CSE 361: Web Security

Infrastructure Security

Nick Nikiforakis
HTTP Desync Attacks
HTTP Front-End and Back-End Servers

- In real-world settings, you often have a reverse proxy
  - Multiple incoming requests (different TCP connections)
  - Single TCP connection between front-end and back-end servers

- What happens if nginx and Django have a different understanding of how long a HTTP request is?
HTTP: How does a server determine length of content?

• Option 1: Content-Length: $length header
  • Read the value, subsequently read $length bytes

• Option 2: Transfer-Encoding: chunked
  1. Read single line, treat as hexadecimal representation of $length for data to come
     • Stop reading if $length = 0
  2. Read $length bytes
  3. Go to step 1

• What happens if you have both?
  • RFC 2616 says: If a message is received with both a Transfer-Encoding header field and a Content-Length header field, the latter MUST be ignored.
Pitfalls in parsing HTTP headers

- What happens if front-end takes first occurrence of the header, but back-end takes the last?

```
POST / HTTP/1.1
Host: example.com
Content-Length: 6
Content-Length: 5
12345G
```

[https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn]
Pitfalls in parsing HTTP headers

• Now assume a second request, from a **benign client**

```plaintext
POST / HTTP/1.1
Host: example.com
Cookie: session=1234
Content-Length: 6
123456

POST / HTTP/1.1
Host: example.com
Cookie: session=1234
Content-Length: 6
12345G

POST / HTTP/1.1
Host: example.com
Content-Length: 6
Content-Length: 5
12345

POST / HTTP/1.1
Host: example.com
Content-Length: 6
123456

GPOST / HTTP/1.1
Host: example.com
Cookie: session=1234
Content-Length: 6
123456
```

[https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn]
Result: Desync Attacks

- Front- and Back-end have different understanding of requests
  - Can smuggle in requests
  - Can hijack other user's requests
Result: Desync Attacks

• Front- and Back-end have different understanding of requests
  • Can smuggle in requests
  • Can hijack other user's requests

```
POST / HTTP/1.1
Host: example.com
Content-Length: 57
Content-Length: 1

1POST HTTP /sendmessage?to=attacker
Host: example.com
X: X

POST /sendmessage?to=user HTTP/1.1
Host: example.com
Content-Length: 14
message=secret

Victim

POST / HTTP/1.1
Host: example.com
Content-Length: 57
Content-Length: 1

1POST HTTP /sendmessage?to=attacker
Host: example.com
X: X

POST HTTP /sendmessage?to=user HTTP/1.1
Host: example.com
X: XPOST /sendmessage?to=user HTTP/1.1
Host: example.com
Content-Length: 14
message=secret

Backend
```
Additional problems in Desync attacks

• Some back-end systems look for the substring "chunked"
  • Attack: use Content-Length to fool front-end into accepting single request, use Transfer-Encoding: Xchunked to force TE for back-end

• Some back-end systems allow for whitespaces other than space
  • Transfer-Encoding:\	chunked ignored by front-end, understood by back-end
Transport Layer Security
Network Attacker

• Resides somewhere in the communication link between client and server
• Tries to disturb the confidentiality, integrity and authenticity of the connection
  • Observation of traffic (passive eavesdropper)
  • Fabrication of traffic (e.g., injecting fake packets)
  • Disruption of traffic (e.g., selective dropping of packets)
  • Modification of traffic (e.g., changing unencrypted HTTP traffic)
• "Man in the middle"
Possible types of a network attacker

- Within same network (ARP poisoning)
- Internet Service Provider (complete access to all traffic)
- Law Enforcement (access to traffic for specific user/to specific server)
- ... GCHQ, NSA, et al. (everywhere really)
Network attackers on the Web

• Active attacker
  • tries to modify, e.g., the response
    • Comcast added advertisements into unencrypted sites

• Passive attacker
  • tries to eavesdrop on exchange to learn information
    • e.g., credentials
Security in HTTP

• There is no security in HTTP.
• Solution: encapsulate HTTP traffic in secure channel
  • used to be SSL (HTTP via SSL)
  • nowadays Transport Layer Security (TLS)
• TLS adds security to HTTP
  • end-to-end encryption
  • server authentication
  • optional client authentication (rarely used in practice)
Transport Layer Security

- Replaces Secure Sockets Layer (SSL)
- Provides security for connection
  - integrity ensured by HMAC
  - confidentiality ensured by symmetric encryption
  - authentication with public-key cryptography
- Support numerous Cipher Suites
  - define key exchange, encryption and MAC types
# TLS Cipher Suites (RFC 5246)

<table>
<thead>
<tr>
<th>Cipher Suite</th>
<th>Key Exchange</th>
<th>Cipher</th>
<th>MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS_NULL_WITH_NULL_NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>TLS_RSA_WITH_NULL_MD5</td>
<td>RSA</td>
<td>NULL</td>
<td>MD5</td>
</tr>
<tr>
<td>TLS_RSA_WITH_NULL_SHA</td>
<td>RSA</td>
<td>NULL</td>
<td>SHA</td>
</tr>
<tr>
<td>TLS_RSA_WITH_NULL_SHA256</td>
<td>RSA</td>
<td>NULL</td>
<td>SHA256</td>
</tr>
<tr>
<td>TLS_RSA_WITH_RC4_128_MD5</td>
<td>RSA</td>
<td>RC4_128</td>
<td>MD5</td>
</tr>
<tr>
<td>TLS_RSA_WITH_RC4_128_SHA</td>
<td>RSA</td>
<td>RC4_128</td>
<td>SHA</td>
</tr>
<tr>
<td>TLS_RSA_WITH_3DES_EDE_CBC_SHA</td>
<td>RSA</td>
<td>3DES_EDE_CBC</td>
<td>SHA</td>
</tr>
<tr>
<td>TLS_RSA_WITH_AES_128_CBC_SHA</td>
<td>RSA</td>
<td>AES_128_CBC</td>
<td>SHA</td>
</tr>
<tr>
<td>TLS_RSA_WITH_AES_256_CBC_SHA</td>
<td>RSA</td>
<td>AES_256_CBC</td>
<td>SHA</td>
</tr>
<tr>
<td>TLS_RSA_WITH_AES_128_CBC_SHA256</td>
<td>RSA</td>
<td>AES_128_CBC</td>
<td>SHA256</td>
</tr>
<tr>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>RSA</td>
<td>AES_256_CBC</td>
<td>SHA256</td>
</tr>
<tr>
<td>TLS_DH_anon_WITH_RC4_128_MD5</td>
<td>DH_anon</td>
<td>RC4_128</td>
<td>MD5</td>
</tr>
<tr>
<td>TLS_DH_anon_WITH_3DES_EDE_CBC_SHA</td>
<td>DH_anon</td>
<td>3DES_EDE_CBC</td>
<td>SHA</td>
</tr>
<tr>
<td>TLS_DH_DSS_WITH_AES_128_CBC_SHA</td>
<td>DH_DSS</td>
<td>AES_128_CBC</td>
<td>SHA</td>
</tr>
<tr>
<td>TLS_DH_RSA_WITH_AES_128_CBC_SHA</td>
<td>DH_RSA</td>
<td>AES_128_CBC</td>
<td>SHA</td>
</tr>
<tr>
<td>TLS_DHE_DSS_WITH_AES_128_CBC_SHA</td>
<td>DHE_DSS</td>
<td>AES_128_CBC</td>
<td>SHA</td>
</tr>
<tr>
<td>TLS_DHE_RSA_WITH_AES_128_CBC_SHA</td>
<td>DHE_RSA</td>
<td>AES_128_CBC</td>
<td>SHA</td>
</tr>
<tr>
<td>TLS_DH_anon_WITH_AES_128_CBC_SHA256</td>
<td>DH_anon</td>
<td>AES_128_CBC</td>
<td>SHA256</td>
</tr>
<tr>
<td>TLS_DH_DSS_WITH_AES_256_CBC_SHA</td>
<td>DH_DSS</td>
<td>AES_256_CBC</td>
<td>SHA</td>
</tr>
<tr>
<td>TLS_DHE_DSS_WITH_AES_256_CBC_SHA</td>
<td>DHE_DSS</td>
<td>AES_256_CBC</td>
<td>SHA</td>
</tr>
<tr>
<td>TLS_DHE_RSA_WITH_AES_256_CBC_SHA</td>
<td>DHE_RSA</td>
<td>AES_256_CBC</td>
<td>SHA</td>
</tr>
</tbody>
</table>

**No protection**

- Uses RSA (certificate) for key exchange, AES 256 in CBC mode for encryption and SHA256 as MAC

**Uses ephemeral Diffie-Hellman with RSA for key exchange, AES 256 CBC for encryption and SHA256 as MAC**
(Very simplified) connection establishing in TLS with RSA

PMS = random

Nonceserver, cipherlistserver

certificate server

pubsub PMS

ChangeCipherSpec

MS = KDF(PMS, Nonceserver, Nonceclient)

Nonceclient, cipherlistclient

ChangeCipherSpec

MS = KDF(PMS, Nonceserver, Nonceclient)
(Very simplified) connection establishing in TLS with RSA

- Client sends list of available ciphers
  - server answers with his list
  - server selects first common suite
    - by default, uses client's priority, not its own

- Server transmits RSA certificate (including public key)

- Client generates PreMasterSecret, sends encrypted PMS to server

- Client and Server derive MasterSecret from PMS and nonces

- Crypto keys derived from MasterSecret used for data exchange
(Very simplified) connection establishing in TLS

PMS = random

MS = KDF(PMS, Nonceserver, Nonceclient)

Give me your private key now!
Decrypting TLS traffic after the fact

- Requirement for decryption: compute MasterSecret
  - based on nonces and PMS
- Both nonces are transmitted in clear text
- Client-generated PMS is encrypted with public key of server
  - if private key is compromised, attacker may decrypt previously recorded PMS
- Problem: server's RSA key does not change
- Solution: use ephemeral keys for key exchange to achieve forward secrecy
(Perfect) Forward Secrecy

- "Forward Secrecy" refers to inability to decrypt after the fact
  - key material is no longer available
- "Perfect Forward Secrecy" achieved if keys are never reused
  - compromising one key cannot compromise any other keys
- Desirable against Nation State Actors
- Requirement: generate ephemeral keys which cannot be recovered from the traffic
  - even if the private RSA key of the server is leaked
Reminder: Diffie-Hellman

- Two parties want to establish shared key
  - agree on common prime $p$ and generator $g$ (in TLS, given by the server)
  - parties generate private keys $a$ and $b$
  - compute public keys as $g^{\text{private}} \mod p$
  - knowing private keys allows to calculate common key $K$
- Bottom line: once you delete ephemeral private key, common secret cannot be recovered
(Very simplified) connection establishing in TLS with DHE

PMS $S^c \mod p$

Nonce$_{client}$

Nonce$_{server}$

certificate$_{server}$

ServerKeyExchange: $S$, sign($N_c$, $N_s$, $p$, $g$, $S$_{server})

ClientKeyExchange: $C$

$MS = \text{KDF}(PMS, \text{Nonce}_{server}, \text{Nonce}_{client})$

PMS $C^c \mod p$

$MS = \text{KDF}(PMS, \text{Nonce}_{server}, \text{Nonce}_{client})$
SSL Stripping

• Specific case of man-in-the-middle attack
  • presented at Blackhat 2009 by Moxie Marlinspike
• Goal: ensure that all traffic from victim is plaintext
  • attacker establishes HTTPS connection and forwards request
  • changes all links/redirects to HTTP
  • while rewriting HTML to show the target as HTTPS
Forcing HTTPS: HTTP Strict Transport Security

- HTTP header (Strict-Transport-Security) send by server
  - only valid if sent via HTTPS
  - `Strict-Transport-Security: max-age=<expiry in seconds>`
    - `includeSubDomains`: header is valid for all subdomains
    - `preload`: allows for inclusion in preload list
  - ensures that site cannot be loaded via HTTP until expiry is reached
- Caveat: need to visit page once to get header
- solution: HSTS preload list (https://hstspreload.org/)
  - only possible with at least 18 weeks max-age, includeSubDomains and automatic redirect from HTTP
Abusing HSTS for Tracking purposes

- HTTP Strict Transport Security controls connection behavior
  - once header is transmitted to client, all traffic to domain is forced via HTTPS
  - security feature, persisted across browsing sessions (also in incognito)

- Can be used to track user
  1. generate random ID for user on client
  2. for each bit set to 1, retrieve server resource
     - server sets HSTS header for each response
  3. to read ID, include script from each subdomain
     - HSTS-enabled subdomains automatically redirected by browser
     - on server, return id[<bit>] = 1 if script is accessed via HTTPS
HTTPS Certificates
Establishing trust in server's certificate

- Certificate is a "digitally signed document that binds a subject to some other information ... identity certificates bind names to keys... " (Gollmann's Computer Security book)
- "token that binds an identity to a cryptographic key" (Bishop's Computer Security: Art and Science)
- Need means to establish trust in certificate
- Solution: Public Key Infrastructure
- implements a chain of trust to presented certificate

<table>
<thead>
<tr>
<th>Subject Alt Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS Name</td>
</tr>
<tr>
<td>DNS Name</td>
</tr>
<tr>
<td>DNS Name</td>
</tr>
<tr>
<td>DNS Name</td>
</tr>
<tr>
<td>DNS Name</td>
</tr>
<tr>
<td>DNS Name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Public Key Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
</tr>
<tr>
<td>Key Size</td>
</tr>
<tr>
<td>Exponent</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Miscellaneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature Algorithm</td>
</tr>
<tr>
<td>Version</td>
</tr>
<tr>
<td>Download</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fingerprints</th>
</tr>
</thead>
</table>
Establishing trust in server's certificate

- Certificates bind key material to entity
- On technical level, certificate is a combination of
  - the server's public key,
  - its identity (domain name/common name),
  - and validity timestamps
  - (some more information on purpose of certificate and other technical details, e.g., allowed aliases)
- signed with issuer's (Certificate Authority) private key
Establishing trust in server's certificate

• Trusting a certificate boils down to trusting the CA
  • several root CAs integrated into browsers
  • CAs may be allowed to sign other CA's keys

• Example stonybrook.edu
  • certificate signed by InCommon RSA Server CA
  • ... which is signed by USERTrust RSA Certification Authority
  • ... which is in the browser
Interlude: Server Name Indication

- Single server may host several domains
  - TLS connection is made to IP address before Host header is sent
  - certificate validation is based on the domain name though
- In "good old days", only one certificate per IP possible
  - one certificate containing all domains hosted on the machine
  - horrible to maintain (e.g., adding a domain, revoking a domain's certificate)
- Solution: Server Name Indication
  - client sends desired domain name to server
  - server decides which certificate to present
  - widely adopted in browsers and servers since 2010
Validating ownership of a domain

• Most CAs use "domain validation"
  • send email to registrant (or webmaster@domain.com)
  • use DNS TXT entry with random token
  • ask owner to host file at certain path on their domain (Let's encrypt)

• More expensive solution: Extended Validation
  • requires proof of legal identity (e.g., Company) and physical location
  • Distinction no longer visible by default

• Problem: what if ownership changes?
  • need means to revoke certificate
Revoking certificates with CRLs

- Certificate Revocation Lists (CRLs)
  - frequently updated by CAs
  - contains list of all certificates which have been revoked
    - e.g., because of compromised keys
  - downloaded by browsers in regular intervals

- Several issues
  - interval of updates by CAs
  - interval of updates by browsers
  - blacklist does not ensure validity

- CRLs are (being) deprecated from browsers
Online Certificate Status Protocol (OCSP)

- Turn CRL approach around
  - client checks validity on request
  - response is signed to allow for verification
- Also has several drawbacks
  - potentially high load on OCSP servers
  - clients often implement "tryLater" incorrectly
    - connect anyways, check again later
  - privacy leak
    - OCSP server knows all HTTPS servers you visited
- Not enabled by default in all browsers
  - Chrome team claims "performance issues"
OCSP Stapling

- TLS-enabled servers regularly query OCSP server
  - get "ticket" with limited validity
  - OCSP response is "stapled" to TLS handshake
- Yields several benefits
  - less load on OCSP provider
  - no privacy leaks
- ... and a minor drawback
  - slight delay in invalidated certificates
- Certificates can be extended with Must-Staple
  - added to the original certificate

PKI relies on trusted parties

- PKI is based on chain of trust
- Root CAs have ultimate trust
  - by design no restriction for which domains certificates may be issued
- Several attack scenarios
  - compromised root CA may issue any certificate
  - root CA may create intermediate certificate authorities
Abusing Certificate Authorities

• Case 1: DigiNotar
  • Dutch root CA
  • compromised by attacker in 2011
  • over 500 certificates issued to unknown attacker
    • wildcard certificate for google.com (used to MitM Gmail users in Iran)
    • also Yahoo, Mozilla, WordPress, Tor Project, ...
  • Certificates blacklisted by Google, DigiNotar soon removed from list of trusted roots

• Case 2: TürkTrust
  • in December 2012, Google detected *.google.com certificates
  • linked back to TürkTrust
  • TürkTrust accidentally issued two intermediate CA certificates in 2011
  • root CAs removed from Firefox, Chrome kept them, but ignored EV
Abusing Certificate Authorities

• Most recent case: Symantec
  • owns VeriSign and Thawte root CAs
  • represents almost 30% of all valid certificates

• Several "mistakes"
  • improper checks before issuing EV certificates
  • misissued "test" certificates (including google.com, in 2015 and 2017)
  • allowed unauthorized employees access to CA keys
  • up to 30,000 certificates issued without proper validation

• Google proposed harsh reaction
  • decrease validity to at most nine months (until Chrome 64)
  • disable all EV functionality for Symantec certificates
Certificate Transparency

- **Goal:** be able to trace back malicious certificates
  - e.g., in DigiNotar case

- **Proposal:** use third party for append-only log
  - after (pre-)certificate submission, log issues
  - Signed Certificate Timestamp (SCT)
  - CA adds SCT to certificate, signs it, hands out

- **Chrome only allows Symantec certificate with EV if they are in CT logs**
  - enforced since June 2016

- **Since April 2018, all new certificates must have an SCT**

---

https://www.certificate-transparency.org/how-ct-works
DNS Certification Authority Authorization (CAA)

- Can we restrict the Cas who are allowed to issue certificates for our domains?

- DNS entry only meant for authorization of CAs
  - flag currently always set to 0, in the standard for future use
  - `<flag> issue "CA"`
    - allows CA to issue certificates for domain
    - can be set to issue ";" to ensure no certificates are issued
  - `<flag> issuewild "CA"`
    - allows CA to issue wild-card certificates for domain
  - `<flag> iodef "mailto:caa@domain.com"
    - any policy-violating attempt of a certificate must be reported there
    - should also be notified about any certificates being issued
CAA Quiz

• Let's Encrypt can issue certificates for www.stonybrook.edu
• Let's Encrypt and DigiCert can issue certificates for cs.stonybrook.edu
• Nobody can issue wildcard certs for stonybrook.edu
• Nobody can issue certificates for nossl.stonybrook.edu
• Any violations for the latter should be reported to admins@stonybrook.edu

www.stonybrook.edu CAA 0 issue 'letsencrypt.org'
cs.stonybrook.edu CAA 0 issue 'digicert.com'
cs.stonybrook.edu CAA 0 issue 'letsencrypt.org'
stonybrook.edu CAA 0 issuewild ';
nossl.stonybrook.edu CAA 0 issue ';
nossl.stonybrook.edu CAA 0 iodef 'mailto:admins@stonybrook.edu'
OpenSSL Incident

- Code overhaul of OpenSSL in 2006
  - using automated tool Valgrind to look for errors
  - discovered reading of uninitialized data
- Code was rewritten to remove usage of uninitialized data
  - actually, data was meant to add randomness
  - instead, PRNG only seeded with PID - only 32k values
- Result: trivial for attacker to precompute all 32k values
  - allowed for decryption of every TLS connection
  - (same for a lot of SSH keys)
SHA1 collisions

• PKIs use different hash algorithms for digital signatures
  • given the constructions, possibility to find collisions
  • finding colliding inputs allows for falsely signed certificates
  • MD5 broken in 2005, faded out of browsers until ~2011
• SHA1 known to be insecure for a while
  • initially not removed for compatibility
  • removed in 2017, when SHA-1 was broken in practice
  • after merely 9,223,372,036,854,775,808 operations ;-) 
• Bottom line: don't use SHA1 anymore
  • .. and always have a fancy logo
Attacking Crypto
History of Cryptography in the US

- After World War II, US deemed cryptographic algorithms "munition"
  - fell under heavy export control
  - required license to export
- Result: exporting strong encryption was prohibited
  - only weakened variants could be shipped
    (dubbed "export grade ciphers")
  - e.g., RSA only with up to 512bits
- Strong crypto was desired inside US though
  - special EXPORT cipher suites for foreigners made their way into codebase of OpenSSL
logjam/FREAK [CCS2015]

• Best attack on Diffie-Hellman is solving discrete logarithm problem
  • given $A = g^a \mod p$, recover $a$
  • feasible for 512 bits $p$ ("heavy lifting" done in 1 week of precomputation)

• DHE and DHE_EXPORT differ only in key length
  • MitM modifies cipher list of client to only contain DHE_EXPORT
  • In server's response, replaces DHE_EXPORT with DHE
  • client "thinks" server just chose small $p$
• Similar attack against RSA dubbed FREAK
Attacking implementations of crypto

- "Heartbeat" mechanism in OpenSSL to ensure that a server is still alive
  - Hey server. Are you there? If so, respond with "Parrot" (6 chars)
  - "Parrot"
- Heartbleed vulnerability (buffer overread)
  - Server trusts the client for the length of the string
  - Hey server. Are you there? If so, respond with "Parrot" (999 chars)
  - "Parrot e3 5c 29 2b a3 b7 35 93 db 29 2c 66 0d 48 11 64 5f f2 9f 4e d7 ca c4 a6 e3 01 24 97 0a dd 75 15 ec 4a 10 e9 b8 93 98 30 af ba 48 5d d5 57 4d a2 53 28 a0 0a aa fa 5b c2 dd 56 38 15 74 52 6f 80 a2 63 6d 22 cc 58 af 50 cf fb 51 9f 7b f0 c9 f4 4b d5 a5 f1 90 15 09 b5 80 03 02 6b e4 c3 97 0c 23 c8 fb 76"

Adjacent memory containing passwords, cookies, private keys, etc.
Summary

Security in HTTP
- There is no security in HTTP.
- Solution: encapsulate HTTP traffic in secure channel
  - used to be SSL (HTTP via SSL)
  - nowadays Transport Layer Security (TLS)
- TLS adds security to HTTP
  - end-to-end encryption
  - server authentication
  - optional client authentication (rarely used in practice)

Certificate Transparency
- Goal: be able to trace back malicious certificates
  - e.g., in DigiNotar case
- Proposal: use third party for append-only log
  - after pre-certificate submission, log issues
  - Signed Certificate Timestamp (SCT)
  - CA adds SCT to certificate, signs it, hands out
- Chrome only allows Symantec certificate with EV
  - if they are in CT logs
  - enforced since June 2018
- Since April 2018, all new certificates must have an SCT

Establishing trust in server’s certificate
- Trusting a certificate boils down to
  - trusting the CA
  - several root CAs integrated into browsers
  - CAs may be allowed to sign other CAs’s keys
- Example storybrooke.edu
  - certificate signed by InCommon RSA Server CA
  - ... which is signed by USERTrust RSA Certification Authority
  - ... which is in the browser
Credits

• Original slide deck by Ben Stock
• Modified by Nick Nikiforakis