HTTP Security

- Server/Proxy requests username and password

Web server	Proxy server
Unauthorized status code: 401	Unauthorized status code: 407
WWW-Authenticate	Proxy-Authenticate
Authorization	Proxy-Authorization
Authentication-Info	Proxy-Authentication-Info

- Browser remembers result of successful authentication
  - Automatically includes appropriate authorization, eliminating the additional authentication steps
  - For all subsequent requests to the same server

HTTP Security (cont'd)

- Basic authentication
  - *Username* and *password* travel completely exposed across the network
  - Client combines the two, separated by a colon (:), and encodes them according to the rules for Base64 encoding

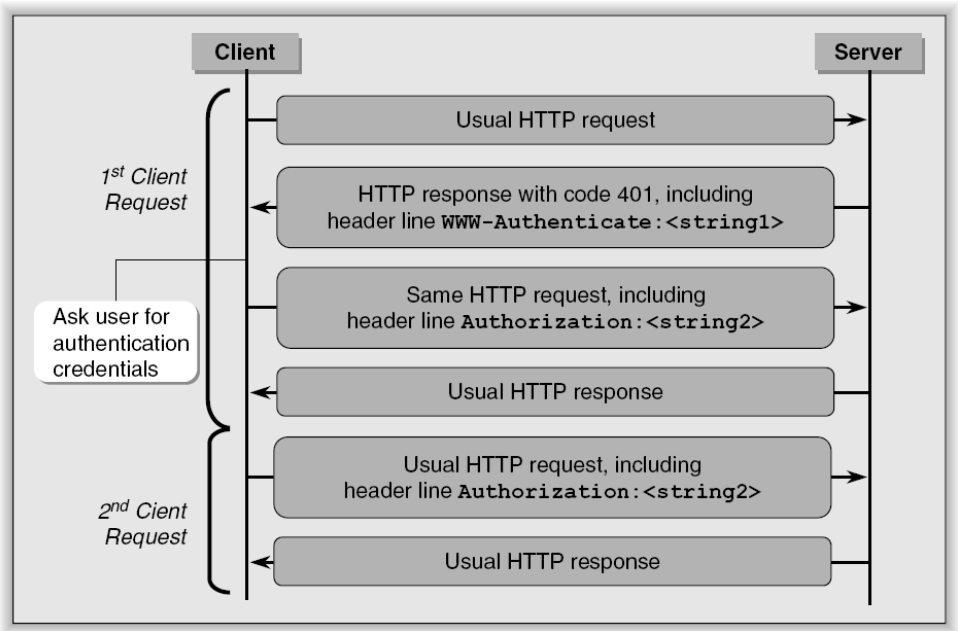
server response:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic
        realm="users@inc.com"
```

retry request:

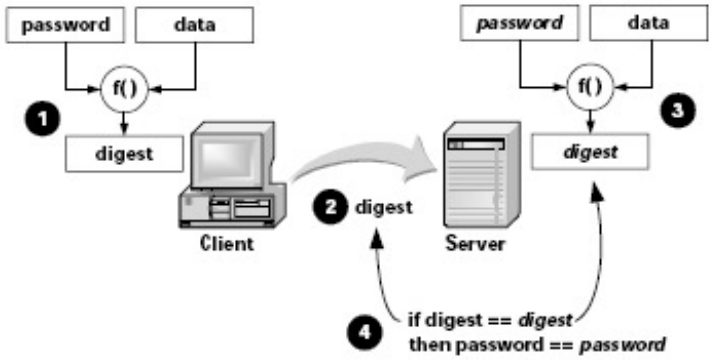
```
GET /my/secret.html HTTP/1.1
Authorization: Basic ZFt2QwxhZGRpbjpvcGVuIHNlcQ==
```

(cont'd)



Message Digests

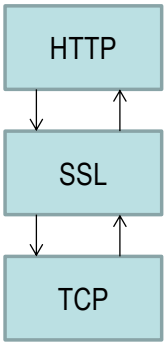
- Basic authentication vulnerable to interception
- Better to use digest authentication
  - Parameters given by server realm, nonce
  - Quality of protection (qop) to use advanced algorithms



***

Secure Socket Layer (SSL)

- HTTP security services do not provide for encryption
- SSL is available to all applications that use TCP for transport
- HTTP secured with SSL (https://)
  - Default port 443
- SSL relies on public key cryptography
  - Uses two keys: public and private
  - One key to encrypt, another key to decrypt
  - Client uses public key to encrypt



SSL (cont'd)

- Requires certificate authorities and public key certificates
  - Also needs public keys of the certificate authority
  - Often preloaded to end systems or software
- Security services provided:
  - Authentication
  - Message integrity
  - Confidentiality
- Can be used to authenticate both client and server
  - Often only used for authentication of server
  - Successful authentication is followed by HTTP transfer

SSL Handshake

- SSL handshake
- (1) Client proposes a series of security algorithms for the communication 'client hello'
- (2) Server selects SSL version and the security algorithms 'server hello'
- (3) Server sends its certificate and the client is responsible to validate it
- (4) Server 'hello done' indicates it's done with initial SSL negotiation
- (5) 'Client Key Exchange' contains cryptographic keys to be used for encryption (encrypted with public key)
- (6) 'Change Cipher Spec' signal encryption using cryptographic keys
- (7) 'Finished' message tests decryption using shared crypto keys
- …

Client Hello

- Client identify the versions of SSL that it supports
  - Ideally uses the latest secure version
- Proposes a series of security capabilities it would like to employ for the communication
  - Security capabilities are known as *Cipher Suites*
  - Identify parameters such as specific cryptographic algorithms and encryption key sizes

Server-Side Messages

- Server responds with a Server Hello message
  - Selects both the SSL version and the security capabilities
  - Chosen from those proposed by the client
- The Server sends a Certificate message
  - Carries its public key certificate
  - Client should ensure that this certificate is valid, that it was issued by a trusted authority, and that it identifies the intended server
- Sends a Server Hello Done message to indicate that it has finished its part of the initial SSL negotiation

Key Exchange

- Client responds with a Client Key Exchange message
  - Contains cryptographic keys to be used to encrypt the data
  - The keys are encrypted using the server's public key, so that only the server is able to decipher and retrieve these keys
- Client sends a Change Cipher Spec message
  - Signal that the client will encrypt all subsequent communications using the cryptographic keys
- Client sends a Finished message, encrypted according to the negotiated cryptographic keys and algorithms
  - The server's ability to successfully decrypt this message ensures that the negotiation has been successful

Server Change Cipher

- The server sends its own Change Cipher Spec message
  - Similar to the client's, it signals that future messages will be encrypted with cryptographic keys
- Concludes the SSL negotiation with a Finished message of its own which
  - Similar to the client's, it is encrypted according to the negotiated parameters
  - Once the client has successfully decrypted this message, it is assured that the negotiation has succeeded

***

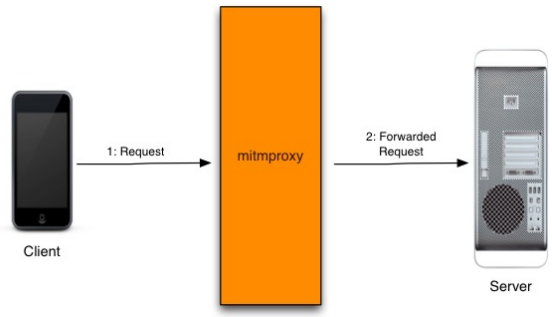
MITM

- Man-in-the-Middle technique
- HTTP proxy mechanism
- Pretend to be server to the client and client to the server
- Can decrypt and inspect the secure communication
- Act as a certification authority (CA) to the client
- Reference:

  https://docs.mitmproxy.org/stable/concepts-howmitmproxyworks/

Explicit HTTP/HTTPS

- Explicit HTTP:
  - GET http://example.com/index.html HTTP/1.1



- Explicit HTTPS
  - CONNECT example.com:443 HTTP/1.1
  - Secure pipe to the server

Complications (1)

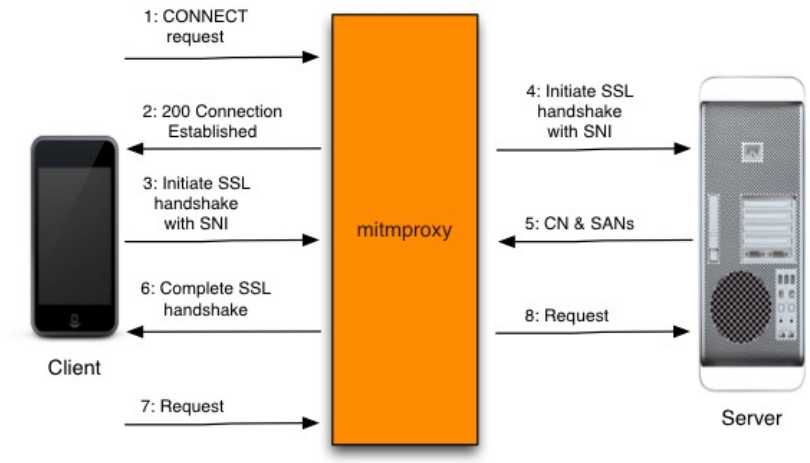
- What's the remote hostname?
  - CONNECT 10.1.1.1:443 HTTP/1.1
- SSL connections are frozen by the proxy
- Proxy uses TLS to connect to the destination to "sniff" the server certificate
  - Common Name, i.e., FQDN
  - Subject Alternative Names, an optional field to specify arbitrary number of alternative domains
- Uses the names to generate a fake certificate and sends to the client

Complications (2)

- Server Name Indication (SNI)
- SSL extension to allow multiple certificates use the same IP address
- Client specifies the remote server name at the start of the SSL handshake
- Server selects the right certificate for the negotiation
- SNI breaks the upstream “sniffing” technique

MITM Proxy

- Let SSL handshake to continue until just after the SNI value is seen by the proxy



***

HTTP/2

- End users or web developers don’t need to know about HTTP/2
- Browsers and applications continue to work the same way, sending and receiving HTTP/1 requests and replies
- However, understanding it could help enhance performance for applications that use HTTP by leveraging the protocol features
- Uses the same scheme (http:// or https://) and port
- Often https negotiation is used to establish “h2” connection between client and server

HTTP 2 Highlights

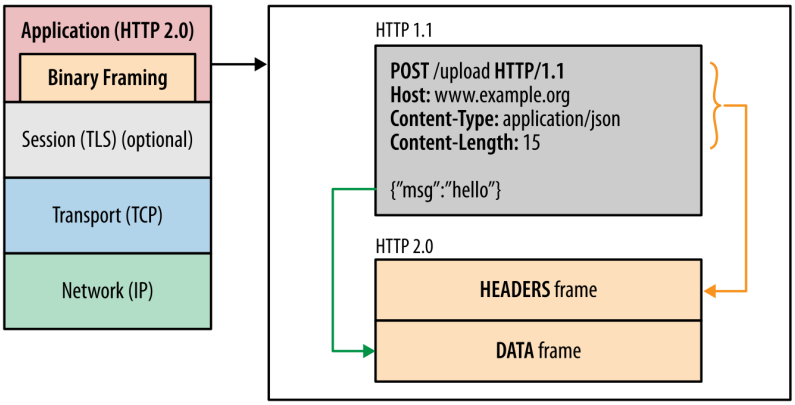
- Binary framing: headers (control) frame and data frame
- Streams, messages and frames
  - Frames: smallest binary unit
  - Stream: bidirectional flow of message in a TCP connection
  - Frames of different streams can interleave over TCP
- Header compression: HPACK
  - Many headers repeat across requests/responses (e.g., cookies)
- Flow control for individual “streams” over TCP: credit-based
- Stream prioritization
- Server push embedded resources
- More …

Binary Frames

- For more efficiency, HTTP/2 is a binary protocol, where HTTP messages are split and sent in defined frames
- Framing usually is done by the lower-level client or libraries used in web browser or web server
- Both chunked encoding and pipelining have head-of-line (HOL) blocking issues
  - The message at the head of the queue prevents subsequent replies from being sent
  - HTTP2: replies can be intermingled on the same TCP connection
- HTTP/1 messages and transfers are supported
  - Some exceptions, e.g., the chunked transfer encoding MUST NOT be used in HTTP/2

HTTP 2 Highlights

Binary framing: headers (control) frame and data frame

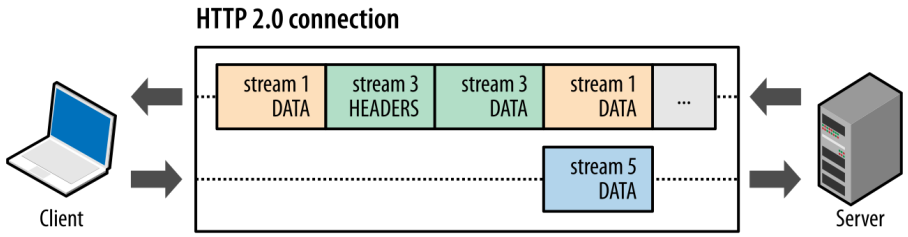


Streams

- HTTP/1 is a synchronous, single request-and-reply protocol
- HTTP/2 allows multiple requests to be in progress at the same time, on a single connection
- Uses a different stream for each HTTP request and reply : request/replay to stream
- Stream identifier in each frame allows multiplexing of streams
  - Client requests use odd number stream IDs
  - Server-initiated pushes use even numbers
- Streams are prioritized by dependency and weight
- Streams are also cancelable

HTTP 2 Highlights

Streams, messages and frames



Priority and Flow Control

- There can be “critical” resources, e.g., HTML, render blocking CSS, and critical JavaScript
- Request critical resources first and then the other items, such as images and asynchronous JavaScript
- Requests are queued in HTTP/1, waiting for a free TCP connection, and the queuing, managed by the browser, decides the priority
- Requests don’t need to be queued by the browser
  - Most libraries allow 100 active streams by default
  - Much more concurrency!

Priority and Flow Control

- For performance, control the interleaving and delivery of frames from different streams
  - Each stream may be assigned an integer weight (1 – 256)
  - Each stream may be given an explicit dependency on another stream
  - Defines a “prioritization tree”
- Weights and dependencies can be changed at any time, e.g., in response to user interaction and other events
- These indicate “preferences” from the client to the server
  - Not required for the server to comply
  - E.g., cannot block server from progress on lower-priority by requiring a higher-priority stream that is blocked

Priority and Flow Control

- HTTP/2 supports flow control a stream level
  - Orthogonal to TCP flow control
- For example, a particular stream can be paused by the user, while allowing other streams to continue and be downloaded

Header Compression

- HTTP/1x provide for compressing the payload but not headers
- Sent messages (requests and replies) carry a lot of repetition
- Many headers don't change during a session
  - E.g.: Cookie, User-Agent, Host, Accept, Accept-Encoding
- Also, some headers can be large compared to the message body
- The new compression algorithm works across multiple messages

Server Push

- Allows the server to reply to a request with more than one message
  - E.g., CSS and JavaScript of used in a page
- Intended to reduce the latency to process a page at the client
- Care must be given to avoid pushing content that is not needed by the client, wasting bandwidth
  - Also, content may be already cached at the client from previous requests