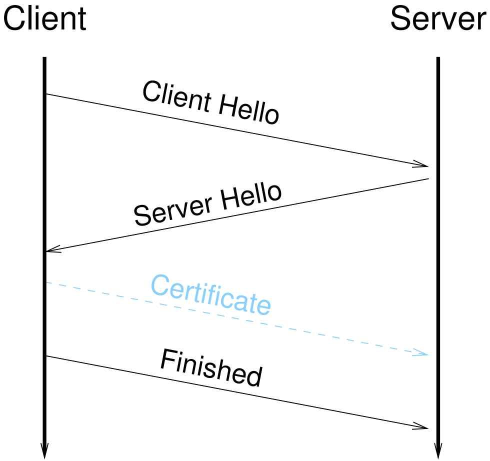
TLS; Web Security 1

TLS Architecture

TLS

- There are several versions of TLS—we'll discuss 1.3, the newest
- There are many variations, options, etc.; we'll stick with the main flow

Data Flow

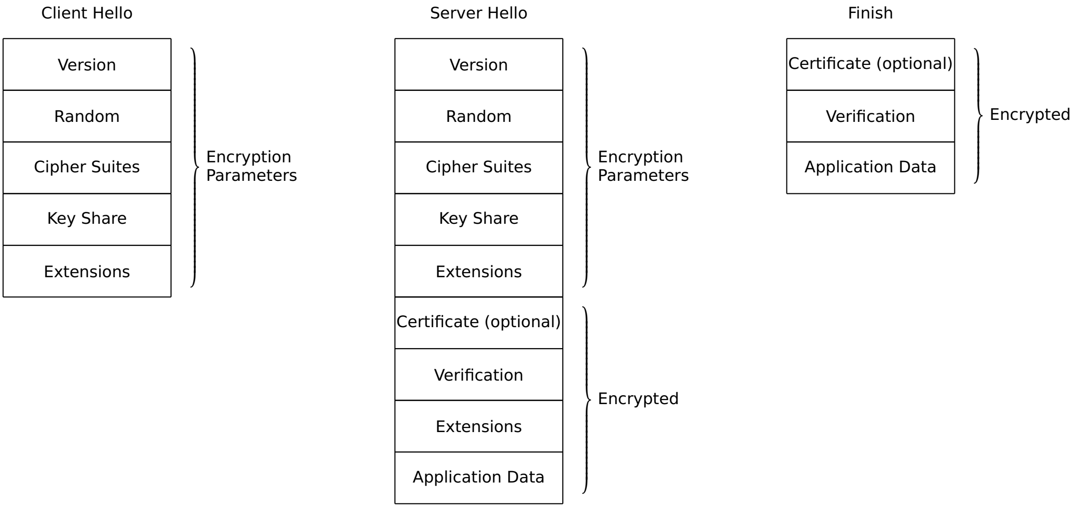


Client
Server
Client Hello
Server Hello
Certificate
Finished

Structure

- Handshake protocol
  - Negotiate version
  - Negotiate encryption parameters
  - Authenticate the connection
  - Create a session key
- Record layer
  - Send messages rather than TCP's simple byte stream
  - Encrypt each message
  - Authenticate each message
  - Detect tampering

Messages


Client Hello
Version
Random
Cipher Suites
Key Share
Extensions
Encryption Parameters
Server Hello
Version
Random
Cipher Suites
Key Share
Extensions
Encryption Parameters
Certificate (optional)
Verification
Extensions
Application Data
Encrypted
Finish
Certificate (optional)
Verification
Application Data
Encrypted

Client Hello

- Random bytes for session key generation
- An offered set of cipher suites
- A Diffie-Hellman exponential
- Optional fields

Server Hello: Plaintext

- Random bytes for session key generation
- The selected cipher suite
- A Diffie-Hellman exponential

Server Hello: Encrypted

- Certificate
- Digital signature on everything it has sent
- Optional fields
- Application data—but the client has not yet authenticated its side

Finish

- Optional certificate
- Digital signature on everything it has sent
- Application data

Many More Options

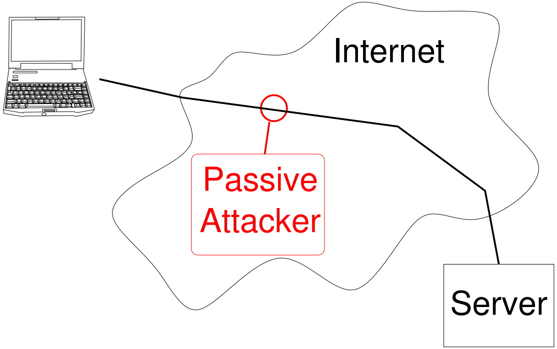
- Session resumption
- Putting a password (“pre-shared key”) in the encryption parameters exchange
- No certificates in either direction
- Simpler (but slightly less secure) key setup
- Downgrade protection
- More. . .

Surveillance Protection

- Note the two Diffie-Hellman exponentials
- They allow encryption to start—between unauthenticated parties—before either side identifies itself
- Certificates are sent after encryption has started
- The protection is imperfect—but it's often good enough

Passive Attackers

- Can eavesdrop on traffic
- Example: tap fibers (yes, that's possible) or hack into other devices
- Does not modify traffic in any way
- Completely blocked by unauthenticated Diffie-Hellman

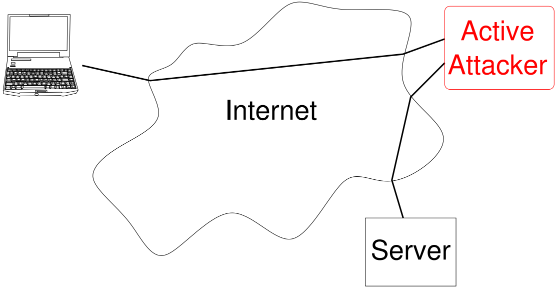


Active Attackers

- Reroutes traffic
- Can play monkey-in-the-middle with unauthenticated Diffie-Hellman:

$$\begin{aligned} A \rightarrow E: &\quad g^{r_A} \bmod p \\ E \rightarrow A: &\quad g^{r_{E_1}} \bmod p \\ E \rightarrow B: &\quad g^{r_{E_2}} \bmod p \\ B \rightarrow E: &\quad g^{r_B} \bmod p \end{aligned}$$

  E then calculates $g^{r_A r_{E_1}} \bmod p$ and $g^{r_{E_2} r_B} \bmod p$ and relays data between A and B, recording it all
- But: the verification step will catch this



Server Name Indication

- Many web servers, especially those run by hosting companies, contain multiple web sites
- Each such web site needs its own certificate
- The client can indicate which site it wants in a Hello message extension, but that is not encrypted—and is visible to censors
- There is work being done to encrypt this, but it's hard

Using TLS

Using TLS

- There are many different TLS implementations—Microsoft has its own, Apple has its own, and there are several open source implementations, notably OpenSSL
- The complexity of the protocol means that implementations *cannot* be simple
- The APIs cannot be simple, either
- But there are some common concepts

Negotiation

- There are things that have to be negotiated, e.g., TLS version and cipher suites
- This means that the applications on either end have to supply their lists
- In many situations, e.g., web servers, site administrators have to be able to control this—which means that the application programmers have to honor their wishes and not rely on defaults
- Example: when there was a new attack on RC4, web sites needed to disable it *before* a new release of the software

Options

- TLS has more than 20 options
- Most aren't used most of the time—but the API has to allow their use
- Again, this is unavoidable complexity

The Record Layer

- TLS programs can't just write to and read from a network socket
- Instead, messages need to be encrypted when being sent, and decrypted and verified on receipt
- The TLS record layer does this—but this means that applications need to speak to it, too

Contexts

- OpenSSL uses *contexts* for the encryption layer and the record layer
- (We saw this concept in SHA2-256)
- In an object-oriented language, each context would be an instantiation of a class—but C isn't object-oriented, so they're simply `structs`
- It is necessary to link the encryption context and the record layer context

Example Setup

```
const SSL_METHOD* method = TLSv1_2_client_method();
if (NULL == method) report_and_exit("TLSv1_2_client_method...");

SSL_CTX* ctx = SSL_CTX_new(method);
if (NULL == ctx) report_and_exit("SSL_CTX_new...");

BIO* bio = BIO_new_ssl_connect(ctx);
if (NULL == bio) report_and_exit("BIO_new_ssl_connect...");
```

From
https://opensource.com/article/19/6/cryptography-basics-openssl-part-1

The Web PKI

Trust Anchors

- Recall that certificates are ultimately issued by a CA
- There isn't The One True Certficate Authority for all possible uses
- TLS programs that use certificates have to supply the proper root
- But for the web, it's more complicated than that. . .

The Web PKI

- Who is the CA for the web?
- There isn't one! Rather, there are many
- That causes problems. . .

Why Are There Many Web CAs?

- If there were just one, it would be a single point of failure—and control—for the entire web
- As a matter of national policy, some countries do not do not want CAs for their organizations to be in other countries
- Even government web sites use certficates from commercial CAs
- Besides, it's better to avoid monopolies when possible

Subject Name	
Country	US
State/Province	District of Columbia
Locality	Washington
Organization	Library of Congress
Common Name	*.loc.gov
Issuer Name	
Country	US
Organization	Entrust, Inc.
Organizational Unit	See www.entrust.net/legal-terms
Organizational Unit	(c) 2012 Entrust, Inc. - for authorized use only
Common Name	Entrust Certification Authority - L1K

It’s a Forest, Not a Tree

- Conceptually, a PKI is a tree: the CA is the root, it can create multiple intermediate CAs, they issue certificates, etc.
- The web has multiple CAs, each of which is a tree
- Any of these CAs can issue a certificate to any web site
- Yes, that can cause problems

Consistency Doesn't Matter

Common Name www.cia.gov

Issuer Name
Country US
Organization DigiCert Inc
Organizational Unit www.digicert.com
Common Name DigiCert SHA2 Extended Validation Server CA

Subject Name
Common Name www.defense.gov

Issuer Name
Country US
Organization Let's Encrypt
Common Name Let's Encrypt Authority X3

Validity
Not Before 8/25/2020, 1:14:49 PM (Eastern Daylight Time)
Not After 11/23/2020, 12:14:49 PM (Eastern Daylight Time)

Subject Name
Country US
State/Province District of Columbia
Locality Washington
Organization Library of Congress
Common Name *.loc.gov

Issuer Name
Country US
Organization Entrust, Inc.
Organizational Unit See www.entrust.net/legal-terms
Organizational Unit (c) 2012 Entrust, Inc. - for authorized use only
Common Name Entrust Certification Authority - L1K

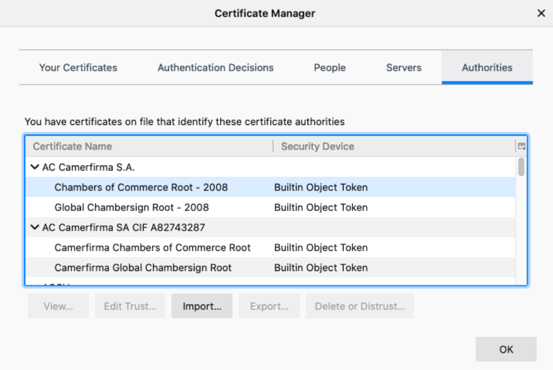
Issuer Name
Country US
Organization Cloudflare, Inc.
Common Name Cloudflare Inc ECC CA-3

Validity
Not Before 8/6/2020, 8:00:00 PM (Eastern Daylight Time)
Not After 8/7/2021, 8:00:00 AM (Eastern Daylight Time)

Subject Alt Names
DNS Name sni.cloudflaressl.com
DNS Name www.fbi.gov

Who Picks these CAs?

- Each browser or OS vendor decides for itself which CAs to trust
- There's a large, common set
- Most follow the standards set by the CA Browser Forum



Misbehaving

- What if a CA misbehaves?
- Can it issue bogus certificates, when the real cert for a site was issued by a different CA?
- Yes!
- It's happened several times, e.g., the Comodo and DigiNotar hacks
- Google designed a distributed cryptographic logging protocol (certificate transparency) to detect such incidents

Web Server Security

Web Security: It’s Not Just Encryption

- Encryption and certificates are very important
- However, it’s far from the biggest issue, precisely because we’ve had SSL and TLS since 1995
- The bigger issues: configuration and code

Web Server Architecture

- Static files
- Programs and scripts
- Infrastructure
- Generally, databases

☞ All can present issues

Web Server Infrastructure

- Several aspects
  - Configuration files
  - Certificates and keys
  - Log files
  - The server itself
  - Other executables
  - More...
- Each of these have security implications
- Protections vary

Protecting Keys

- The server's private key is precious—must be protected
- Primary desired property: confidentiality
- One of the biggest risks to the key is the scripts that serve up pages
- Solutions: HSMs or use of the operating system's permission mechanisms
- (More on that in a few weeks)

Configuration Files

- Primary desired property: integrity
- These files control what the server will hand out; if they're tampered with, erroneous (or sensitive) files may be returned
- Again, we must rely on the OS

Log Files

- Primary desired property: integrity
- Log files are vitally important, both for normal operation, normal errors, and intrusion analysis
- For intrusion analysis, best to keep them on a separate computer—why?

Log Files

- Primary desired property: integrity
- Log files are vitally important, both for normal operation, normal errors, and intrusion analysis
- For intrusion analysis, best to keep them on a separate computer—why?
- So that a successful attacker can't cover their tracks by erasing the log

Serving Up Files

- Conceptually, a web server returns elements of a tree
- In fact, URLs appear to contain filenames
- It is generally not a single subtree of the file system
- Part of the web server configuration determines which file system directories correspond to which part of the URL name space
- Must be careful to offer only the proper files

Programs and Scripts

- All of that is the easy part of web security
- What makes the modern web interesting is *scripts*: programs that consult databases and generate web pages dynamically
- Ensuring that these programs are correct is the hardest part of web security
- Why?

Script Correctness

- Web scripts run in an extremely hostile environment—they *must* be exposed to the outside world and cannot be protected by firewalls
- Attackers can send them arbitrary input
- Program correctness is probably the hardest problem in computer science—and every real web site has to run many such programs

SQL Injection

- Suppose a program is querying an SQL database based on valid userID and query string:

```
snprintf(buf, sizeof buf, "select where user=\"\%s\" &&
    query=\"%s\"", uname, query);
```

SQL Injection

- Suppose a program is querying an SQL database based on valid userID and query string:

```
snprintf(buf, sizeof buf, "select where user=\"\%s\" &&
    query=\"%s\"", uname, query);
```

- What if query is

```
foo" || user="root
```

SQL Injection

- Suppose a program is querying an SQL database based on valid userID and query string:

```
snprintf(buf, sizeof buf, "select where user=\"\%s\" &&
    query=\"%s\"", uname, query);
```

- What if query is

```
foo" || user="root
```

- The actual command passed to SQL is

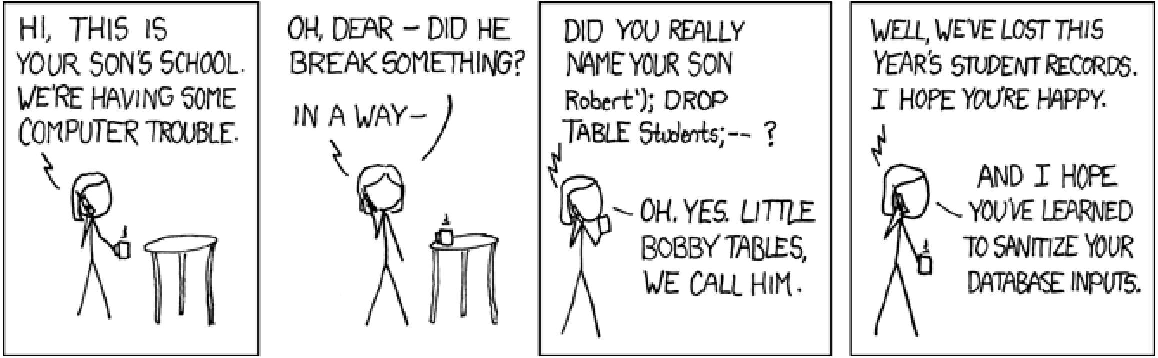
```
select where user="uname" && query = "foo" ||
    user="root"
```

- This will retrieve records it shouldn't have
- Variants on this are one of the biggest causes of web site penetration

What Was Wrong?

- The program was passing a *string* to the database
- The enemy controlled part of the string
- The program didn't make sure that the substitution was safe

Exploits of a Mom



(From https://xkcd.com/327/)

Invoking Programs

- The same sort of thing can happen if external programs are invoked
- Contrast

```
snprintf(buf, sizeof buf, "ls %s", dirname);
system(buf);
```

  with

```
execl("/bin/ls", "ls", dirname, NULL);
```

- What is the difference?

Invoking Programs

- The same sort of thing can happen if external programs are invoked
- Contrast

  ```
  snprintf(buf, sizeof buf, "ls %s", dirname);
  system(buf);
  ```

  with

  ```
  execl("/bin/ls", "ls", dirname, NULL);
  ```

- What is the difference?
- The first example has the same problem as the SQL statement

Filename Parsing

- User supplies pathname; application must check for validity
- Administrator specifies list of accessible files and/or directories
- Sometimes, wildcards—*, ?, and more—are permitted
- Application must *parse* supplied filename
- Remarkably difficult

The ".." Problem

- Attackers try to get at other files
- Simplest attack: put `..` in the path
  `http://example.com/../../../../etc/passwd`
- The `..` can occur later:
  `http://example.com/a/b/../../../../etc/passwd`
- If directory `/dir` is legal, what about `/dir/../dir/file`? Do you want to count levels?
- Watch out for `/dir///../../file`—replicated `/`'s counts as a single one
- Note that `/foo..bar/bletch` is legal
- This problem has been known for about 40 years—and I still see it pop up every year or two

URL Syntax Issues

- Example: in URLs, %xx can specify two hex digits for the character. `%2F` is the same as /
- When is that expanded?
- How is `/foo%2F..%2Fetc/passwd` processed?

Unicode

- Standard for representing (virtually) all of the world's scripts
  ☞ There are proposals for Klingon and Tengwar ("Elvish") codepoints
- *Many* problems!
- Some symbols look the same, but have different values: ordinary /—technically called "solidus"—is U+002F, but U+2044, "fraction slash", looks the same
- "Combining characters" and "grapheme joiners" make life even more complicated. Thus, **á** can be U+00C1 or the two-character sequence U+0041,U+0301
- Comparison rules have to be application-dependent—and watch out for false visual equivalences; these have already been used for attacks, especially with Cyrillic domain names

Cyrillic Homograph Attack on “Paypal”

Glyph	Unicode value in Cyrillic
P	U+0420
a	U+0430
y	U+0443
p	U+0440
a	U+0430
l	U+006C (ASCII)

Access Control

- Many different forms
- Many different types of authentication
- Many interactions

Explicit Access Control

- Access control lists settable by the webmaster for any directory tree
- Passwords or certificates can be configured as well
- Permission can be granted or withheld based on client IP address
- If a directory has no `index.html` file, should the web server just list its contents?
- Applications can do their own authentication and access control
- All of these interact; combinations can be used

A Sample Configuration

Here is a `.htaccess` file for a directory:

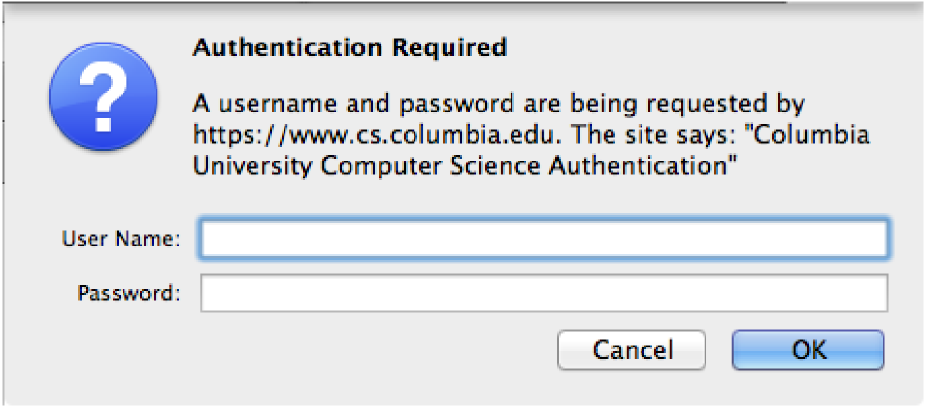
```
<Files *>
    AuthUserFile /home/smb/pwdir/.htpasswd
    AuthGroupFile /dev/null
    AuthName "File Access"
    AuthType Basic
    Require valid-user
</Files>
```

The string `File Access` is displayed to the user. Logins and passwords are stored in `/home/smb/pwdir/.htpasswd`.

Web Authentication

A web password file:

```
user1:e03rzWPNjjZFo
user2:CqkaeLJSVcRpI
```



That's Rarely Used—Why?

- No site-specific display
- No error recovery, e.g., a link for "I forgot my password"
- Too restrictive—no good option for partial display, e.g., of a news article
- A simple linear file doesn't scale up very well
- Web sites generally implement their own authentication

Questions?

(Great blue heron, Morningside Drive, February 16, 2020)