Security at the Transport Layer

CS 352, Lecture 21
http://www.cs.rutgers.edu/~sn624/352-S19

Srinivas Narayana
(heavily adapted from slides by Prof. Badri Nath and the textbook authors)
Transport Layer Security
TLS/SSL
TLS/SSL: A Secure Sockets Layer

• widely deployed security protocol
  • supported by almost all browsers, web servers
  • https
  • billions $/year over TLS/SSL

• Origins: [Woo 1994] implementation by Netscape

• provides
  • confidentiality
  • integrity
  • authentication

• original goals:
  • Web e-commerce transactions
  • encryption (especially credit-card numbers)
  • Web-server authentication
  • optional client authentication
  • minimum hassle in doing business with new merchant

• available to all TCP applications
  • secure socket interface
TLS/SSL and the rest of the protocol stack

- TLS/SSL provides application programming interface (API) to applications
- C and Java libraries/classes readily available
  - Ex: OpenSSL
Could do something like PGP

- but want to send byte streams & interactive data
- want set of secret keys for entire connection
- want certificate exchange as part of protocol: handshake phase
TLS/SSL protocol: Constituents

- **handshake**: Alice and Bob use their certificates, private keys to authenticate each other and exchange shared secret
- **key derivation**: Alice and Bob use shared secret to derive set of keys
- **data transfer**: data to be transferred is broken up into series of records
- **connection closure**: special messages to securely close connection
Step (1): a simple handshake

$MS$: master secret

$EMS$: encrypted master secret

Q: What all might the “master secret” be used for?
Step (2): key derivation

- considered bad to use same key for more than one cryptographic operation
  - use different keys for message authentication code (MAC) and encryption

- four keys:
  - $K_c = \text{encryption key for data sent from client to server}$
  - $M_c = \text{MAC key for data sent from client to server}$
  - $K_s = \text{encryption key for data sent from server to client}$
  - $M_s = \text{MAC key for data sent from server to client}$

- keys derived from key derivation function (KDF)
  - Takes master secret and (possibly) some additional random data and creates the keys
Step (3): Data records

- why not encrypt data in constant stream as we write it to TCP?
  - where would we put the MAC? If at end, no message integrity until all data processed.
  - e.g., with instant messaging, how can we do integrity check over all bytes sent before displaying?
- instead, break stream in series of records
  - each record carries a MAC
  - receiver can act on each record as it arrives
- How does receiver distinguish MAC from data within a record?
  - want to use variable-length records
Defending against replay attacks?

- **Problem:** What if attacker could record and replay all or some records?

- **Solution:**
  - Use nonce (ex: cipher block chaining)
  - Handles both record replay and *connection* replay
Defending against reordering?

- **Problem:** What if attacker re-orders the records?
  - Issue: per-record MAC provides only record-level integrity
  - But record-level integrity is not the same as *stream-level* integrity

- **Solution:**
  - Put sequence number in MAC:
    - \( MAC = MAC(M_x, \text{sequence-number || data}) \)
  - **Note:** no sequence number field on record
    - Instead, sender and receiver implicitly keep seq numbers on records
Defending against truncation?

• **problem:** truncation attack
  • attacker forges TCP connection close segment
  • one or both sides thinks there is less data than there actually is
  • A stream of records in order so far != stream being *complete*

• **solution:** record types, with one type for closure
  • type 0 for data; type 1 for closure

• \( \text{MAC} = \text{MAC}(M_x, \text{sequence} \| \text{type} \| \text{data}) \)
TLS/SSL: summary so far

Handshake

hello
certificate, nonce
Nonce, $K^+(MS) = EMS$
type 0, seq 1, data
type 0, seq 2, data
type 0, seq 1, data
type 0, seq 3, data
type 1, seq 4, close
type 1, seq 2, close

Key derivation

Encrypted data transfer

bob.com

Special record type for connection close
But our TLS/SSL description isn’t complete yet

- how long are fields?

- which encryption protocols?

- Could we implement *negotiation*?
  - allow client and server to support different encryption algorithms
  - allow client and server to choose together specific algorithm before data transfer
TLS/SSL “Cipher Suite”

• Cipher suite
  • public-key algorithm
  • symmetric encryption algorithm
  • MAC algorithm

• TLS/SSL supports several cipher suites

• Negotiation: client, server agree on cipher suite
  • client offers choices
  • server picks one

Common symmetric ciphers
- AES – Advanced Encryption Standard
- DES – Data Encryption Standard: block
- 3DES – Triple strength: block
- ChaCha: stream
- RC4 – Rivest Cipher 4: stream

SSL Public key encryption
- RSA with DH

Message authentication code
- HMAC-MD5 and others
Purpose

1. server authentication
2. negotiation: agree on crypto algorithms
3. establish keys
4. client authentication (optional)
1. client sends list of algorithms it supports, along with client nonce
2. server chooses algorithms from list; sends back: choice + certificate + server nonce
3. client verifies certificate, extracts server’s public key, generates pre_master_secret, encrypts with server’s public key, sends to server
4. client and server independently compute encryption and MAC keys from pre_master_secret and nonces
5. client sends a MAC of all the handshake messages
6. server sends a MAC of all the handshake messages
Improved Handshake with Negotiation (3/5)

Why steps (5) and (6)?
(MAC of all handshake messages)

• client typically offers range of algorithms, some strong, some weak
• man-in-the middle could delete stronger algorithms from list
• last 2 steps prevent this
  • Note that the last two messages are encrypted
Improved Handshake with Negotiation (4/5)

• Why two nonces?
  • suppose Trudy sniffs all messages between Alice & Bob
  • next day, Trudy sets up TCP connection with Bob, sends exact same sequence of records
    • Bob (Amazon) thinks Alice made two separate orders for the same thing
    • solution: Bob sends different random nonce for each connection. This causes encryption keys to be different for each connection.
    • Trudy’s messages will fail Bob’s integrity check
Key derivation as part of handshake (5/5)

• client nonce, server nonce, and pre-master secret are fed into a pseudo random-number generator
  • produces master secret
• master secret and nonces are fed into another random-number generator to get a key block
• key block sliced and diced:
  • client MAC key
  • server MAC key
  • client encryption key
  • server encryption key
  • client initialization vector (for CBC)
  • server initialization vector (for CBC)
TLS/SSL protocol messages

record header: content type; version; length

MAC: includes sequence number, MAC key $M_x$

fragment: each SSL fragment $2^{14}$ bytes (~16 Kbytes)
TLS/SSL record format

data and MAC encrypted (symmetric key algorithm)
Real TLS/SSL connection

TCP FIN follows everything henceforth is encrypted
TLS/SSL: the big picture

• A protocol to agree on a set of ciphers for security properties
  • Not a cipher itself

• Can be used by any application at the app layer

• Customized for security properties at the record-level and stream-level, to work with real applications

• Latest standard: TLS v1.3 (Aug 2018)
Activity

• Look at the certificate for a website on your browser

• badssl.com

• https://www.selfsignedcertificate.com

• openssl x509 -in test-self-signed-cert-cert.cert -text -noout
Security: The Big Picture

• Cryptography: building block for security on the Internet today
  • Symmetric key (ex: AES) and public key (ex: RSA)
  • Provide confidentiality, but need more work for other properties

• Message integrity through MAC

• Digital signatures: authenticity, integrity, non-repudiation

• Certificate authorities

• Real applications and protocols: PGP and TLS/SSL

• Many real benefits from the tools and techniques in this field

• Many real problems remain