

TLS Security, TLS 1.3

Martin Stanek

Department of Computer Science
Comenius University
`stanek@dcs.fmph.uniba.sk`

Cryptology 1 (2022/23)

Content

Fixing issues in practice

- Trust, Checking certificates, OCSP stapling
- Certificate pinning, Certificate Transparency
- Using TLS: HSTS, STARTTLS

Selected vulnerabilities and problems

- Implementation bugs
- Protocol/cryptography problems

TLS 1.3

Trust – certificates (PKI)

- ▶ trusted CA certificates distributed by browsers/OS
- ▶ example: Firefox (december 2022):
 - ▶ 56 unique CA owners (organizations/companies)
 - ▶ 168 CA certificates
- ▶ Do you trust them all?
- ▶ Certificate validation – chain, expiration, server name, signatures, check revocation, ... *bugs are common*
- ▶ User – let's ignore warnings/errors

Trust – reality

- ▶ (2014-2015) Lenovo Superfish – self-signed CA pre-installed, automatic MITM attack (inserting ads to web pages), private key shared among installations
- ▶ (2011) DigiNotar (NL) – compromised since 2009, fake certificates (MITM), removed from the list of trusted CA, bankruptcy
- ▶ (2011) Comodo – registration authority account compromised, 9 fake certificates
- ▶ (2017-2018) distrust of Symantec CA (and its subordinates: Thawte, GeoTrust, RapidSSL) – business sold to DigiCert
- ▶ (2018) Trustico (former reseller for Symantec) – sending 23.000 private keys to DigiCert by e-mail ... to revoke the certificates
- ▶ Serrano et al. *A complete study of P.K.I. (PKI's Known Incidents)*, 2019

Checking certificates

- ▶ checking certificate status: OK or revoked?
- ▶ several standard options:
 - ▶ CRL (Certificate revocation list) – a list signed by CA, issued frequently (e.g. at least every 24 hours); can be large (e.g. GlobalSign's CRL from 22 kB to 4.7 MB thanks to Heartbleed)
 - ▶ OCSP (Online Certificate Status Protocol) – requesting info from CA; response with timestamp; signed by CA
- ▶ non-standard approach:
 - ▶ CRLSet (Chrome), OneCRL (Firefox) – list of selected revoked certificates distributed as an update to the browser (Chrome – selected certificates; FF – intermediate certificates)

OCSP stapling

- ▶ problems with OCSP:
 - ▶ What to do if there is no response from CA – block or allow?
 - ▶ user privacy (CA learns what certificates client wants to check)
 - ▶ CA flooded with requests related to sites with high traffic.
 - ▶ slower user experience.
- ▶ TLS Certificate Status Extension
- ▶ idea: server requests OCSP response at regular intervals and adds it as Certificate Status message in the Handshake
 - ▶ the response cannot be forged (timestamp, signed by CA)
- ▶ Multiple Certificate Status Request Extension
 - ▶ providing status for all certificates in a chain
 - ▶ original extension: only for server's own certificate
- ▶ OCSP Must-staple
 - ▶ certificate extension – server must staple, otherwise the certificate is invalid

HTTP Public key pinning (HPKP)

- ▶ problem: compromised CA issues fake certificates
- ▶ bind host to known public-key (or keys)
- ▶ information in HTTP header
- ▶ “trust on first use” mechanism
- ▶ limitations
 - ▶ cannot detect MITM attack in the first connection
 - ▶ attacker can even insert own pinning info in this case
- ▶ now deprecated, replaced by Certificate Transparency

Certificate Transparency

- ▶ goals:
 - ▶ make hard for a CA to issue a certificate for domain that is not visible to domain owner
 - ▶ allow to monitor and audit issued certificates (e.g. by domain owners or CA)
 - ▶ protect users against certificates issued maliciously or mistakenly
- ▶ Certificate Transparency log
 - ▶ Merkle tree of certificate chains (or precertificate chains)
 - ▶ publicly verifiable
 - ▶ signed root
- ▶ CA publishes certificates (precertificates) to public logs
- ▶ SCT – Signed Certificate Timestamp – log's promise to incorporate the certificate in the Merkle tree
 - ▶ new certificates contains SCT(s)

HSTS

- ▶ SSL Stripping
 - ▶ attacker: MITM proxy replacing links https with http links
 - ▶ user clicks on a link ...
 - ▶ victim communicates with attacker via http
 - ▶ attacker communicates with the web server via https
- ▶ HSTS (HTTP Strict Transport Security, RFC 6797)
 - ▶ HSTS headers over https – instructing browser to use only https for all future requests
 - ▶ browser transforms all http links into https links
 - ▶ browser does not allow unsecured connections to the web server
- ▶ limitations
 - ▶ HSTS header stripped in first visit (pre-loaded list of HSTS sites in browsers – does not scale)
- ▶ supported: Firefox, Chrome, Safari

STARTTLS

- ▶ Opportunistic TLS, switch from plaintext to TLS connection
- ▶ STARTTLS command
 - ▶ supported: SMTP, POP3, IMAP, LDAP, etc.
- ▶ STRIPTLS attack – removing STARTTLS

Apple “goto” fail (2014)

```
SSLVerifySignedServerKeyExchange(...)
{
    OSStatus  err;
    ...
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    ...
    err = sslRawVerify(...)

fail:
    ...
    return err;
}
```

Heartbleed (2014)

- ▶ bug in OpenSSL (2012 – 2014)
- ▶ heartbeat extension (RFC 6520):
 1. $A \rightarrow B$: please, reply with these 3 bytes: “abc”
 2. $B \rightarrow A$: “abc”
- ▶ the problem:
 1. $A \rightarrow B$: please, reply with these 10000 bytes: “abc”
 2. $B \rightarrow A$: “abc” 00 a3 30 e2 ... 7f
- ▶ possible leaks: private keys, master secret, passwords, ...
- ▶ huge impact on security – OpenSSL everywhere
- ▶ client software affected as well (server can issue a heartbeat request too)

Do you use some 3rd party code critical to the security of your system?

How well is it maintained, reviewed, audited?

Discussion on security of TLS (informal)

- ▶ TLS 1.2 (RFC 5246)
- ▶ Appendix F: Security Analysis
 - ▶ F.1. Handshake Protocol
 - ▶ F.1.1. Authentication and Key Exchange
 - F.1.1.1. Anonymous Key Exchange
 - F.1.1.2. RSA Key Exchange and Authentication
 - F.1.1.3. Diffie-Hellman Key Exchange with Authentication
 - ▶ F.1.2. Version Rollback Attacks
 - ▶ F.1.3. Detecting Attacks Against the Handshake Protocol
 - ▶ F.1.4. Resuming Sessions
 - ▶ F.2. Protecting Application Data
 - ▶ F.3. Explicit IVs
 - ▶ F.4. Security of Composite Cipher Modes
 - ▶ F.5. Denial of Service
 - ▶ F.6. Final Notes
- ▶ interesting part: D.4. Implementation Pitfalls

Problem with PKCS #1 v1.5 (RSA encryption)

- ▶ (1998) Bleichenbacher
- ▶ RSA key exchange
- ▶ plaintext (after padding) contains bytes 00 02 (ClientKeyExchange)
- ▶ server's response for a ciphertext indicates correct/incorrect format
- ▶ access to this oracle allows to decrypt arbitrary ciphertext (i.e. pre-master secret)
- ▶ requires approx. 1 million queries for 1024 bit modulus

Problem with PKCS #1 v1.5 – solutions

- ▶ treat incorrect format as a correct one:

“In any case, a TLS server MUST NOT generate an alert if processing an RSA-encrypted premaster secret message fails, or the version number is not as expected. Instead, it MUST continue the handshake with a randomly generated premaster secret.”

- ▶ use RSA-OAEP (not used in TLS 1.2):

“The RSAES-OAEP encryption scheme defined in [PKCS1] is more secure against the Bleichenbacher attack. However, for maximal compatibility with earlier versions of TLS, this specification uses the RSAES-PKCS1-v1_5 scheme.”

- ▶ avoid RSA key exchange altogether (TLS 1.3)

ROBOT (2017)

- ▶ Böck, Somorovsky, Young (2017)
- ▶ ROBOT (Return Of Bleichenbacher's Oracle Threat)
- ▶ affected vendors: F5, Cisco ACE, Citrix, Radware, ...
- ▶ affected web sites: Facebook, PayPal, etc.
- ▶ countermeasures specified in TLS 1.2 not implemented correctly
- ▶ expected: TLS alert 20 (bad_record_mac) after Finished for all RSA decryption failures
- ▶ examples of problems:
 - ▶ immediate reset of TCP connection for prefix/padding errors
 - ▶ ChangeCipherSpec/Finished withheld – waiting for these messages iff the ciphertext in ClientKeyExchange was valid
 - ▶ specific alert (not 20) vs. timeout allowed for valid ciphertext
 - ▶ different error messages for valid and invalid ciphertexts

Timing attack, PRNG

- ▶ Timing attack

- ▶ (2003) Boneh, Brumley
- ▶ timing RSA decryptions (processing ClientKeyExchange by server)
- ▶ analyzing correlations between time and private key allows to reconstruct the private key
- ▶ improvements for RSA with CRT and Montgomery multiplication exist
- ▶ solution: RSA blinding

- ▶ PRNG

- ▶ client and server nonces, pre-master secret for RSA key exchange, DH private parameters for (EC)DHE key exchange, IV for AEAD
- ▶ unpredictability, initialization
- ▶ (2006-2008) Debian OpenSSL PRNG initialization bug
- ▶ (2017) DUHK – Don't Use Hardcoded Keys – some (usually older) FIPS 140-2 certified implementations of ANSI X9.31 PRNG (seed key hardcoded in firmware)

“Renegotiation” problem (2009)

- ▶ renegotiation – a new handshake can be initialized anytime by client or server (inside an existing TLS connection)
- ▶ attacker creates his own TLS connection with the server and inserts victim’s handshake as a renegotiation ...inserting arbitrary data as a prefix to the victim’s data
- ▶ root cause: missing binding between renegotiation handshake and TLS parameters in use
- ▶ RFC 5746 (TLS Renegotiation Extension):
 - ▶ basically, Hello messages must contain verification data from previous Finished messages

BEAST (2011)

- ▶ BEAST (Browser Exploit Against SSL/TLS)
- ▶ known vulnerability of CBC mode implementation made practical
- ▶ CBC mode (TLS 1.0 and older) – IV for a packet is the last block from previous packet
- ▶ assumptions: plaintext (request) manipulation (e.g. Javascript), access to the ciphertext (eavesdropping)

BEAST (2)

- ▶ attack idea:
 - ▶ let C_0, \dots, C_l be a ciphertext (a packet)
 - ▶ attacker wants to decrypt P_j
 - ▶ attacker chooses plaintext with the first block $P'_1 = C_{j-1} \oplus C_l \oplus x$
 - ▶ then $C'_1 = E_k(P'_1 \oplus C_l) = E_k(C_{j-1} \oplus x)$
 - ▶ therefore $C'_1 = C_j \Leftrightarrow x = P_j$
 - ▶ guessing an entire block is hard
 - ▶ attacker moves the boundaries of plaintext (e.g. a cookie) so he will guess just 1 byte (other bytes can be known “any_session_id=?”)
 - ▶ different shift allows “ny_session_id=7?”) etc.
- ▶ fix in TLS 1.1 – explicit IV
- ▶ other mitigation strategy in browsers: 1/n-1 record splitting
- ▶ remark:
 - ▶ weakness in CBC known since 2002; TLS 1.1 standardized in 2006
 - ▶ practical attack: BEAST in 2011
 - ▶ default support for TLS 1.1 in browsers in 2011: none

CRIME (2012)

- ▶ CRIME (Compression Ratio Info-leak Made Easy)
- ▶ using compression in SSL/TLS for leaking plaintext information
- ▶ assumptions: compression, plaintext (request) manipulation, access to the ciphertext (eavesdropping)
- ▶ attack idea:
 - ▶ attacker adds data to plaintext, e.g. “Cookie: company_session_id=??”
 - ▶ ciphertext will be probably shorter if “??” is guessed correctly
 - ▶ subsequent bytes can be tested after correct guess
 - ▶ “playing” with the plaintext to see differences when padding is used
 - ▶ divide et impera approach: add strings for half of all guessed values \Rightarrow less requests, faster attack
- ▶ fix: disable compression in TLS

POODLE (2014)

- ▶ POODLE (Padding Oracle On Downgraded Legacy Encryption)
- ▶ attacking CBC mode padding in SSL 3.0
- ▶ SSL 3.0 prescribes only the value of the last byte in padding (e.g. 07 for a complete padding block and 3DES algorithm)
- ▶ padding is not an input to MAC (SSL: MAC-then-Encrypt)
- ▶ client and server might be willing to use SSL 3.0 in case of problems with the handshake (even though they both support TLS)
- ▶ assumptions: attacker can create requests (e.g. Javascript), manipulate the ciphertext, and observe server's responses

POODLE (2)

- ▶ padding oracle attack, and shifting unknown plaintext (e.g. cookie)
- ▶ request || MAC fits to block boundary (i.e. padding fills an entire block)
GET /path Cookie: secret=xx || **xxxxxxYX** || ... || MAC || padding 07
- ▶ attacker replaces last ciphertext block with the CT block for the red block
server continues \Rightarrow final byte 07, compute **X** (prob. 1/256)
server rejects \Rightarrow new handshake; new test
- ▶ attacker shifts the plaintext to decrypt other bytes of cookie:
GET /path2 Cookie: secret=x || **xxxxxxY** || X.. || MAC || padding 07
etc.
- ▶ fix – remove SSL 3.0 from browsers (at least disable)

POODLE (3)

- ▶ TLS padding is a special case of SSL 3.0 padding

msg || 00

msg || 01 01

msg || 03 03 03 03

- ▶ What if padding verification is performed as in SSL 3.0 case?
- ▶ no need for fallback to SSL 3.0
- ▶ POODLE for TLS (SSL Pulse, <https://www.ssllabs.com/ssl-pulse/>)
vulnerable servers:
 - ▶ 10.1% (XII/2014)
 - ▶ 1.9% (XII/2016)
 - ▶ 0.4% (XII/2018)
 - ▶ 0.0% (XII/2020) ...end of story?

POODLE continues

- ▶ various methods of detecting padding oracles
 - ▶ in the handshake (Finished message) or in application data
 - ▶ in/valid/missing MAC combined with in/valid padding
 - ▶ different ciphersuites behave differently, ...
- ▶ GOLDENDOODLE, Zombie POODLE, Sleeping POODLE, ...
- ▶ Merget et al. (2019): *Scalable Scanning and Automatic Classification of TLS Padding Oracle Vulnerabilities*
 - ▶ systematic analysis of servers' behavior
 - ▶ conservative estimate: 1.83% of TLS servers vulnerable

FREAK (2015)

- ▶ “export-grade” algorithms – RSA 512 bits
- ▶ clients (usually) did not request such algorithms ...but they accepted such short RSA public key
 - ▶ note: TLS 1.1 and older allow temporary RSA key in key exchange
- ▶ supported by web servers (March 2015, approx. 36%)
- ▶ MITM attack:
 1. attacker replaces cipher suites with export RSA in ClientHello
 2. attacker decrypts pre-master secret and computes keys
 3. knowing master secret allows “fixing” Finished messages
- ▶ no need to factorize 512 bit RSA often:
 - ▶ apache: generated when server starts and reused when needed
 - ▶ March 2015: 1 factorization ~ 7.5 hours in EC2 for 104 USD

Logjam (2015)

- ▶ “export-grade” algorithms again – DHE (short group, e.g. 512-bit)
- ▶ clients (usually) did not request ...but they accepted
- ▶ MITM attack (similar to FREAK):
 1. attacker replaces cipher suites with export DHE in ClientHello
 2. attacker computes dlog, obtains pre-master secret and computes keys
 3. knowing master secret allows “fixing” Finished messages
- ▶ computing dlog (even with export-grade parameters) is not possible in real-time
 - ▶ 2 phases: pre-computation (slow, but depends on group and g only) and on-line (for given input, fast)
 - ▶ DHE_EXPORT: 99% hosts choose one of three 512-bit primes
- ▶ remark: fixed groups are used everywhere (IPSec, TLS, SSH); 1024-bit primes within reach of a state-sponsored agency
 - ▶ passive eavesdropping on VPN, SSH?

DROWN (2016)

- ▶ DROWN (Decrypting RSA using Obsolete and Weakened eNcryption)
- ▶ TLS and SSL 2.0 enabled on server (or servers), sharing the same RSA key
- ▶ RSA key exchange
- ▶ cross-protocol attack
- ▶ decrypting pre-master secret from TLS 1.x RSA key exchange
 - ▶ using SSL 2.0 to mount Bleichenbacher's attack
 - ▶ exploiting export-grade cipher suite, and other SSL 2.0 weaknesses
- ▶ fix: disable SSL 2.0 everywhere

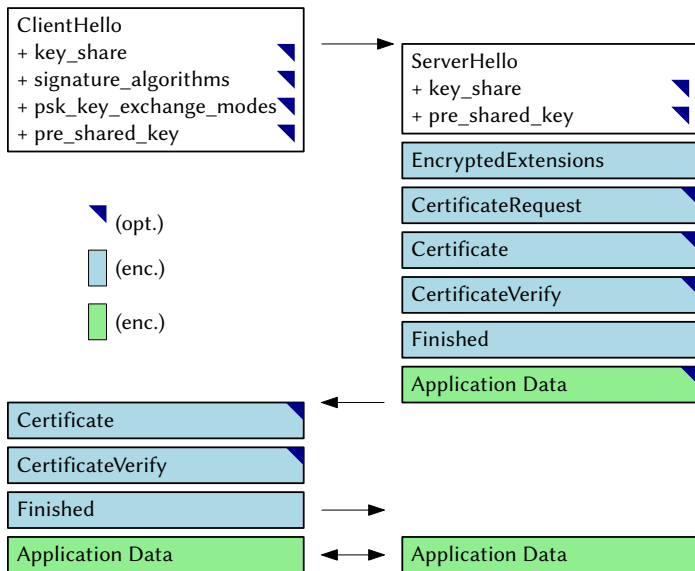
TLS 1.3 – major changes from TLS 1.2

- ▶ RFC 8446 (2018)
- ▶ AEAD ciphers only (support for non-AEAD ciphers removed)
- ▶ public-key key exchange with forward secrecy (static RSA and Diffie-Hellman removed)
- ▶ redesigned key derivation function: HMAC-based Extract-and-Expand Key Derivation Function (HKDF)
- ▶ reworked handshake: 1-RTT (1 round trip time) mode
- ▶ new zero round-trip time (0-RTT) mode
- ▶ other things removed: custom DHE groups, support for compression, DSA
- ▶ RSA-PSS is used instead of PKCS#1 v1.5 for handshake signatures
- ▶ TLS supports three basic key exchange modes:
 - ▶ Diffie-Hellman (over the finite fields and or elliptic curves)
 - ▶ pre-shared symmetric key (PSK)
 - ▶ combination of PSK and Diffie-Hellman

TLS 1.3 – mandatory cipher suites

- ▶ symmetric cipher suite (AEAD + hash function HKDF):
 - ▶ MUST: TLS_AES_128_GCM_SHA256
 - ▶ SHOULD: TLS_AES_256_GCM_SHA384, TLS_CHACHA20_POLY1305_SHA256
- ▶ digital signatures:
 - ▶ MUST: rsa_pkcs1_sha256 (for certificates), rsa_pss_rsae_sha256 (for CertificateVerify and certificates), and ecdsa_secp256r1_sha256
- ▶ key exchange:
 - ▶ MUST: secp256r1 (NIST P-256)
 - ▶ SHOULD: X25519

TLS 1.3 – Handshake protocol



Discussion on security of TLS 1.3 (informal)

- ▶ TLS 1.3 (RFC 8446)
- ▶ Appendix E. Overview of Security Properties
 - ▶ E.1. Handshake
 - ▶ E.1.1. Key Derivation and HKDF
 - ▶ E.1.2. Client Authentication
 - ▶ E.1.3. 0-RTT
 - ▶ E.1.4. Exporter Independence
 - ▶ E.1.5. Post-Compromise Security
 - ▶ E.1.6. External References
 - ▶ E.2 Record Layer
 - ▶ E.2.1. External References
 - ▶ E.3. Traffic Analysis
 - ▶ E.4. Side-Channel Attacks
 - ▶ E.5. Replay Attacks on 0-RTT
 - ▶ E.5.1. Replay and Exporters
 - ▶ E.6. PSK Identity Exposure
 - ▶ E.7. Sharing PSKs
 - ▶ E.8. Attacks on Static RSA
- ▶ interesting part: C.3. Implementation Pitfalls

TLS 1.3 – Selfie

- ▶ Selfie (2019) – first protocol attack on TLS 1.3
 - ▶ rarely used case of (external) PSK authentication
 - ▶ scenario: client can also be a server
 - ▶ simple reflection: attacker resend all messages back to client
 - ▶ client establishes connection with itself
 - ▶ limited impact in practice