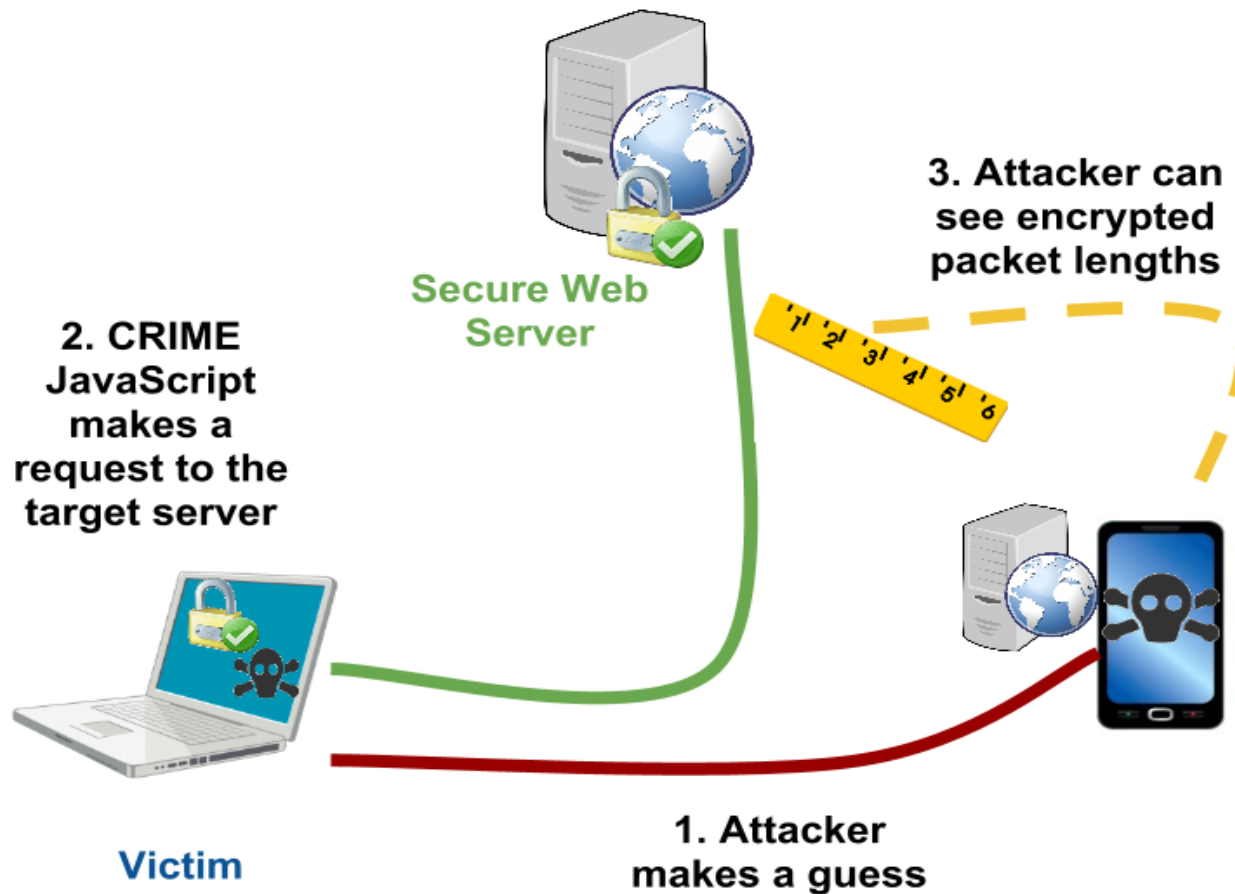
Length of SSL record data (s...): 754 Packets: 98 Displayed: 98 Marked: 0 Profile: Default

CRIME algorithm

- $\text{len}(\text{encrypt}(\text{compress}(\text{input} + \text{public} + \text{secret})))$ is *leaked*
 - input: URL path
 - public: known headers
 - secret: cookie
- Algorithm:
 - Make a guess, ask browser to send a request with path as guess.
 - Observe length of the request that was sent.
 - Correct guess is when length is *different* than usual.

```
GET /twid=a
Host: twitter.com
User-Agent: Chrome
Cookie: twid=secret
...
GET /twid=s
Host: twitter.com
User-Agent: Chrome
Cookie: twid=secret
```

CRIME in a slide



ME in CRIME is mass exploitation

- Worked for 45% of browsers: Chrome and Firefox.
- Worked for all SPDY servers: Gmail, Twitter, etc.
- Worked for 40% of SSL/TLS servers: Dropbox, GitHub, etc.

ME in CRIME is also made easy

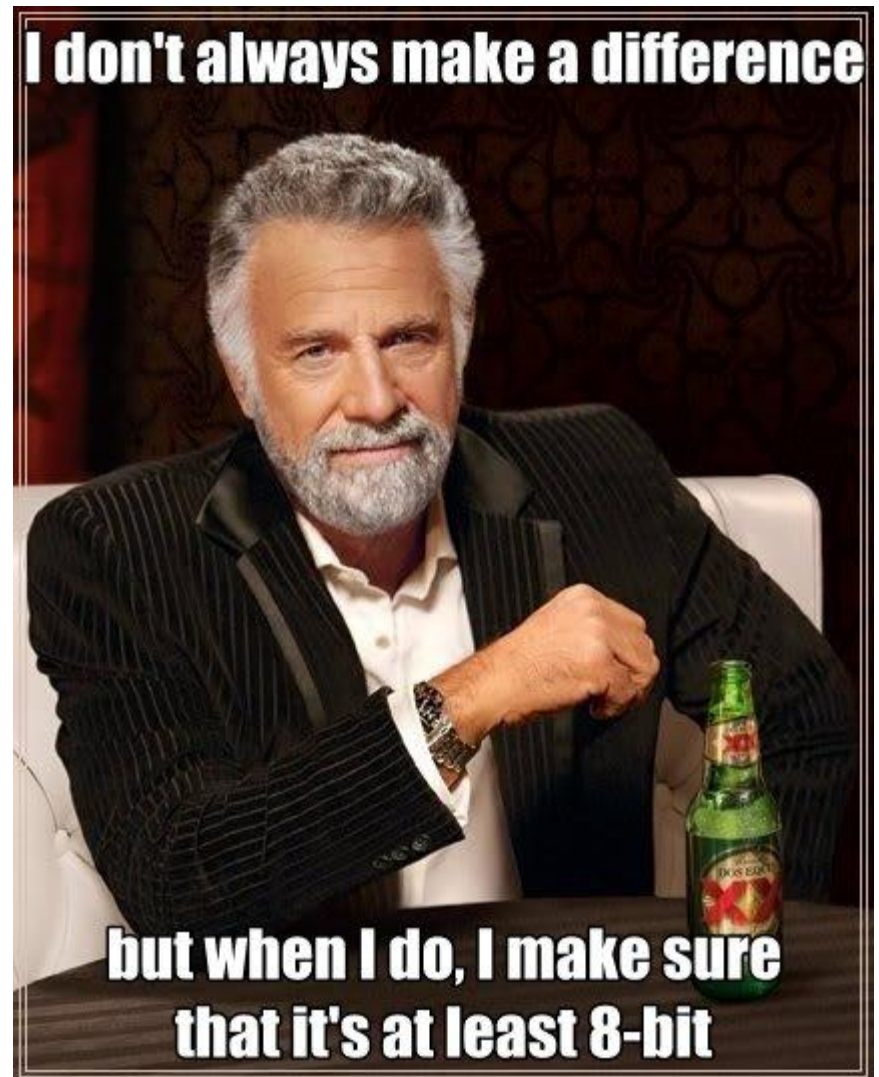
- JavaScript is optional.
- Fast Hollywood-style decryption. The best algorithm requires on average 6 requests to decrypt 1 cookie byte.
- Worked for all TLS versions and all ciphersuites (AES and RC4).
- Work even if HSTS is active and preloaded.

CRIME is the new BEAST

- BEAST opened the path to CRIME
 - Easy to perform chosen-plaintext attack against HTTPS.
 - Use URL path to decrypt cookie.
 - Move data across layer boundary.
- What's new?
 - SSL compressed record length info-leak, instead of CBC mode with chained IVs vulnerability.
 - New boundaries: compressor window size and TLS record size, instead of block cipher's block size.

So length is leaked

- Length is the number of bytes, but DEFLATE outputs bits.
- Length of request with a match must have a difference of at least 8 bits.
 - A 63-bit request looks exactly the same as a 59-bit on the wire.



First attack: Two Tries

- Recall window size: if the distance from the current string to the previous occurrence is greater than window size, it won't be replaced.
- Window size is essentially a data boundary. Let's move thing across it!
- For each guess, send two requests (hence Two Tries)
 - req1 with the guess inside the window of the cookie.
 - req2 is a permutation of req1, with the guess outside.

Two Tries: length difference

- If guess is incorrect:
 - guess won't be replaced by a reference to cookie in neither req1 nor req2.
 - hence, $\text{len}(\text{req1}) == \text{len}(\text{req2})$.
- If guess is correct:
 - guess will be replaced by a reference to cookie in req1.
 - guess *won't* be replaced in req2, because it's *outside* the window.
 - hence, $\text{len}(\text{req1}) \neq \text{len}(\text{req2})$.

Two Tries

- Oracle:
 - If `len(req1) != len(req2)`, then the guess is correct;
 - It's incorrect otherwise.

GET /**ABCDEF****twid=s**<padding>Cookie: twid=secret

GET /**twid=s****ABCDEF**<padding>Cookie: twid=secret

Two Tries

- Pros:

- Work for TLS compression, SPDY and HTTP gzip as well.
- False positive free with a few tricks.

- Cons

- Require $O(W)$ requests, where W is cookie charset.
- May fail when cookie contains repeated strings.
- Depend on deep understanding of DEFLATE and zlib's deflate.c to create a 8-bit difference.

SPDY

- A new open networking protocol for transporting web content.
- Similar to HTTP, with particular goals to reduce web page load latency and improve web security.
- SPDY achieves reduced latency through *compression*, multiplexing, and prioritization.

SPDY

- Standardized: selected by IETF as the starting point for HTTP 2.0.
- Servers: Google, Twitter, Wordpress, F5 Networks, Cloudflare, Apache httpd, nginx, etc.
- Clients: Chrome, Firefox, Opera (beta), etc.

Compression in SPDY

- DEFLATE is used to compress headers.
- SPDY uses the same compression context for all requests in one direction on a connection.
 - repeated strings in new requests can be replaced by references to old requests.

CRIME for SPDY

- The shared compression context is a two-edged sword
 - Better compression.
 - Subsequent compressed headers are so small that zlib decides to use *fixed* Huffman table.
- Recall that fixed Huffman table uses 7-9 bit codes. Hence, it's easier to have a difference of 8 bits.

CRIME for SPDY

1. Send a request to "reset" the compression context (i.e., prepare the dictionary).
2. Send another request with a wrong guess to get the base length.
3. For each guess, send a request. Use the base length to spot possible correct guesses.

CRIME for SPDY

GET /aatwid=a HTTP/1.1\r\n (-84, 5)aa(-20, 5)a(-84, 71)
Host: twitter.com\r\n
User-Agent: Chrome\r\n
Cookie: twid=secret\r\n

GET /bbtwid=b HTTP/1.1\r\n (-84, 5)bb(-20, 5)b(-84, 71)
Host: twitter.com\r\n
User-Agent: Chrome\r\n
Cookie: twid=secret\r\n

CRIME for SPDY

GET /**rr**twid=**r** HTTP/1.1\r\n (-84, 5)rr(-20, 5)r(-84, 71)
Host: twitter.com\r\n
User-Agent: Chrome\r\n
Cookie: twid=**secret**\r\n

GET /**ss**twid=**s** HTTP/1.1\r\n (-84, 5)ss(-20, 6)(-84, 71)
Host: twitter.com\r\n
User-Agent: Chrome\r\n
Cookie: twid=**secret**\r\n

CRIME for SPDY

- Pros

- Still $O(W)$, but with a smaller constant than Two Tries.
- Very fast, thanks to SPDY.
- Also false positive free.

- Cons

- Can't send many requests at a time if server sets a maximum limit.
- Different browsers have different implementations of SPDY header compression.

CRIME for SPDY

- Workaround
 - Chrome and Firefox have disabled header compression in their SPDY implementations.



- SPDY/4 will make CRIME irrelevant (hopefully).

Compression in TLS

- Specified in RFC 3749 (DEFLATE) and RFC 3943 (LZS).
- Chrome (NSS), OpenSSL, GnuTLS, etc. implement DEFLATE.
- If data is larger than maximum record size (16K), it split-then-compress each record independently (in a separate zlib context).

CRIME for TLS Compression: 16K-1

- 16K is essentially another boundary. BEAST's chosen-boundary attack strikes again!
- Make a request so big that it will be split into two records such that:
 - 1st record: GET /<padding>Cookie: twid=s
 - 2nd record: ecret
- Simulate the compression of the 1st record for every candidate.
- Send the request, obtain the compressed length of its 1st record. Use it to select possible correct bytes.

16K-1

16K { GET /AAAAAFILLTHERECORDAAAAA HTTP/1.1\r\n
Host: twitter.com\r\n
Cookie: twid=S . ° ○ }

16K { ecret\r\n }

16K-1 POC

```
def next_byte(cookie, known, alphabet=BASE64):
    candidates = list(alphabet)
    while len(candidates) != 1:
        url = random_16K_url(known)
        record_lens = query(url)
        length = record_lens[0]
        record = "GET /%s%s%s" (url, REQ, known)
        good = []
        for c in candidates:
            if len(compress(record + c)) == length:
                good.append(c)
        candidates = good
    return candidates[0]
```

CRIME for TLS Compression

- Pros

- Require only $O(\log W)$ requests. Can choose between longer offline compression or larger number of online requests.
- False positive free.
- Compression algorithm independent.

- Cons

- While server-side deployment is 40%, Chrome was the only browser that supported TLS compression.
- zlib versions on victim and attacker should be the same.

CRIME for TLS Compression

- Workaround
 - Chrome has disabled compression in its ClientHello.

TLS Compression

2004 - 2012

HTTP response gzip compression

- The most popular compression on the Internet.
- Just a wrapper on top of DEFLATE.
- CRIME works like a charm.

CRIME for HTTP gzip

- Requirement: server echoes back some client input in the response (e.g., /search?q=crimeN0tF0uddd).
- Use the echoed input to extract PII or XSRF token embedded in the response.
- Two Tries may work, but we haven't tested it yet.

"We believe"

- TLS compression may resurrect in the near future
 - "Browsers are not the only TLS clients!"
- HTTP gzip may be a bigger problem than both SPDY and TLS compression
 - If you control the network, then a XSRF token is as good as, if not better, a session cookie.
- Remember: compression is *everywhere*.

Thanks

- Google, Mozilla, and Dropbox.
- Dan Boneh, Agustin Gianni, Kenny Paterson, Marsh Ray, Eduardo Vela and many other friends.
- EKOPARTY xD xD xD!!

Related work

- *John Kelsey*, Compression and Information Leakage of Plaintext.
- *Adam Langley*, [post](#) to SPDY mailing list.

Questions?

<https://twitter.com/julianor> or thaidn@gmail.com