



# Network-based Origin Confusion Attacks against HTTPS Virtual Hosting

**Antoine Delignat-Lavaud**, Karthikeyan Bhargavan  
Prosecco, Inria Paris-Rocquencourt

# The Web Security Protocol Stack

## JavaScript runtime

(scripts, content scripts, DOM, WebWorkers, ASM.js, ...)

## HTML5 / Browser environment

(same origin policy, frames, windows, postMessage, CSP, ...)

## HTTP Protocol

(Redirections, cookies, HSTS, Origin header, Virtual Hosting, ...)

## PKIX / X.509

(certificates, certification authorities, OCSP, certificate transparency, ...)

## Transport Layer Security (TLS/SSL)

(SNI, session resumption, Channel ID, ALPN, ...)

# Our Previous Efforts

- miTLS: verification of the TLS protocol based on a **reference implementation with refinement types**
- **Positive results** [S&P'13, CRYPTO'14]: precise (but complex) statement of TLS API's security goals
- **Attacks** [S&P'14, S&P'15]: Triple Handshake, SMACK, FREAK...

# Current Challenge

- How do the **low-level cryptographic guarantees** of TLS translate to **high-level Web security guarantees**?
- Moving up the protocol stack, things quickly get messy (e.g. secure cookies, HSTS... break layering abstraction)
- Cross-protocol attacks (e.g. Cookie Cutter [S&P'14])

# In this paper

We present a new class of attacks against the deployment of TLS on the Web that allow a **network adversary** to bypass the cross-origin isolation between domains that **share some transport-layer credentials**.

# HTTP Reminder

GET `/p/ath?k=v` HTTP/1.1

Host: `www.x.com:8080`

Cookie: User=Alice

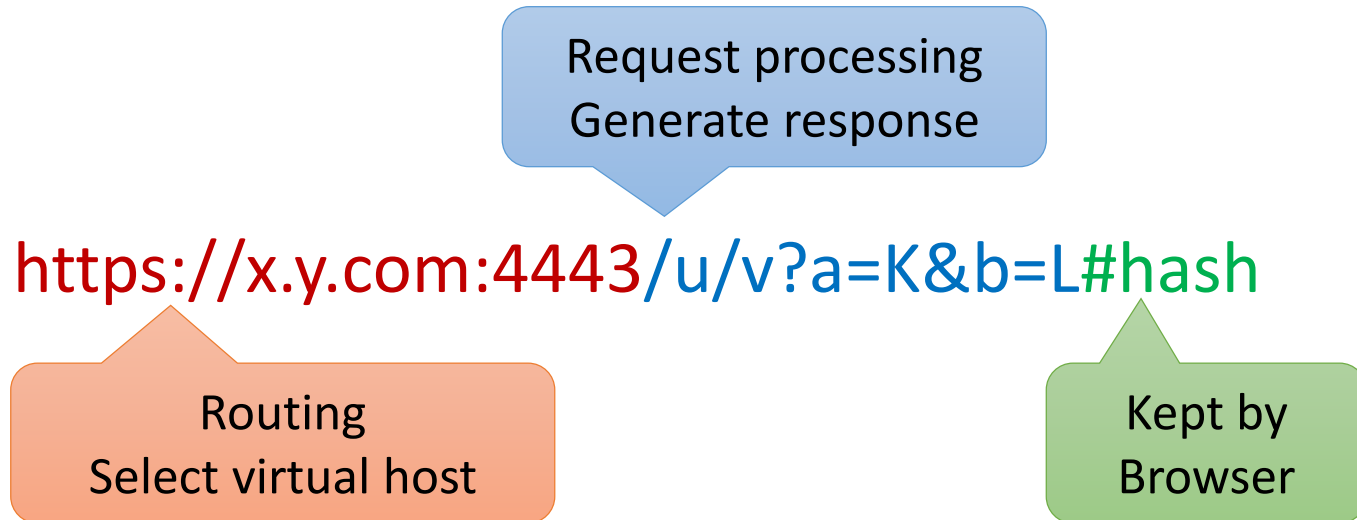
HTTP/1.1 200 OK

Content-Type: text/plain

Content-Length: 9

Hi Alice!

# HTTP Virtual Hosting



# HTTP Reminder

POST `/login` HTTP/1.1

Host: `login.x.com:444`

Origin: `https://y.com`

Content-Length: 20

User=Alice&Pass=1234

HTTP/1.1 302 Redirect

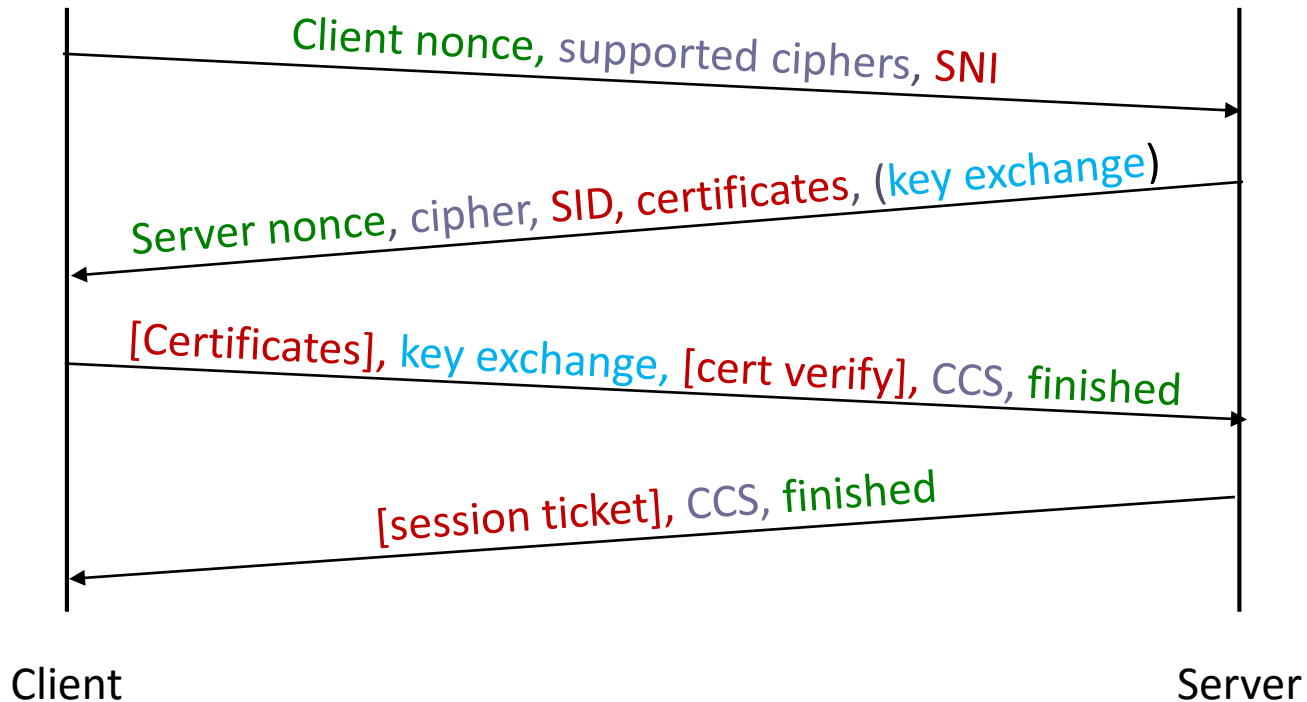
Location: /

Set-Cookie: SID=XXX;  
domain=`.x.com`;  
`secure`; httpOnly

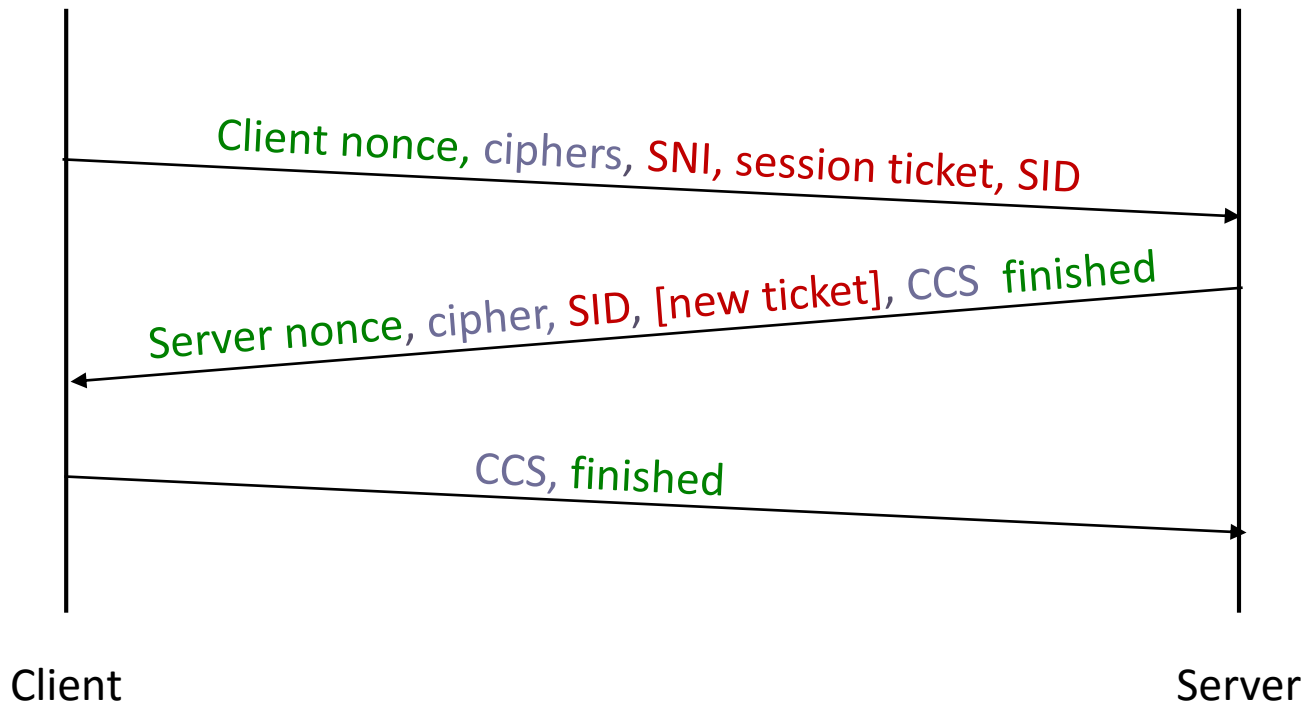
Strict-Transport-Security:  
max-age=ZZZ;  
`includeSubDomains`



# HTTPS Reminder



# HTTPS Reminder



# TLS vs HTTP Identity

- Transport layer

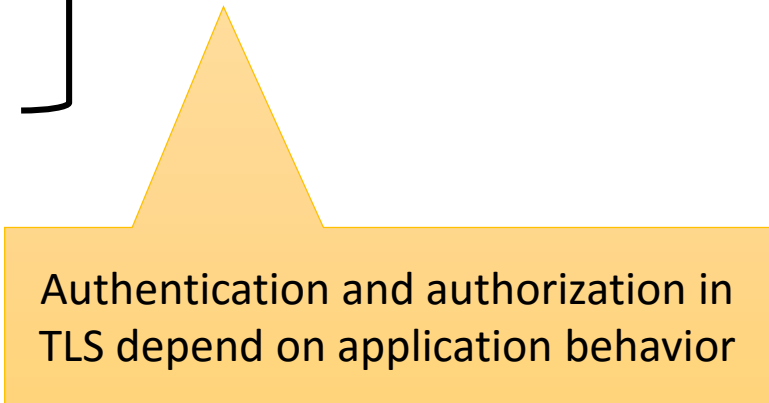
- Server Name Indication (SNI)
- Certificate (union of CN and SAN)
- Session identifier
- Session ticket, Channel ID



Managed by application!

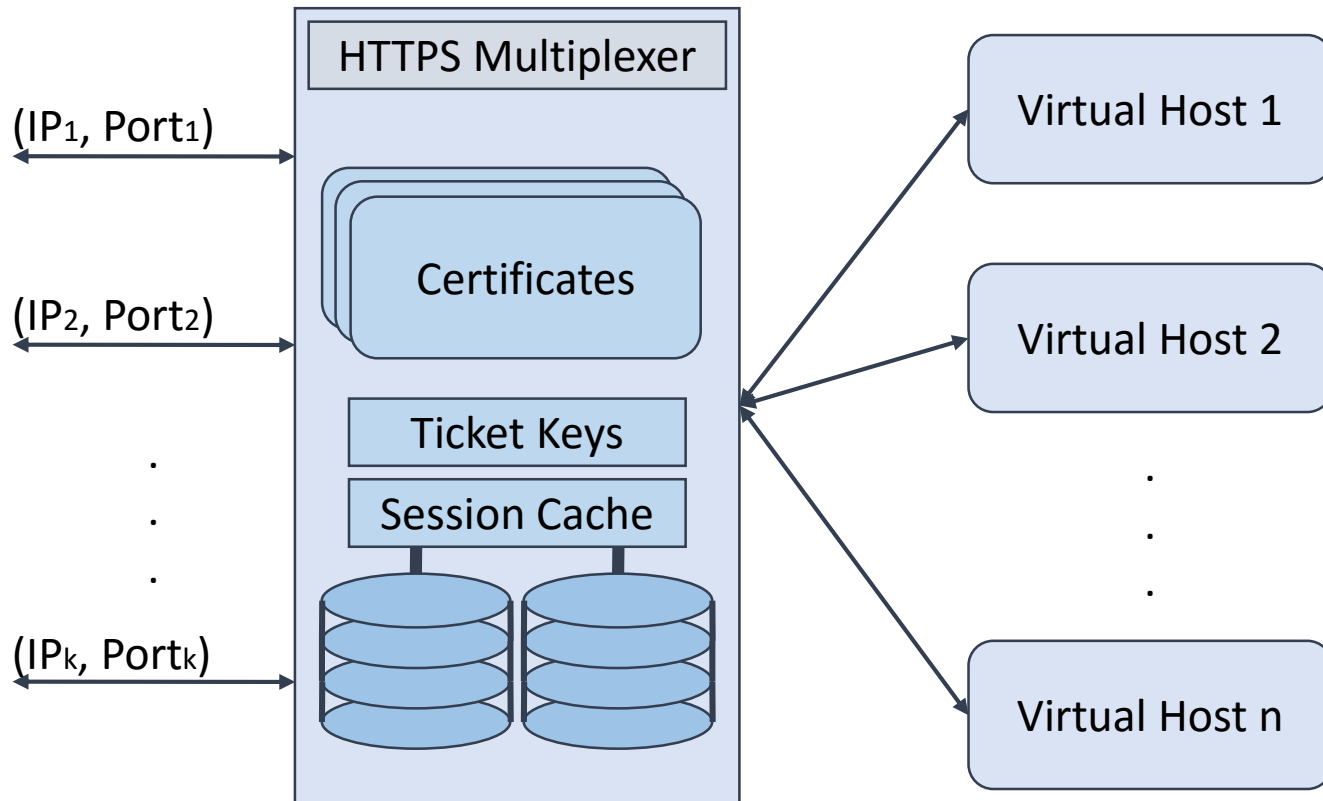
- Application layer

- Host header [Origin header]



Authentication and authorization in TLS depend on application behavior

# HTTPS Multiplexing



# Virtual Hosting in Practice

```
ssl_session_ticket_key "/etc/ssl/ticket.key";
ssl_session_cache shared:SSL:1m;

server { #1
    listen 1.2.3.4:443 ssl;
    server_name www.a.com;
    ssl_certificate "/etc/ssl/a.pem";
    root "/srv/a";
}
server { #2
    listen 4.3.2.1:443 ssl;
    server_name ~^(?<sub>api|dev)\.a\.com$;
    ssl_certificate "/etc/ssl/a.pem";
    root "/srv/api";
}
server { #3
    listen 2.1.4.3:443 ssl;
    server_name www.learn-a.com;
    ssl_certificate "/etc/ssl/learn-a.pem";
    root "/srv/learn";
}
```

# Request routing

- (IP, port) of request = (IP, port) of chosen host
- TLS settings picked from host whose name matches SNI, or default (**fallback**)
- Request is routed to host whose name matches Host header, or default (**fallback**)

# Virtual Host Confusion

- Current routing algorithm do not guarantee the selected virtual host was **intended** to process the request
- In many cases, a network attacker can redirect a request to an unintended server without causing a TLS error
  - Known vector [C. Jackson et al., CCS'07]

# Virtual Host Confusion

- Two servers for the same domain on different ports (a.com:443 & a.com:4443)
  - Port always ignored in Host header.
  - Attacker can redirect freely between ports
  - **Port is essentially useless for same-origin policy**



# Virtual Host Confusion

- One certificate {x.a.com, y.a.com} (or \*.a.com)
- Server at IP X only handles x.a.com
- Server at IP Y only handles y.a.com
  - Attacker can redirect packets from X to Y
  - Server at Y returns a page from y in x.a.com origin

# Exploiting Virtual Host Confusion

A network attacker can **transfer HTTP weaknesses and vulnerabilities** (e.g. XSS, user contents, open redirectors, cross-protocol redirections, X-Frame-Options, CORS, ...) across **origins that are related at the transport layer**.

# Is this really a concern in practice?

- Cloud hosting: **unrelated, mutually distrusting domains** are served through the **same servers**
- Multi-domain (SAN) & wildcard certificates are extremely common and often include domains with **different HTTP security characteristics**
- Some content delivery networks (CDN) use **shared certificates** for customer domains (e.g. Cloudflare)

# Concrete Attacks

Watch exploit videos on <https://bh.ht.vc>

# Impersonation through Akamai

- The default Akamai virtual host is an open proxy:  
<https://a248.e.akamai.net/6/6/6/attacker.com/query>
- Routing fallback to an open proxy allows impersonation

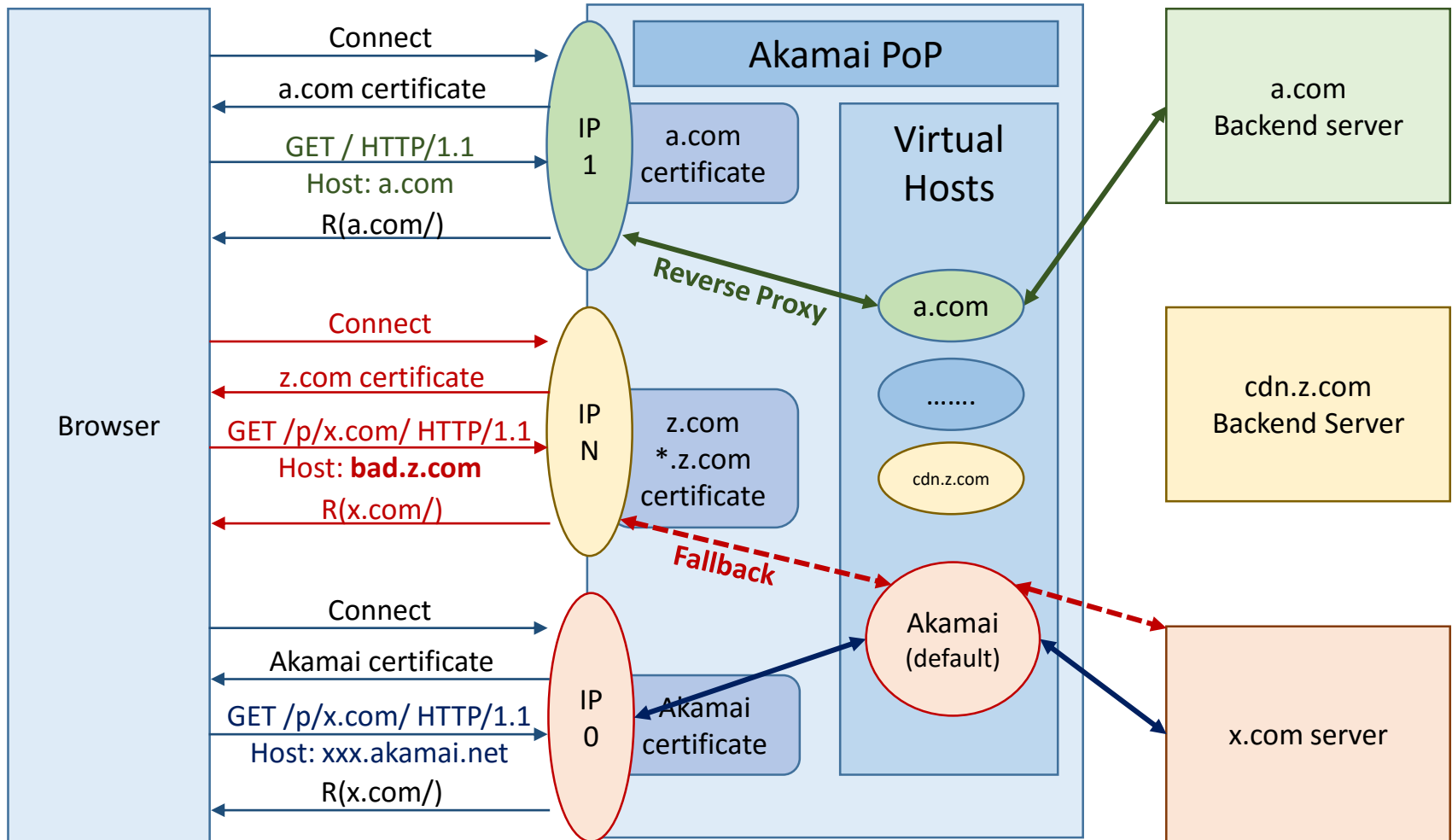


**Hello, I am the attacker.**

**Here are the cookies you sent me:**

```
array(4) {  
  ["__utma"]=>  
  string(54) "176212410.819947620.1386092553.1386092553.1386096268.2"  
  ["__utmc"]=>  
  string(9) "176212410"  
  ["__utmz"]=>  
  string(70) "176212410.1386092553.1.1.utmcsr=(direct)|utmccn=(direct)  
  ["__utmb"]=>  
  string(25) "176212410.2.10.1386096268"  
}
```

# Impersonation through Akamai



# Impersonation through Akamai

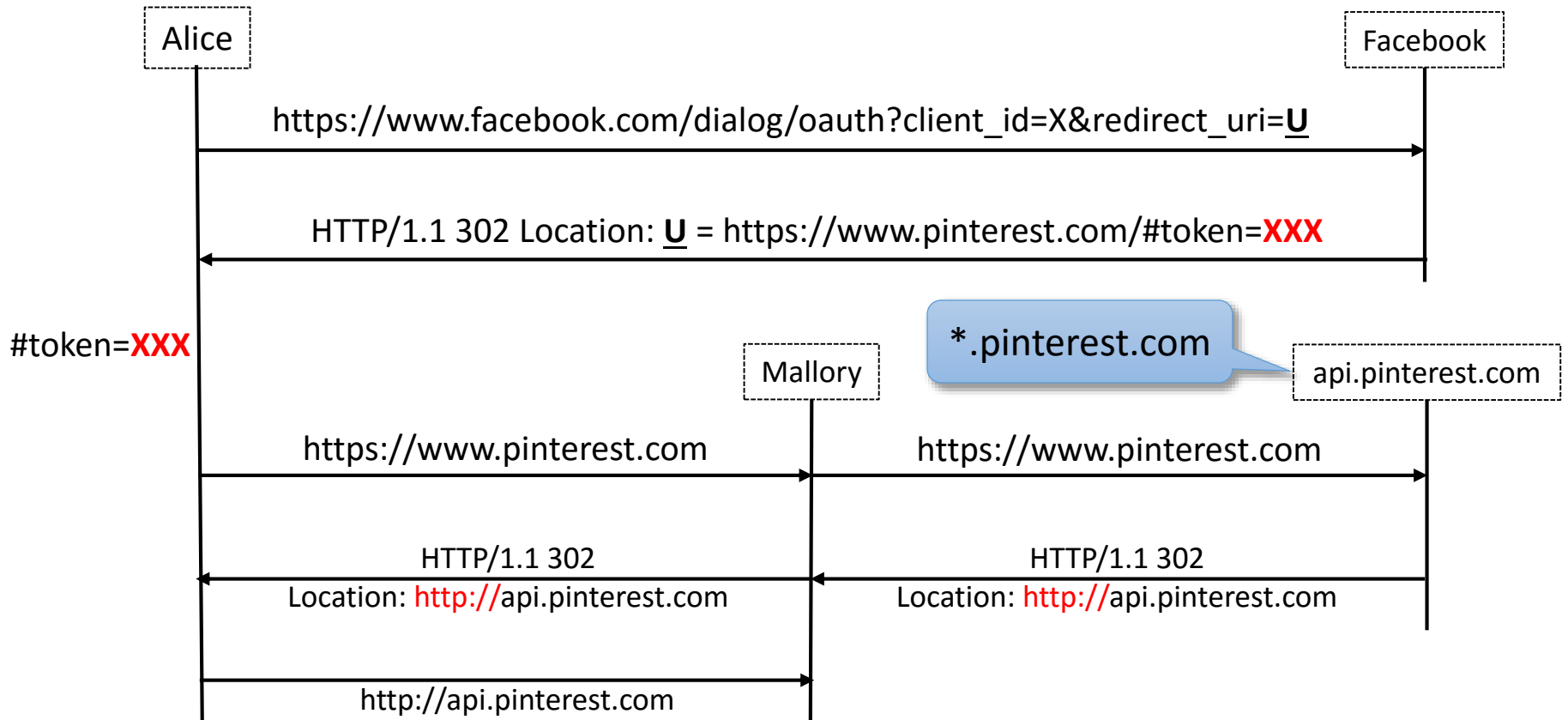
- Attack was present for 15+ years
- At least 12,000 domains affected including 7 out of top 10 Alexa websites in the US
- Quickly fixed following our report

# SSO & Cross-Protocol Redirections

- OAuth's `redirect_uri` access control is origin based
- If the token origin can be confused with **any origin with a redirect-to-HTTP**, attacker can steal token
  - Token is in *URL fragment* (preserved by redirection): attacker can inject script in HTTP response to steal it
- Cross-protocol redirection **should be avoided**
  - **Attack built into Google:** `nosslsearch.google.com`



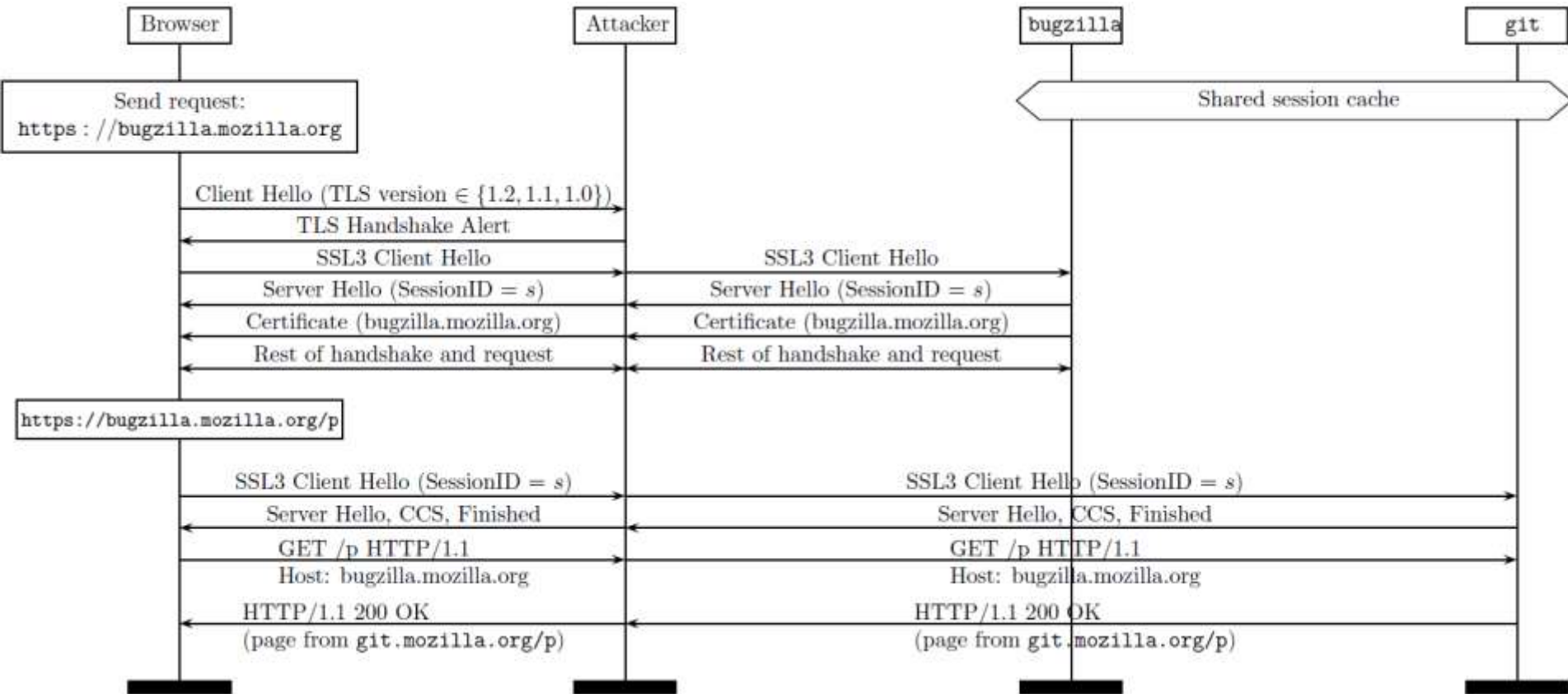
# Attack: Pinterest User Impersonation



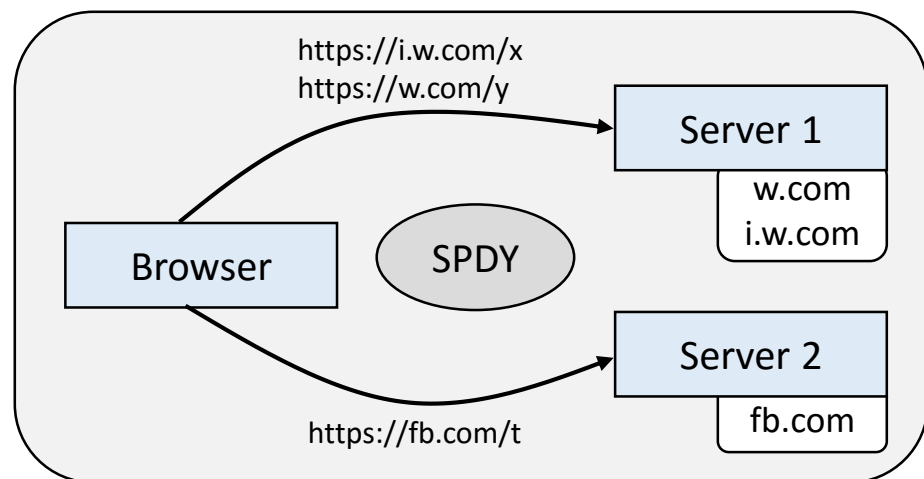
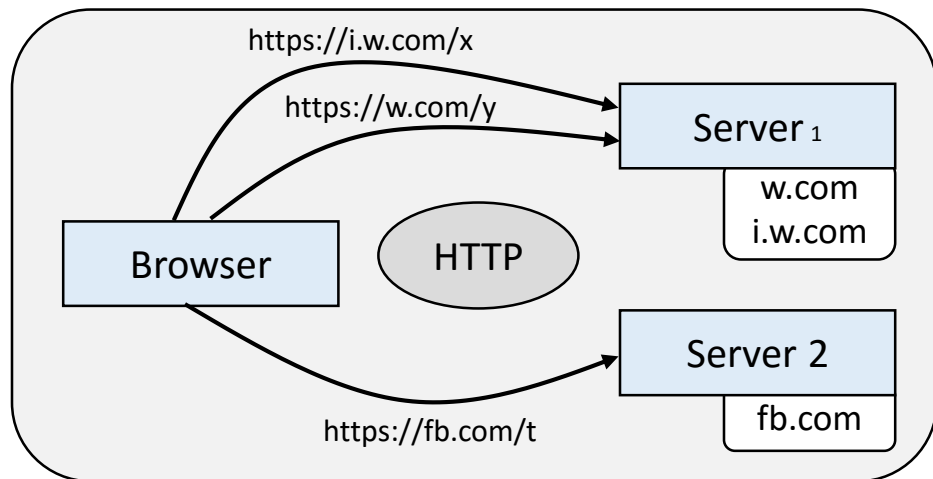
# TLS Session Cache

- 3 kinds of TLS authentication:
  - Certificate/PKIX
  - Valid session identifier in server cache
  - Valid session ticket encrypted by server key
- If a session cache or ticket key is shared across servers with different hosts, **certificate check can be completely bypassed**
  - VHC between domains that do not share certificates

# Cross-certificate XSS against Mozilla



# SPDY/HTTP2 Connection Sharing



OK to reuse an existing connection to send a request if the **domain of the new request is covered by the certificate validated when the session was created**, and it points to the IP address of the peer of the existing connection

# Attack: SPDY Pooling Impersonation

- Attacker baits user to **click through a certificate warning** on an **unimportant domain** (e.g. captive portal on public Wifi)
- Unbeknown to the user, the certificate includes malicious SAN (e.g. \*.google.com, \*.facebook.com, ...)
- The attacker point the DNS of these domains his down IP to enable SPDY connection pooling, and triggers requests to collect cookies
  - Bypasses all TLS MitM defenses in Chrome (pinning...)

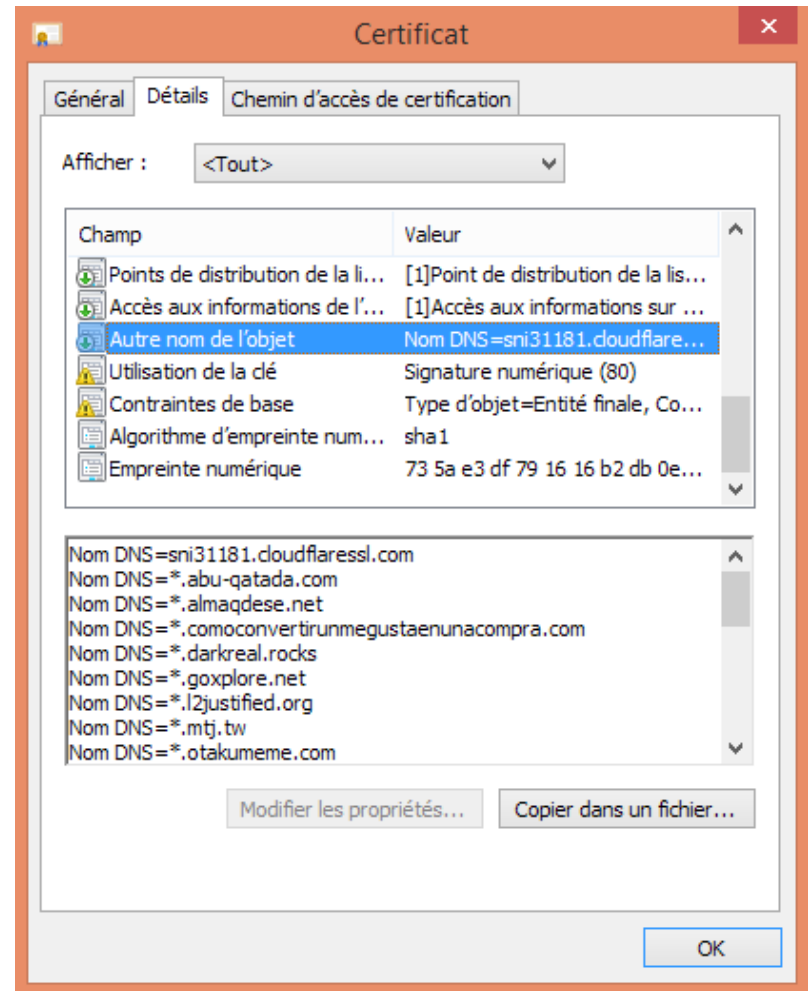
# Attack: SPDY Pooling Impersonation

The image shows two browser windows. The top window is titled "SSL Error" and shows the address bar with "https://localhost". The main content area is a yellow warning box with a white background. It contains a yellow warning triangle icon with an exclamation mark. The text reads: "The site's security certificate is not trusted!". Below this, it says: "You attempted to reach localhost, but the server presented a certificate issued by an entity that is not trusted by your computer's operating system. This may mean that the server has generated its own security credentials, which Chrome cannot rely on for identity information, or an attacker may be trying to intercept your communications." It then states: "You should not proceed, especially if you have never seen this warning before for this site." At the bottom of the warning box are two buttons: "Proceed anyway" and "Back to safety", and a link: "Help me understand".

The bottom window is titled "accounts.google.c" and shows the address bar with "https://accounts.google.com". The main content area is white and contains the text: "This should be impossible because of certificate pinning in Chrome".

# “Same-Certificate Policy” & HTTP2

- What does it mean security-wise when **domains appear in the same certificate?**
- Dangerous **new threat model**: attacker may inject himself **within the connection** used for honest requests



# Impact of this Paper

- Fixed impersonation of Akamai CDN customers
- Improved TLS cache isolation in popular HTTPS multiplexers: Nginx (CVE- 2014-3616), Stingray
- Fixed SPDY sharing oops in Chrome (CVE-2013-6659)
- Over \$10,000 worth of bug bounties
- **Routing fallback remains unsolved**
- **Connection sharing concerns remain in HTTP2**



# Thank you! Questions?

**For further discussions, please attend the Web Security Architecture panel on the W3C track!**

**Thursday 14:00 in Sala della Scherma**



# User Contents & Other Weak Origins

- Common to use different top-level domain for user contents to avoid related-domain attacks (e.g. cookies)
  - dropboxusercontent.com, googleusercontent.com
- May be defeated by virtual host confusion if the weak & strong origins are in the same certificate
  - Transfer XSS in mxr.mozilla.org to addons.mozilla.org (Hansen & Sokol, HTTPS Can Byte Me, BH'10)
- **Weakest origins should use separate certificates**

# Attack: Dropbox Account Theft

- Data **on the user's own account** is often on a higher trust domain to access session cookie
  - Dropbox: own files on dl-web.dropbox.com
  - Certificate valid for \*.dropbox.com
- **Short lived cookie forcing** allows temporary forcing of the attacker's session
- Abuse VHC between dl-web and www subdomains to load malicious page under www.dropbox.com
- Recover victim session after forced cookie expires