The BEAST Wins Again: Why TLS Keeps Failing to Protect HTTP

Antoine Delignat-Lavaud, Inria Paris
Joint work with K. Bhargavan, C. Fournet, A. Pionti, P.-Y. Strub
INTRODUCTION

- **Introduction**
- Cookie Cutter
- Virtual Host Confusion
  - Crossing Origin Boundaries
  - Shared Reverse Proxies
- Triple Handshake
- Conclusion

- Shared Session Cache
- SPDY Connection Pooling
Why do we need TLS?

1. **Authentication**
   - Must be talking to the right guy

2. **Integrity**
   - Our messages cannot be tampered

3. **Confidentiality**
   - Messages are only legible to participants

4. **Privacy?**
   - Can’t tell who we are and what we talk about
1. **Authentication**
   - Must be talking to the right guy

2. **Integrity**
   - Our messages cannot be tampered

3. **Confidentiality**
   - Messages are only legible to participants

4. **Privacy?**
   - Can’t tell who we are and what we talk about

---

**Why do we need TLS?**

- **Active Attacks** (MitM)
- **Passive Attacks** (Wiretapping)
What websites expect of TLS

• **Web attacker**
  – Controls malicious websites
  – User visits honest and malicious sites in parallel
  – Web/MitB attacks: CSRF, XSS, Redirection...

• **Network attacker**
  – Captures (passive) and tampers (active) packets
What websites expect of TLS

- **Web attacker**
  - Controls malicious websites
  - User visits honest and malicious sites in parallel
  - Web/MitB attacks: CSRF, XSS, Redirection...

- **Network attacker**
  - Captures (passive) and tampers (active) packets

Strictly stronger
If a website W served over HTTP is secure against a Web attacker, then serving W over HTTPS makes it secure against a network attacker.
What websites expect of TLS

If a website W served over HTTP is secure against a Web attacker, then serving W over HTTPS makes it secure against a network attacker.
• TLS optional by default in HTTP
• Cookies helplessly broken
• TLS adds own identity and session systems
  – May not agree with the HTTP ones
• HTTPS MITM is a beast
  – Arbitrary requests, run JS, side channels...
Not in this talk

• Heartbleed, GnuTLS SID corruption
  – No excuse for memory corruption bugs
• “Goto fail”, GnuTLS SA-2014-2, CCS bug
  – No excuse for bad implementation of protocol
• Broken PKI (ANSSI, Indian CCA)
  – Can’t be helped, but improving overall
In this talk

• Active network attacks against HTTPS
  – Public networks
  – DNS attacks
  – Corporate/ISP proxies
  – Governments

• TLS exploits enabled by HTTP capabilities
• Active network attacks against HTTPS
  – Public networks
  – DNS attacks
  – Corporate/ISP proxies
  – Governments
• TLS exploits enabled by HTTP capabilities
• Active network attacks against HTTPS
  – Public networks
  – DNS attacks
  – Corporate/ISP proxies
  – Governments

• TLS exploits enabled by HTTP capabilities

In this talk

Beastly Attacks

Only useful against strongest websites (Google, Facebook, Twitter, Amazon...)

black hat USA 2014
• Renegotiation attack [Ray, Rex ‘09]
  – Protocol logic flaw; nice cookie exploit
• BEAST [Rizzo, Duong ‘11]
  – Adaptive chosen plaintext + block boundary
  – Exploits known IV vulnerability
  – Can recover encrypted data
Beastly Attacks

• CRIME/BREACH [Rizzo Duong ’12; Prado et al ‘13]
  – Adaptive chosen plaintext + Length side channel
  – Timing variant TIME [Be’ery, Shulman ‘13]

• Padding Oracle [Vaudenay ‘02]
  – Timing variant Lucky13 [Al Fardan, Paterson et al. ‘13]

• More timing attacks are likely
COOKIE CUTTER
CANCEL HSTS AND STEAL SECURE SESSION COOKIES

- Introduction
  - Cookie Cutter
  - Virtual Host Confusion
    - Crossing Origin Boundaries
    - Shared Reverse Proxies
  - Triple Handshake
  - Conclusion
- Shared Session Cache
- SPDY Connection Pooling
Reminder: HTTPS is optional

- Attack: SSL stripping [Marlinspike, BH’09]
  - Attacker proxies HTTP requests to HTTPS server
- Defences:
  - Strict Transport Security (HSTS)
  - HTTPS Everywhere and similar extensions
  - User awareness
Reminder: HTTPS and cookies

• Shared HTTP/HTTPS cookie store
• Cookies don’t follow SOP
  – No port; non-public DNS suffix of domain
• ‘secure’ flag: don’t send over HTTP
• Server can’t tell if set over HTTP or HTTPS
“HTTPS is insufficient to prevent a network attacker from obtaining or altering a victim's cookies [...]; by default, cookies do not provide confidentiality or integrity from network attackers, even when used in conjunction with HTTPS.”

Adam Barth, RFC 6265
Impact has increased in modern applications
  – Asynchronous actions (AJAX)
  – No user feedback to session replacement
  – User data sent to attacker account
• Defeats many CSRF protections too
  – The deputies are still confused, Lundeen, BHEU’13
Defending against cookie forcing

• Do not use cookies
• Use HSTS (not HTTPS Everywhere)
  – With includeSubDomains option
  – On top-level domain of website
  – Do not use any subdomain (unless sent to top once)
• Bind cookie to TLS channel (Chrome: Channel ID)
Alice

http://docs.google.com/A

http://docs.google.com/A?XXXXX

Mallory


POST /login HTTP/1.1 [...] user=alice&password=123456&goto=...

Google

HTTP/1.1 302 Redirect [...] Location: http://doc.google.com/A?XXXXX
Set-Cookie: SID=beefcafe1337; domain=.google.com ; secure; httpOnly;
Connection: Keep-Alive

You are being redirected to doc.google.com ...
• TLS weakness: **truncation** [Wagner, WEC’96]
  – TLS (close_notify alert) vs TCP (RSET) termination
  – Well known (Pironti, BH’13)
• HTTP weaknesses
  – Plaintext injection (e.g. semi-open redirector)
  – Security depending on **presence** of header/flag
  – Liberal parsing of malformed HTTP messages
• If browser accepts the truncated cookie, it is stored **without the secure flag**

• *Need an HTTP request to sniff cookie*

• What about HSTS?
  – Strict-Transport-Security: max-age=10000; incl...
  – Truncate max-age to get rid of HSTS in <10s
• Reject malformed HTTP messages / headers
• Enforce close_notify (chunked encoding?)
• Chromium: CVE-2013-2853
• Safari: APPLE-SA-2014-04-22-1
• IE and FF correctly reject truncated headers
VIRTUAL HOST CONFUSION
BREAK SAME ORIGIN POLICY AND CERTIFICATE VALIDATION

- Introduction
- Cookie Cutter
- Virtual Host Confusion
  - Crossing Origin Boundaries
  - Shared Reverse Proxies
- Triple Handshake
- Conclusion
- Shared Session Cache
- SPDY Connection Pooling
Public Key Infrastructure (PKI)

Certification path

- VeriSign
- VeriSign Class 3 Extended Validation SSL SGC CA
- login.live.com

- Endpoint certificate
- Intermediate CA certificate
- Root Certification Authority certificate
Are certificates checked properly?

- GnuTLS: check_if_ca (2014)
- NSS (and others): null byte in CN (BH 2009)
- ...
- Path length, key usage, signature, revocation...
Can CAs be trusted?

With M. Abadi, A. Birrell, I. Mironov, T. Wobber and Y. Xie (NDSS’14)
PKI madness

  - Marlinspike, Sotirov, Jarmoc, Hansen...
- Academic papers (see e.g. Clark et al. survey)
- Certificate Transparency, DANE, TACK, Perspectives, Convergence, ...
Background: HTTPS multiplexing

- Virtual Host 1
- Virtual Host 2
- Virtual Host n

- Certificates
- Ticket Keys
- Session Cache

(IP₁, Port₁)
(IP₂, Port₂)
(IPₖ, Portₖ)
Background: HTTPS multiplexing

https://x.y.com:4443/u/v?a=K&b=L#hash
Background: TLS handshake

Client nonce, supported ciphers, (SNI)

Server nonce, cipher, [SID], certificates, (key exchange)

Certificates, key exchange, cert verify, CCS, finished

[Session Ticket], CCS, finished
Background: TLS resumption

Client

Client nonce, ciphers, (SNI, ticket), SID

Server nonce, cipher, [SID], (New ticket), CCS, finished

CCS, finished, data

Server
TLS vs HTTP identity

• Transport layer
  – Server Name Indication (SNI)
  – Certificate (union of CN and SAN)
  – Session identifier
  – Session Ticket

• Application layer
  – Host header
Virtual host configuration

- IP address and port
- Name (for SNI and Host header)
- Certificate
- Session cache, session ticket key
- Ciphers, client authentication, OCSP staple...
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

• **Fallback**: no guarantee selected host was intended to handle the request:
  – Could be meant for **different port**
  – Could be meant for **different IP address** that shares the **same certificate** (or overlapping one), **session database** or **ticket encryption key**

• Known vector [Jackson, CCS’07]
Two TLS servers on the same domain but on different ports

- Port always ignored in Host header.
- Attacker can redirect freely between ports
- Port is essentially useless for same-origin policy
Simple Examples

• 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
Host confusion ingredients

• TLS weaknesses
  – Resumption authenticates nothing (not even SNI)
  – Downgrade to SSL3 to get rid of SNI and ticket
  – Multi-domain and wildcard certificates

• HTTP weakness
  – Virtual host fallback: a request for x.com should not return a page meant to be served on y.com
Virtual host confusion can **transfer weaknesses and vulnerabilities** (e.g. XSS, user contents, open redirectors, cross-protocol redirections, X-Frame-Options, CORS, ...) across **origins**

- Transfer XSS in mxr.nozilla.org to addons
  (Hansen & Sokol, HTTPS Can Byte Me, BH’10)
CROSSING ORIGIN BOUNDARIES
STEAL OAUTH/OPENID TOKENS, SECRET URL FRAGMENTS...

- Introduction
- Cookie Cutter
  - Virtual Host Confusion
    - Crossing Origin Boundaries
    - Shared Reverse Proxies
  - Shared Session Cache
  - SPDY Connection Pooling
- Triple Handshake
- Conclusion
Cross-protocol redirection is harmful

- OAuth redirect_uri access control is origin based
- If the token origin can be confused with any origin with a redirect-to-HTTP, attacker wins
  - 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
PING www.pinterest.com (174.129.239.78) 56(84) bytes of data.
--- www.pinterest.com ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

root@argon:~# ping api.pinterest.com
PING api-origin.pinterest.com (54.225.157.104) 56(84) bytes of data.
--- api-origin.pinterest.com ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1006ms
Host confusion with user content origin

Common to use different top-level domain to avoid related-domain cookie attacks
  – dropboxusercontent.com, googleusercontent.com

User content origins should use separate certificates
Exploit: user contents

- 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
- **Short lived cookie forcing** allows temporary forcing of attacker session
  - Break into high trust origin, recover victim session
1. Attacker stores malicious file on his account
2. Temporary forcing of attacker session on victim
3. Rebind www.dropbox.com to dl-web.dropbox.com
4. Compromise victim’s session
EXPLOIT: SHARED SESSION CACHE
CONFUSE ORIGINS ACROSS CERTIFICATES

- Introduction
- Cookie Cutter
  - Virtual Host Confusion
    - Crossing Origin Boundaries
      - Shared Reverse Proxies
    - Triple Handshake
  - Shared Session Cache
  - SPDY Connection Pooling
- Conclusion
3 kinds of TLS authentication:
- Certificate
- 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.
Beware of TLS session cache

• Session cache sharing more common than ticket key sharing across servers
  – Seen on Amazon, Mozilla and Yahoo servers

• To exploit, **downgrade connection to SSL3**
  – Tickets have precedence over session identifier
1. Create SSL3 session on bugzilla.mozilla.org
2. Point bugzilla.mozilla.org to git.mozilla.org
3. Resume session and request malicious file
4. Virtual host fallback
EXPLOIT: SHARED REVERSE PROXY
IMPERSONATE THOUSANDS OF TOP RANKED WEBSITES

✓ Introduction
✓ Cookie Cutter
➢ Virtual Host Confusion
   ✓ Crossing Origin Boundaries
   ➢ Shared Reverse Proxies
➢ Triple Handshake
➢ Conclusion

✓ Shared Session Cache
➢ SPDY Connection Pooling
Beware of shared reverse proxies

- Shared reverse proxies are common (e.g. CDN)
- Handling of TLS is always awkward
  - CloudFlare: domain packing in one certificate
  - Akamai: dedicated IP for customer certificate
  - Google Apps: SNI (or dedicated IP)
- What is the fallback virtual host?
  - Akamai: default host is an open proxy (!)
Demo: Akamai
Preventing host confusion

- Do not mix low-trust and high-trust (sub)domains in certificates
- Configure a fallback host on every IP, that returns an error code (not a redirection)
  - Nginx: default_server option of listen directive
  - Apache: first VirtualHost that matches IP/port
• Server-side cache only required for SSL3 and can often be disabled
  – If required, server should have proper cache partition or let admin configure explicit shards (shared:XYZ:1m)

• With a server-wide ticket key, make sure all servers have the same configured hosts
  – Isolation of name-based hosts is weak in TLS
SPDY CONNECTION POOLING

WHO’S CONFUSING WHAT NOW?

- Introduction
- Cookie Cutter
  - Virtual Host Confusion
    - Crossing Origin Boundaries
    - Shared Reverse Proxies
  - Triple Handshake
  - Conclusion
- Shared Session Cache
  - SPDY Connection Pooling
Problem: websites use subdomains for origin isolation; requires a handshake for each

Idea: let’s reuse sessions even for requests to a different domain if:

1. New domain covered by initial certificate
2. DNS points to same server
SPDY connection pooling

**HTTP**
- Browser
- Server 1: w.com, i.w.com
- Server 2: fb.com
- https://fb.com/t
- https://w.com/y
- https://i.w.com/x

**SPDY**
- Browser
- Server 1: w.com, i.w.com
- Server 2: fb.com
- https://fb.com/t
- https://w.com/y
- https://i.w.com/x
None of the security theorems proved on TLS apply to browsers that reuse connections.

Every session-specific guarantees extends to all domains in the session’s certificate.

Standard in current HTTP2 IETF drafts.
Exploits

Sorry, not patched yet
TRIPLE HANDSHAKE
BREAKING CLIENT CERTIFICATE AUTHENTICATION

✓ Introduction
✓ Cookie Cutter
✓ Virtual Host Confusion
  ✓ Crossing Origin Boundaries
  ✓ Shared Reverse Proxies
✓ Shared Session Cache
✓ SPDY Connection Pooling

➢ Triple Handshake
➢ Conclusion

© Raed667
Reminder: TLS Handshake

- Handshake creates **new TLS session**
- Key exchange yields pre-master secret (PMS)
- Master secret: hash of PMS and nonces
- **Session parameters**: PMS, client & server certificates, cipher, session identifier
Background: Ray & Rex 2009 Attack

Alice

Handshake (New TLS Session)

Mallory

Handshake (New TLS Session)

Google

GET /malicious/action HTTP/1.1
X-Ignore-This:

Renegotiation

GET /legitimate/action HTTP/1.1
Cookie: SID=xyz
Background: Ray & Rex 2009 Attack

• TLS Weakness
  – Renegotiation doesn’t bind old and new sessions
  – Implementations allow server certificate to change
  – Implementations concatenate data across sessions

• HTTP Weakness
  – Message format is unstructured: can inject prefix
Mitigation: Ray & Rex 2009 Attack

- Mandatory renegotiation indication extension
- SRI = verify_data (hash of message log) of latest handshake on current connection
- SRI binds new TLS session to previous one
- Fresh connection: empty SRI
TLS session headache

2009 Renegotiation Attack

Triple Handshake Attack

synchronize

black hat
USA 2014
3Shake Step 1

User $u$
Client $C$

Attacker
Server $M$

Server $S$

TLS Handshake

TLS session ($sid$):
$\text{an} \text{on}(C) \rightarrow \text{cert}_M$
$ms, cr, sr$

TLS Handshake

TLS session ($sid$):
$\text{an} \text{on}(M) \rightarrow \text{cert}_S$
$ms, cr, sr$
3Shake Step 1

• C <-> M and M <-> S use same PMS
  – RSA: M re-encrypts C’s PMS under S’ public key
  – DHE: M sends degenerate group parameters

• PMS, MS, sid aren’t unique to a TLS session
3Shake Step 2

User $u$

Client $C$

Attacker

Server $M$

Server $S$

Resume$(sid)$

... Resume$(sid)$

Resumed $(sid)$:

$\text{anon}(C) \rightarrow \text{cert}_M$

$ms, cr', sr', cvd, svd$

Resumed $(sid)$:

$\text{anon}(M) \rightarrow \text{cert}_S$

$ms, cr', sr', cvd, svd$
• Resume C <-> M on C <-> S
  – TLS resumption doesn’t preserve authentication
• M doesn’t need to tamper any message: C and S agree on the same verify_data
• *tls-unique* binding broken after resumption
Data (injected by M) = GET /secret/data HTTP/1.1
Host: S
X-Ignore-This:

Data’ (sent by C) = GET / HTTP/1.0
Host: M
...
• M can trigger C <-> S renegotiation
  – *Certificate can still change*
• If S asks for client certificate, *C thinks she logs in on M, but actually authenticates to S*
• M can inject data to S before renegotiation
  – *Implementations still concats data across sessions*
3Shake Impact

• Conditions
  – C is willing to authenticate on M with her certificate
  – C ignores certificate change during renegotiation
  – S concatenates the data sent by M and C

• Impact
  – M can inject malicious data authenticated as C on S
• C can block server certificate changes
  – Chomium (CVE-2013-6628)
  – Safari (APPLE-SA-2014-04-22-2)
  – Internet Explorer (KB257591)
• We propose $MS' = PRF(PMS, {tls-session-hash})$
  – $tls-session-hash = $hash of the handshake messages that created the session up to client key exchange
CONCLUSION
WHY TLS FAILS TO PROTECT HTTP

✓ Introduction
✓ Cookie Cutter
✓ Virtual Host Confusion
  ✓ Crossing Origin Boundaries
  ✓ Shared Reverse Proxies
✓ Triple Handshake
✓ Shared Session Cache
✓ SPDY Connection Pooling

--> Conclusion
Lessons: Cookie Cutter

• “Liberal in what you accept”
  – Parsing is security critical, malformed = reject
• Security should not rely on anything being present (additions can relax security)
• Beware of side-effects on data processed before its integrity is confirmed
Lessons: virtual host confusion

• We want:
  – Routing to only depend on authenticated inputs
  – Consistent routing on servers sharing credentials

• Your job to achieve authenticated, consistent routing in current HTTPS software

• Beware of the “same-certificate policy”
  – Same-certificate attacker is possible!
Lessons: triple handshake

• We have a big TLS API problem
  – TLS isn’t just a drop-in socket replacement
  – All difficult problems handed off to the application

• Crypto values from handshake (PMS, MS, SID, verify_data) don’t identify session or participants
  – Will be fixed; lesson learned for TLS 1.3
What we are doing about it

• miTLS: verified TLS implementation
  – No more “goto fail” bugs
  – Performance vs “heartbleed” trade-off

• Verified protocol libraries
  – TLS API is too difficult for applications to use
  – Verify TLS + thin protocol wrapper together

• WebSpi, F*: evaluating the security of websites
### QUESTIONS

**Thanks**

<table>
<thead>
<tr>
<th>Google</th>
<th>HackerOne</th>
<th>Adam Langley</th>
<th>Alex Rice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mozilla</td>
<td>Dropbox</td>
<td>Piotr Sikora</td>
<td>Stephen Ludin</td>
</tr>
<tr>
<td>Microsoft</td>
<td>Akamai</td>
<td>Anton Mityagin</td>
<td>Eric Rescola</td>
</tr>
<tr>
<td>Facebook</td>
<td>Apple</td>
<td>Brian Sniffen</td>
<td>Ryan Sleevi</td>
</tr>
</tbody>
</table>