EECS 440 System Design of a Search Engine

Winter 2021

Lecture 6: SSL

Nicole Hamilton
https://web.eecs.umich.edu/~nham/
nham@umich.edu

Agenda

1. Course details.
2. Correction on the Host: parameter.
3. Review of reading an HTTP webpage.
4. Read an HTTPS webpage.
5. TLS, SSL and the OpenSSL library.
6. LinuxGetSSL.

Agenda

1. Course details.
2. Correction on the Host: parameter.
3. Review of reading an HTTP webpage.
4. Read an HTTPS webpage.
5. TLS, SSL and the OpenSSL library.
6. LinuxGetSSL.

details

1. Group photos due Feb 8 (tonight) and I’ll want to meet with each group shortly after.
2. Project plans due Feb 14.
3. Homework 3 due date TBD.

Agenda

1. Course details.
2. Correction on the Host: parameter.
3. Review of reading an HTTP webpage.
4. Read an HTTPS webpage.
5. TLS, SSL and the OpenSSL library.
6. LinuxGetSSL.

Host: parameter

Lots of servers host lots of websites at the same IP address and port number.

They distinguish which website you mean by the Host: parameter.

So, it’s not redundant.

My websites are on a GoDaddy machine with lots of other websites.

```
tcsh-2% ./LinuxGetSsl https://nicolehamilton.com
Service = https, Host = nicolehamilton.com, Port = , Path =
Host address length = 16 bytes
Family = 2, port = 443, address = 160.153.46.5
GET / HTTP/1.1
Host: nicolehamilton.com
User-Agent: LinuxGetSsl/2.0 nham@umich.edu (Linux)
Accept: */*
Accept-Encoding: identity
Connection: close


HTTP/1.1 200 OK
Date: Thu, 19 Sep 2019 17:02:10 GMT
Server: Apache
Upgrade: h2,h2c
Connection: Upgrade, close
Last-Modified: Thu, 11 Oct 2018 21:59:57 GMT
ETag: "c4206db-3f6f-577fb177483a1"
Accept-Ranges: bytes
Content-Length: 16239
:
```

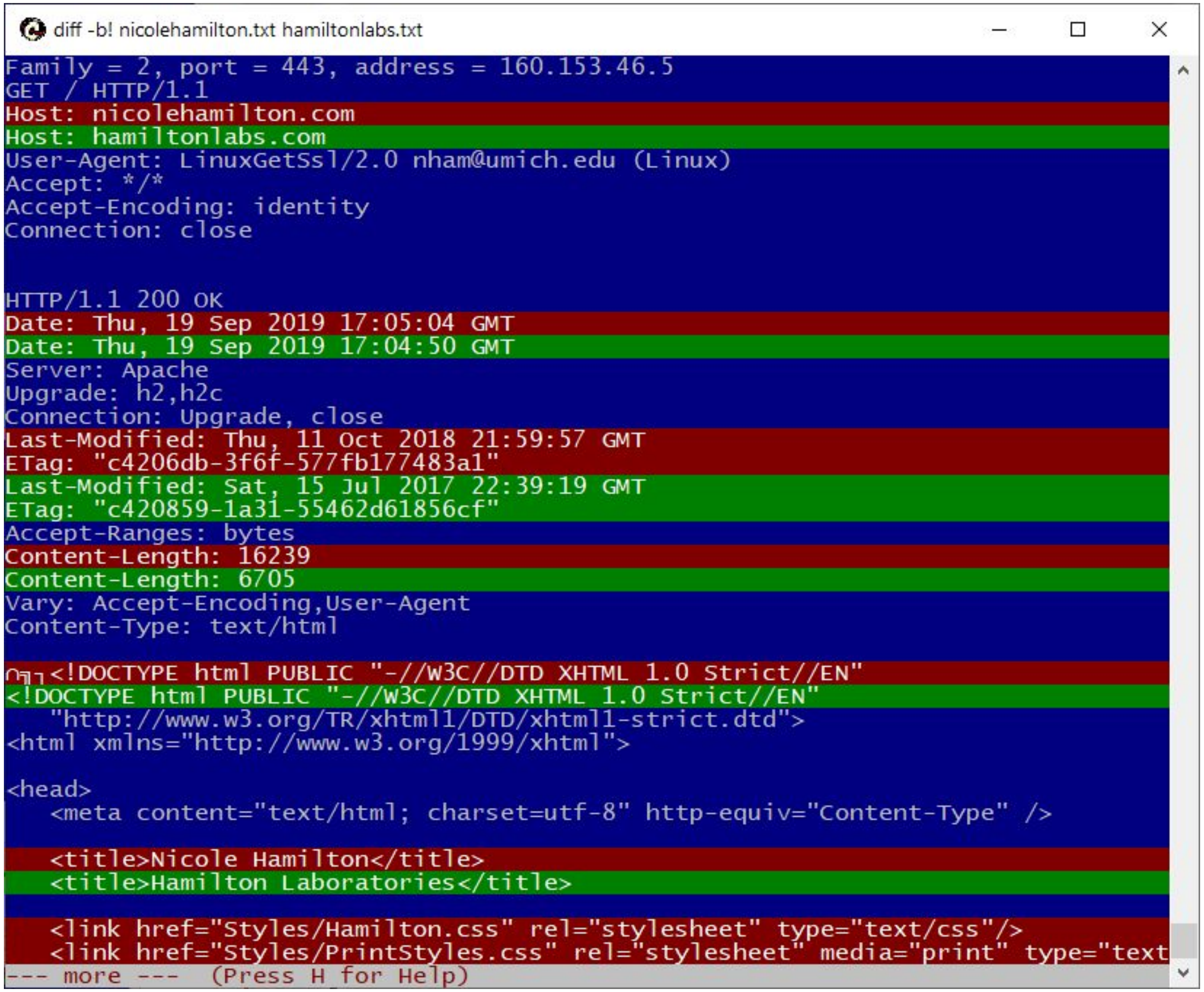
So, both nicolehamilton.com and hamiltonlabs.com are at 160.153.46.5:443.

```
tcsh-2% ./LinuxGetSsl https://nicolehamilton.com
Service = https, Host = nicolehamilton.com, Port = , Path =
Host address length = 16 bytes
Family = 2, port = 443, address = 160.153.46.5
GET / HTTP/1.1
Host: nicolehamilton.com
User-Agent: LinuxGetSsl/2.0 nham@umich.edu (Linux)
Accept: */*
Accept-Encoding: identity
Connection: close


HTTP/1.1 200 OK
Date: Thu, 19 Sep 2019 17:02:10 GMT
Server: Apache
Upgrade: h2,h2c
Connection: Upgrade, close
Last-Modified: Thu, 11 Oct 2018 21:59:57 GMT
ETag: "c4206db-3f6f-577fb177483a1"
Accept-Ranges: bytes
Content-Length: 16239
:
```

Both nicolehamilton.com and hamiltonlabs.com are at 160.153.46.5:443.

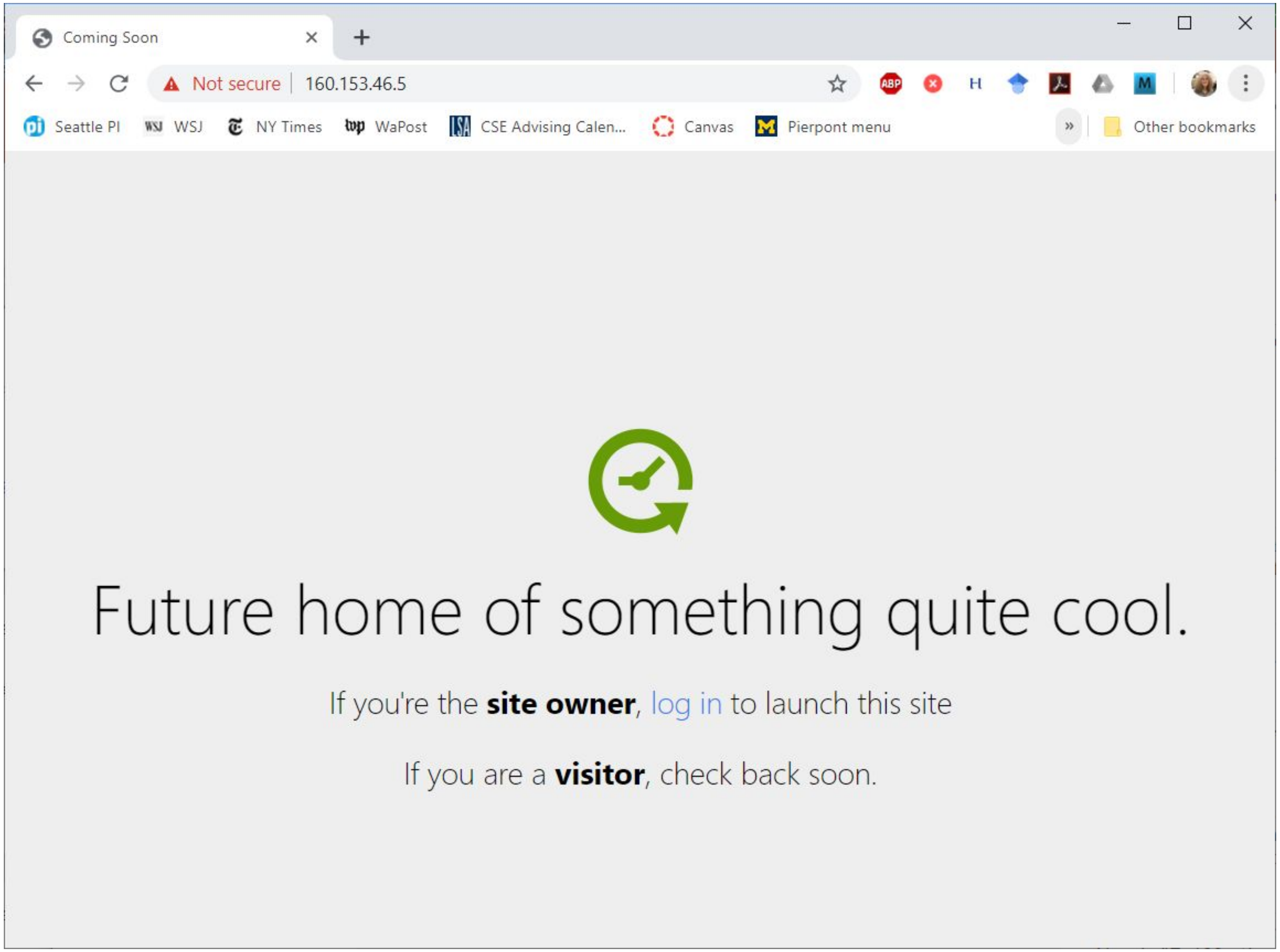
```
tcsh-3% ./LinuxGetSsl https://hamiltonlabs.com
Service = https, Host = hamiltonlabs.com, Port = , Path =
Host address length = 16 bytes
Family = 2, port = 443, address = 160.153.46.5
GET / HTTP/1.1
Host: hamiltonlabs.com
User-Agent: LinuxGetSsl/2.0 nham@umich.edu (Linux)
Accept: */*
Accept-Encoding: identity
Connection: close


HTTP/1.1 200 OK
Date: Thu, 19 Sep 2019 17:03:31 GMT
Server: Apache
Upgrade: h2,h2c
Connection: Upgrade, close
Last-Modified: Sat, 15 Jul 2017 22:39:19 GMT
ETag: "c420859-1a31-55462d61856cf"
Accept-Ranges: bytes
Content-Length: 6705
:
```

The server response depends on which Host: was specified.



```
Family = 2, port = 443, address = 160.153.46.5
GET / HTTP/1.1
Host: nicolehamilton.com
Host: hamiltonlabs.com
User-Agent: LinuxGetSsl/2.0 nham@umich.edu (Linux)
Accept: */*
Accept-Encoding: identity
Connection: close


HTTP/1.1 200 OK
Date: Thu, 19 Sep 2019 17:05:04 GMT
Date: Thu, 19 Sep 2019 17:04:50 GMT
Server: Apache
Upgrade: h2,h2c
Connection: Upgrade, close
Last-Modified: Thu, 11 Oct 2018 21:59:57 GMT
ETag: "c4206db-3f6f-577fb177483a1"
Last-Modified: Sat, 15 Jul 2017 22:39:19 GMT
ETag: "c420859-1a31-55462d61856cf"
Accept-Ranges: bytes
Content-Length: 16239
Content-Length: 6705
Vary: Accept-Encoding,User-Agent
Content-Type: text/html

∩╗┐<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
   <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />

   <title>Nicole Hamilton</title>
   <title>Hamilton Laboratories</title>

   <link href="Styles/Hamilton.css" rel="stylesheet" type="text/css"/>
   <link href="Styles/PrintStyles.css" rel="stylesheet" media="print" type="text
--- more ---  (Press H for Help)
```

If you specify Host: 160.153.46.5, you get GoDaddy's login page for that server.



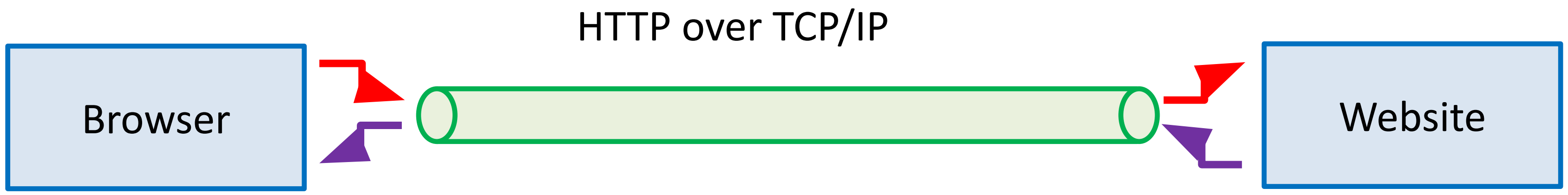
Future home of something quite cool.

If you're the **site owner**, log in to launch this site

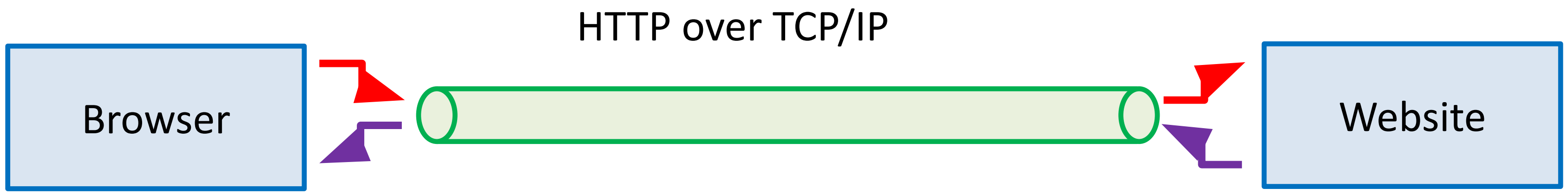
If you are a **visitor**, check back soon.

Agenda

1. Course details.
2. Correction on the Host: parameter.
3. Review of reading an HTTP webpage.
4. Read an HTTPS webpage.
5. TLS, SSL and the OpenSSL library.
6. LinuxGetSSL.



Imagine the connection between a browser and a website as a long pipe.
At each end is a socket you can read or write from as if it was a file.
Anything written into one end pops out and can be read at the other.



To read a page from a website:

1. Look up the TCP/IP address of the website.
2. Create a socket.
3. Connect the socket to that address.
4. Send a GET message to request the page.
5. Read what comes back.

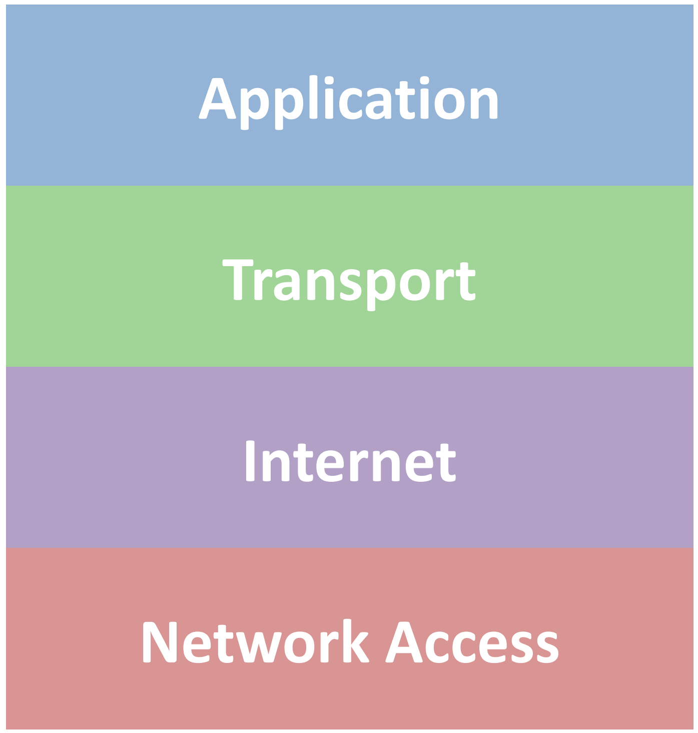
TCP/IP Model

DHCP, DNS, FTP, HTTP, HTTPS, POP, SMTP, SSH, etc.

TCP and UDP

IP address: IPv4 or IPv6

Link level: MAC address
Physical: Cable, fiber, wireless



To get an IP address

1. Parse the HTTPS path to identify the host (domain name) we’re trying to reach.
2. Find the IP address for that host using a Domain Name Server (DNS).

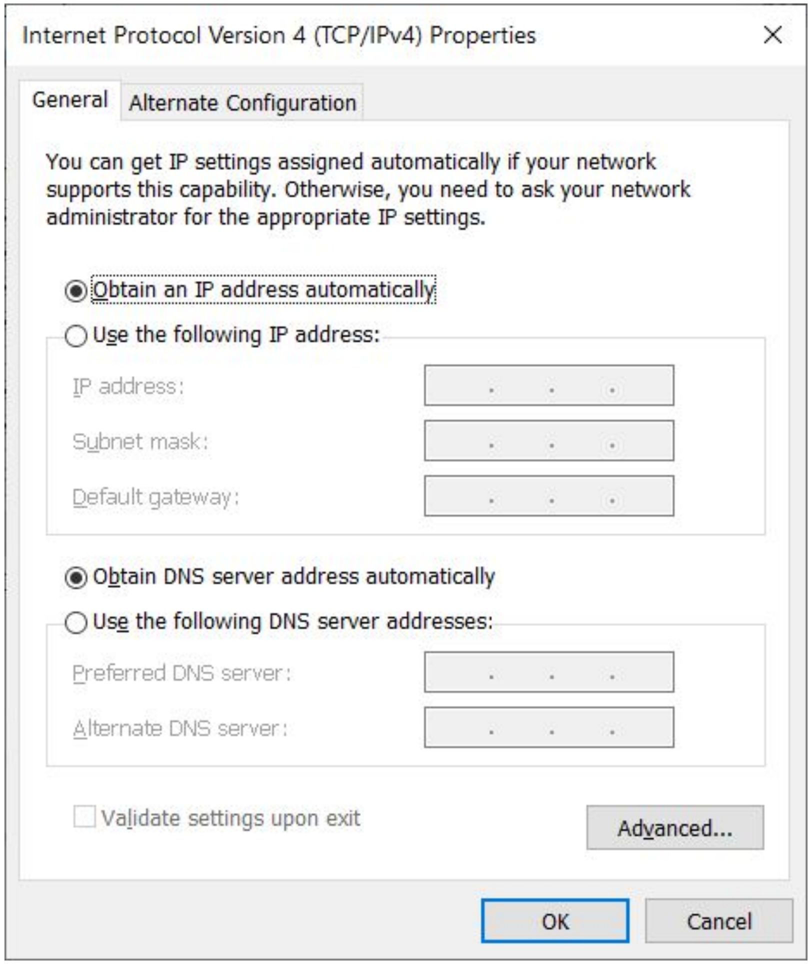
DNS Records

```
Type Name Value     TTL  Actions
A    @    160.153.46.5  600 seconds
A    admin    160.153.46.5  600 seconds
A    mail 160.153.46.5  600 seconds
:
CNAME     webmail   @    1 Hour
CNAME     www  @    1 Hour
:
MX   @    mail.hamiltonlabs.com (Priority: 0)  1 Hour    Edit
NS   @    ns61.domaincontrol.com  1 Hour
NS   @    ns62.domaincontrol.com  1 Hour
SOA  @    Primary nameserver: ns61.domaincontrol.com.    600 seconds
```

An A record defines a host address.
A CNAME record defines a canonical name for alias.
An MX (Mail eXchange) record defines a mail server.
An NS record defines a name server.
An SOA (Start of Authority) defines the primary name server.

DHCP

We usually rely on DHCP (Dynamic Host Configuration Protocol) to assign an IP address to our machine and DNS server.



Internet Protocol Version 4 (TCP/IPv4) Properties

General | Alternate Configuration

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

● Obtain an IP address automatically
○ Use the following IP address:

IP address: . . .
Subnet mask: . . .
Default gateway: . . .

● Obtain DNS server address automatically
○ Use the following DNS server addresses:

Preferred DNS server: . . .
Alternate DNS server: . . .

☐ Validate settings upon exit

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *node, const char *service,
               const struct addrinfo *hints,
               struct addrinfo **res);

void freeaddrinfo(struct addrinfo *res);
```

Given node and service, which identify an Internet host and a service, getaddrinfo() returns one or more addrinfo structures, each of which contains an Internet address that can be specified in a call to bind(2) or connect(2).

Here’s an example use.

```
// Get the host address, supplying hints for
// what we're looking for.

struct addrinfo *address, hints;
memset( &hints, 0, sizeof( hints ) );
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;

int getaddrResult = getaddrinfo( url.Host,
      *url.Port ? url.Port : "80", &hints, &address );
```

Later, it must be freed.

```
freeaddrinfo( address );
```

This is what the addrinfo structure looks like. It contains an Internet address that can be specified in a call to bind(2) or connect(2).

```
struct addrinfo {
   int                ai_flags;
   int                ai_family;
   int                ai_socktype;
   int                ai_protocol;
   socklen_t          ai_addrlen;
   struct sockaddr   *ai_addr;
   char              *ai_canonname;
   struct addrinfo   *ai_next;
};
```

The interesting part is the ai_addr, the actual IP address, which we can print.

```
 PrintAddress( ( sockaddr_in * )address->ai_addr,
      sizeof( struct sockaddr ) );
```

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
int close(int fd);
```

socket() creates an endpoint for communication and returns a file descriptor that can be used for reading and writing.

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
int close(int fd);
```

The domain argument specifies a communication domain. Here are the most common:

Name	Purpose
AF_UNIX, AF_LOCAL	Local communication
AF_INET	IPv4 Internet protocols
AF_INET6	IPv6 Internet protocols

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
int close(int fd);
```

The socket has the indicated type, which specifies the communication semantics. The most common is SOCK_STREAM, a sequenced, reliable connection with two-way byte streams.

The protocol is usually IPPROTO_TCP.

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr,
        socklen_t addrlen);
```

connect() connects the socket to the specified IP address. The addrlen argument specifies the size of addr structure.

A sockaddr * is actually a generic pointer caste from one of several possible address structures, depending on the type of a connection. For an internet connection, you'll actually use a sockaddr_in (an internet sockaddr).

Here’s an example creating a socket and connecting it to an address.

```
// Create a TCP/IP socket.

int s = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );
assert( s != -1 );

// Connect the socket to the host address.

int connectResult = connect( s, address->ai_addr,
      sizeof( struct sockaddr ) );
assert( connectResult == 0 );
```

```c
#include <sys/types.h>
#include <sys/socket.h>

ssize_t send(int sockfd, const void *buf, size_t len, int flags);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

send() writes data into the socket. recv() reads data. Flags allow close-on-exec, noblocking reads/writes and other options.

The only difference between send() and write() or between recv() and read() is the presence of flags. With a zero flags argument, send() is equivalent to write() and recv() is equivalent to write().

Here's a sample GET message we might send. Notice that the User-agent field must contain your contact info in 440.

```
GET / HTTP/1.1
Host: www.nytimes.com
User-Agent: LinuxGetUrl/2.0 nham@umich.edu (Linux)
Accept: */*
Accept-Encoding: identity
Connection: close
```

Some sites will not even respond without a User-Agent field. It's a free text field and can be anything as long as it exists. It's typically the name of the software product that generated the Get + a slash followed by a version number. The OS or build environment is usually given in parens.

The Accept: and Accept-Encoding: fields are not required but typically provided.

Here’s an example reading from a socket and writing to stdout.

```
char buffer[ 10240 ];
int bytes;

while ( ( bytes = recv( s, buffer, sizeof( buffer ), 0 ) ) > 0 )
   write( 1, buffer, bytes );
```

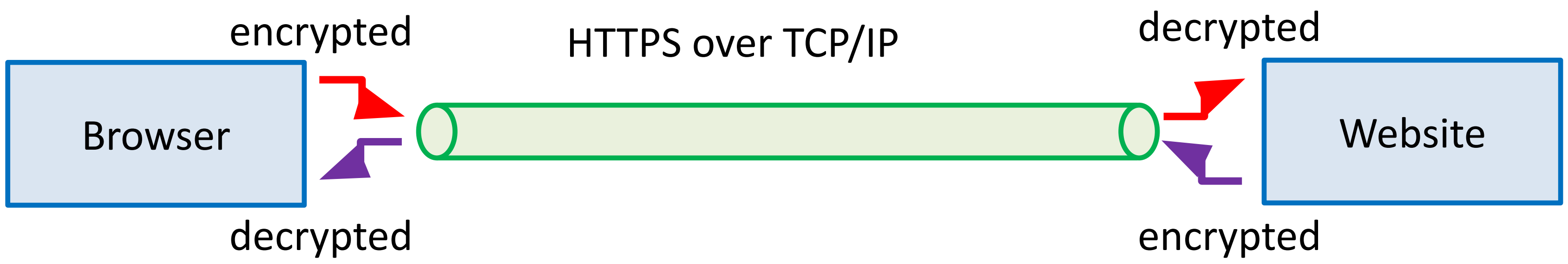
We’ll talk more about read( ) and write( ) when we get to the filesystem.

Here's the entire main( ) for LinuxGetUrl, minus only all the code.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int main( int argc, char **argv )
   {
   // Parse the URL

   // Get the host address.

   // Create a TCP/IP socket.

   // Connect the socket to the host address.

   // Send a GET message.

   // Read from the socket until there's no more data, copying it to
   // stdout.

   // Close the socket and free the address info structure.
   }
```

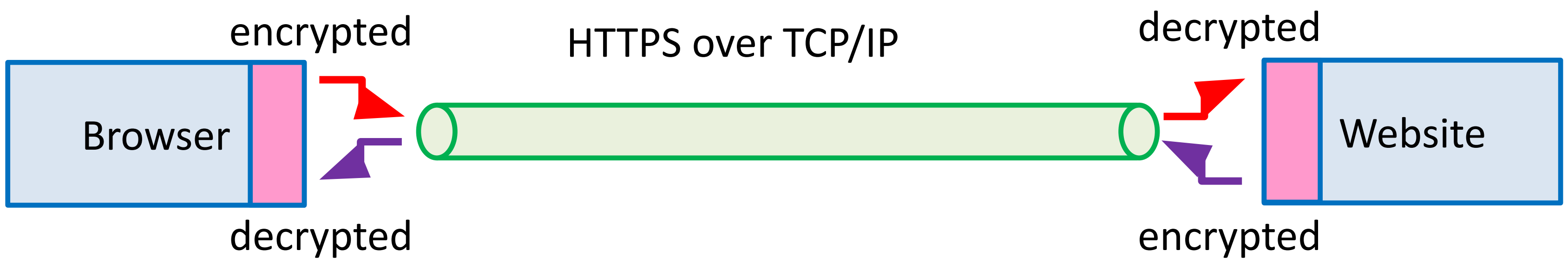
Agenda

1. Course details.
2. Correction on the Host: parameter.
3. Review of reading an HTTP webpage.
4. Read an HTTPS webpage.
5. TLS, SSL and the OpenSSL library.
6. LinuxGetSSL.



Under HTTPS, data is encrypted before being sent and decrypted when received using a public key mechanism that allows both ends to agree on a secret session key.

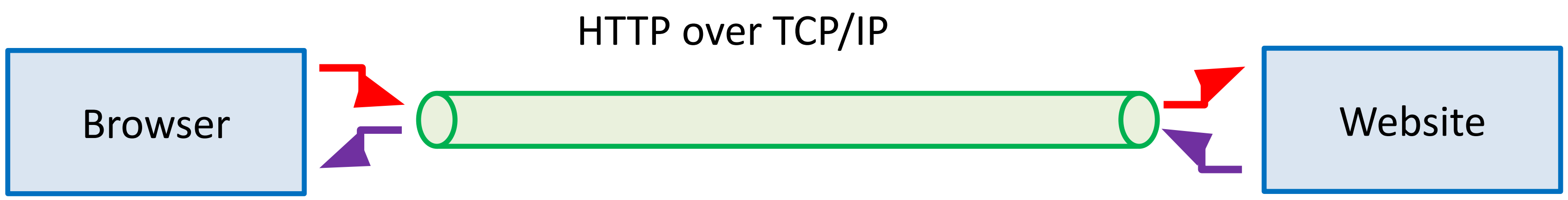
Done using a Secure Socket Layer (SSL) wrapper around a regular socket.



Under HTTPS, data is encrypted before being sent and decrypted when received using a public key mechanism that allows both ends to agree on a secret session key.

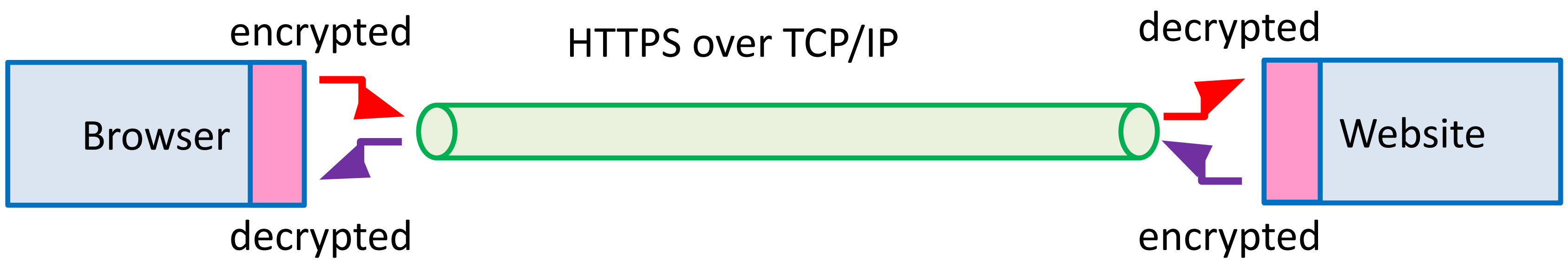
Done using a Secure Socket Layer (SSL) wrapper around a regular socket.

Here, we'll use the OpenSSL library.



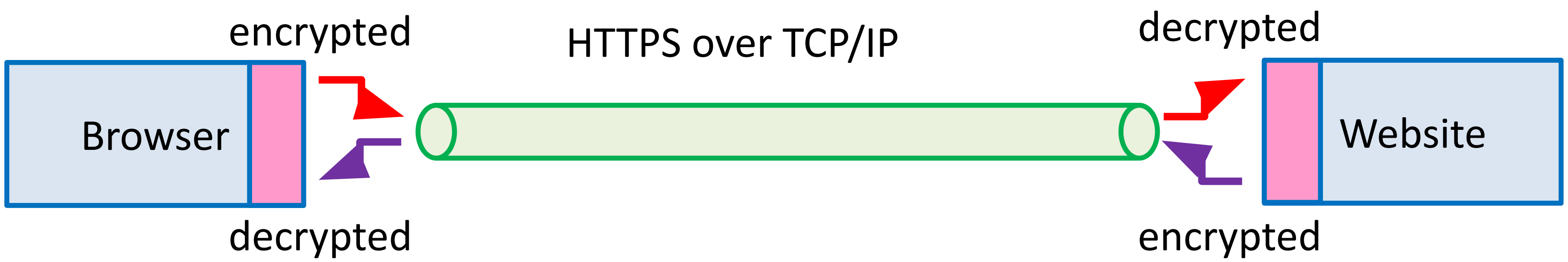
To read a page from a website:

1. Look up the TCP/IP address of the website.
2. Create a socket.
3. Connect the socket to that address.
4. Send a GET message to request the page.
5. Read what comes back.



To read a page from a website:

1. Look up the TCP/IP address of the website.
2. Create a socket.
3. Connect the socket to that address.
4. Build an SSL layer and establish a secure connection.
5. Send a GET message to request the page.
6. Read what comes back.



To read a page from a website:

1. Look up the TCP/IP address of the website.
2. Create a socket.
3. Connect the socket to that address.
4. *Build an SSL layer and establish a secure connection.*
5. Send a GET message to request the page.
6. Read what comes back.

Secret communications

Traditionally, two parties would have to agree on a method and key for secret communications.

A book owned by both parties might be used with messages encrypted as references to page, line and word numbers, *PPPLLWW*.

Mechanical methods like the German Enigma relied on secret hardware and a key.

Problems:

1. You need a way of communicating securely how you'll do it before you can do it.

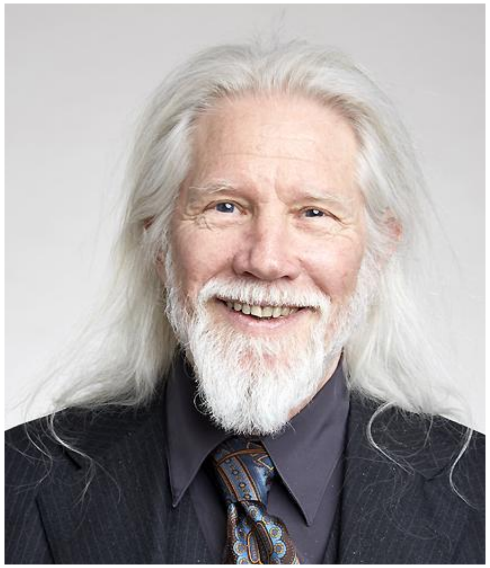
2. Secrecy depends on the secrecy of both the key and the method.

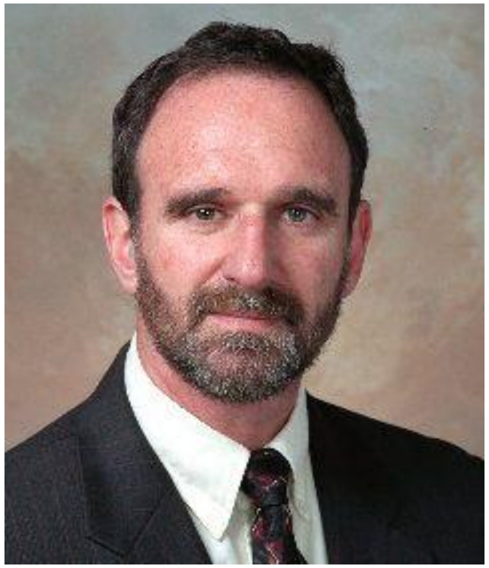
Image source: https://en.wikipedia.org/wiki/Enigma_machine

Insights

1. The security of the system should only depend on secrecy of the key, not the secrecy of the method.
2. It should be possible for anyone to see how messages are encrypted, given the key, but without the key, knowing how it's done isn't helpful in breaking the message.

Diffie-Hellman key exchange, 1976

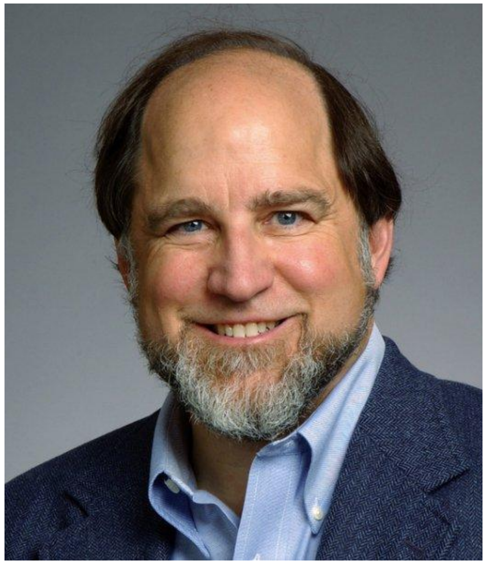

Whitfield Diffie

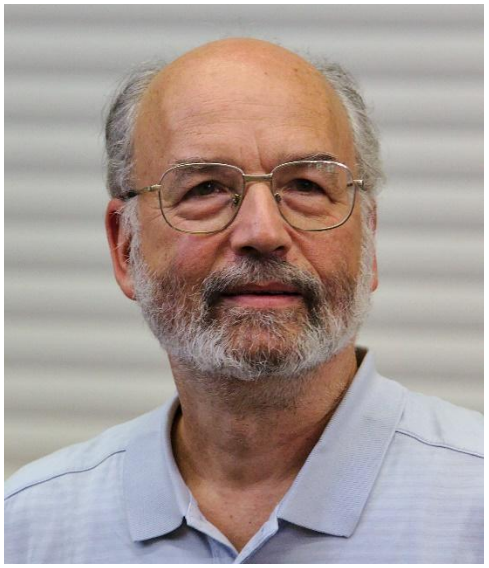

Martin Hellman

Ralph Merkle

Image sources: https://en.wikipedia.org/wiki/Whitfield_Diffie
https://en.wikipedia.org/wiki/Martin_Hellman
https://en.wikipedia.org/wiki/Ralph_Merkle

RSA public key encryption, 1978


Ron Rivest


Adi Shamir

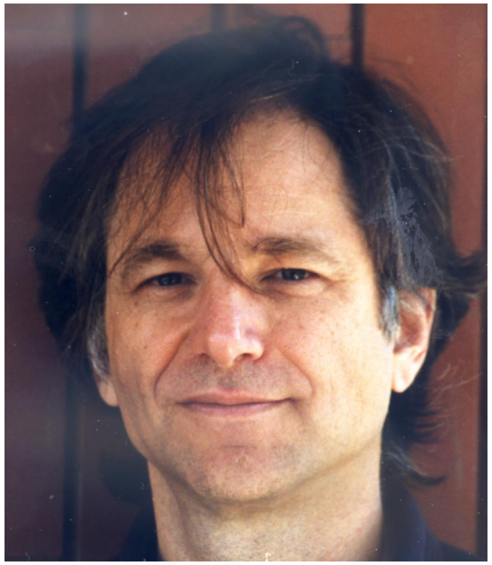

Leonard Adleman

Image sources: https://en.wikipedia.org/wiki/Ron_Rivest
https://en.wikipedia.org/wiki/Adi_Shamir
https://en.wikipedia.org/wiki/Leonard_Adleman

Public key encryption

Uses pairs of private and public keys that are related by a mathematical formula that's hard to reverse, factoring of very large numbers.

To get started:

1. I create (or pick) a private key which I keep secret,
2. then use that to create a public key, which I can share with the world.

If I want to send you an encrypted message:

1. I encrypt using my private key and your public key.
2. You decrypt using my public key and your private key.

Does not require a secure channel for initial exchange of secret keys.

SSL / TLS Handshake

Client

Sends a message to server with a list of cipher suites and SSL/TLS versions it supports.

Chooses a cipher suite and TLS version and sends its certificate and public key.

Verifies certificate, extracts public key, uses it to encrypt a new pre-master key, which it sends to the server.

Uses its private key to decrypt the pre-master key.

Both use the pre-master key to compute a shared secret key.

Sends encrypted message announcing it's switching to encryption with the shared secret.

Decrypts and verifies message, sends message encrypted with the shared secret.

Both use encryption using the shared secret for the rest of the session.

Server

Agenda

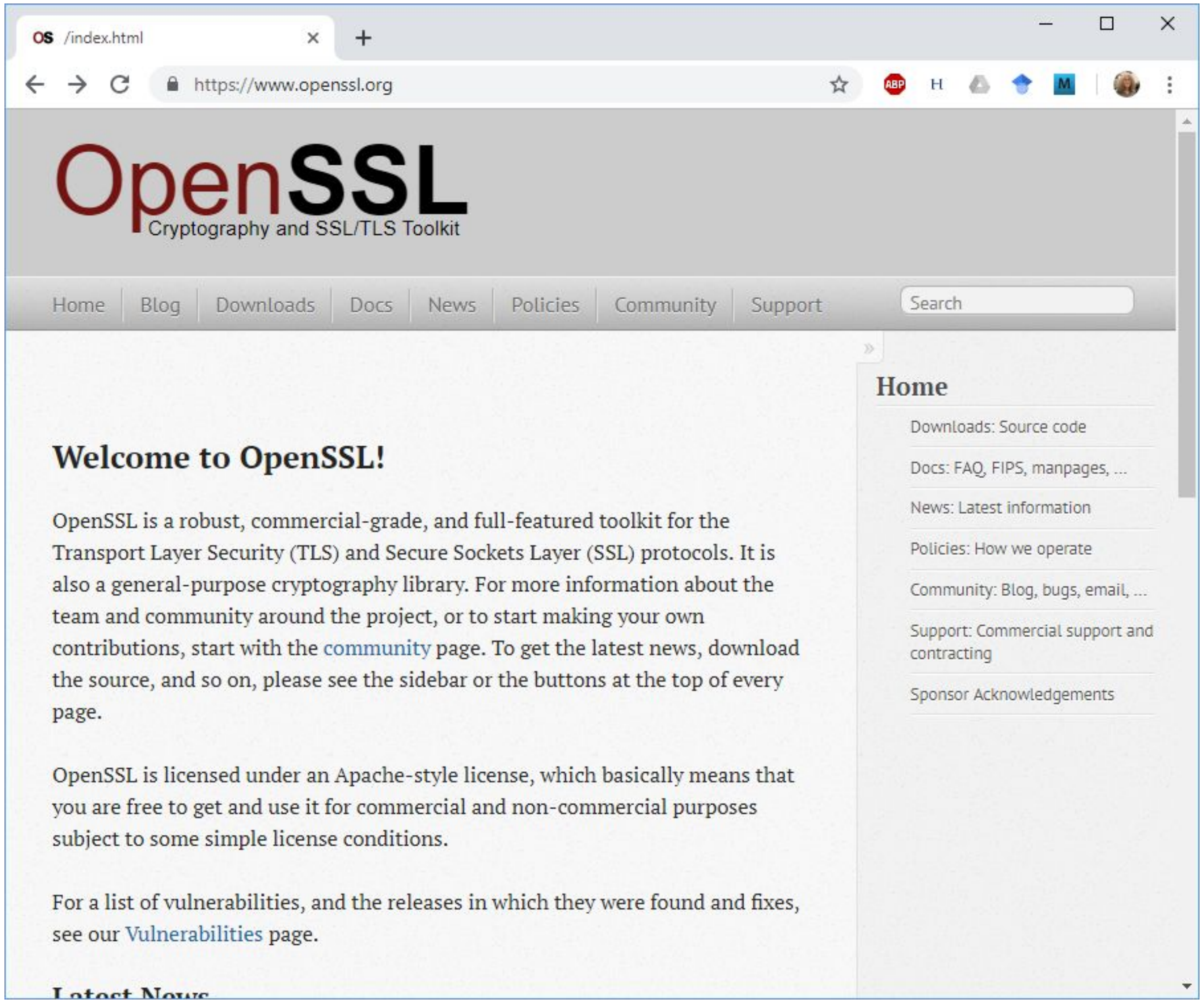
1. Course details.
2. Correction on the Host: parameter.
3. Review of reading an HTTP webpage.
4. Read an HTTPS webpage.
5. TLS, SSL and the OpenSSL library.
6. LinuxGetSSL.

OpenSSL

The SSL/TLS handshake is too complex and rigorous to write on our own.

OpenSSL is the one library you get use.

It works on both Windows and Linux.



Home

Downloads: Source code

Docs: FAQ, FIPS, manpages, ...

News: Latest information

Policies: How we operate

Community: Blog, bugs, email, ...

Support: Commercial support and contracting

Sponsor Acknowledgements

Welcome to OpenSSL!

OpenSSL is a robust, commercial-grade, and full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is also a general-purpose cryptography library. For more information about the team and community around the project, or to start making your own contributions, start with the community page. To get the latest news, download the source, and so on, please see the sidebar or the buttons at the top of every page.

OpenSSL is licensed under an Apache-style license, which basically means that you are free to get and use it for commercial and non-commercial purposes subject to some simple license conditions.

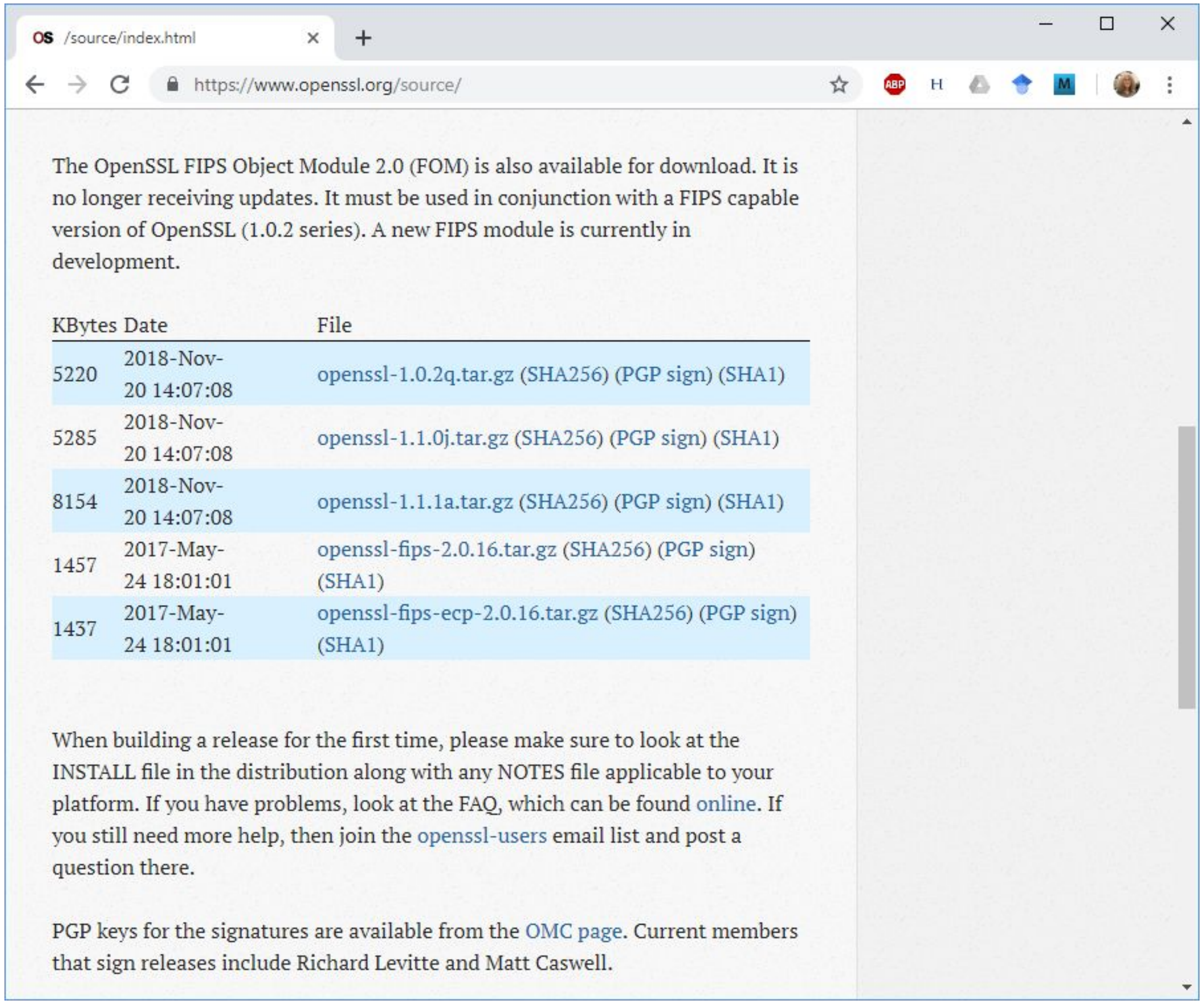
For a list of vulnerabilities, and the releases in which they were found and fixes, see our Vulnerabilities page.

OpenSSL

But the downloads are all source only.

You have to build and install it yourself.

Hint: It will not build in directory path with spaces, e.g., your Google drive path.



OS /source/index.html

https://www.openssl.org/source/

The OpenSSL FIPS Object Module 2.0 (FOM) is also available for download. It is no longer receiving updates. It must be used in conjunction with a FIPS capable version of OpenSSL (1.0.2 series). A new FIPS module is currently in development.

KBytes	Date	File
5220	2018-Nov-20 14:07:08	openssl-1.0.2q.tar.gz (SHA256) (PGP sign) (SHA1)
5285	2018-Nov-20 14:07:08	openssl-1.1.0j.tar.gz (SHA256) (PGP sign) (SHA1)
8154	2018-Nov-20 14:07:08	openssl-1.1.1a.tar.gz (SHA256) (PGP sign) (SHA1)
1457	2017-May-24 18:01:01	openssl-fips-2.0.16.tar.gz (SHA256) (PGP sign) (SHA1)
1437	2017-May-24 18:01:01	openssl-fips-ecp-2.0.16.tar.gz (SHA256) (PGP sign) (SHA1)

When building a release for the first time, please make sure to look at the INSTALL file in the distribution along with any NOTES file applicable to your platform. If you have problems, look at the FAQ, which can be found online. If you still need more help, then join the openssl-users email list and post a question there.

PGP keys for the signatures are available from the OMC page. Current members that sign releases include Richard Levitte and Matt Caswell.

OpenSSL

1. Install the OpenSSL library.

```
sudo apt-get install libssl-dev
```

2. Include the openssl header.

```
#include <openssl/ssl.h>
```

3. Compile and link the SSL and crypto libraries.

```
g++ LinuxGetSsl.cpp -lssl –lcrypto
    -o LinuxGetSsl
```

```c
#include <openssl/ssl.h>

int SSL_library_init(void);
```

SSL_library_init() initializes the SSL library.

```c
#include <openssl/ssl.h>

int SSL_library_init(void);
SSL_CTX *SSL_CTX_new(const SSL_METHOD *method);
SSL *SSL_new(SSL_CTX *ctx);
```

SSL_library_init() initializes the SSL library.

SSL_CTX_new() creates a new SSL_CTX context object as framework to establish TLS/SSL enabled connections. Various options regarding certificates, algorithms etc. can be set in this object.

SSL_new() creates a new SSL structure need to hold the data for a TLS/SSL connection

Here’s an example initializing the SSL layer.

```
// Build the SSL layer.

SSL_library_init( );

SSL_CTX *ctx = SSL_CTX_new( SSLv23_method( ) );
assert( ctx );
SSL *ssl = SSL_new( ctx );
assert( ssl );
```

```c
#include <openssl/ssl.h>

int SSL_set_fd(SSL *ssl, int fd);
```

SSL_set_fd() sets the file descriptor fd as the input/output facility for the TLS/SSL (encrypted) side of ssl. fd will typically be the socket file descriptor of a network connection.

```c
#include <openssl/ssl.h>

int SSL_connect(SSL *ssl);
```

SSL_connect() initiates the TLS/SSL handshake with a server.

Here’s an example initializing the SSL layer.

```
// Fill in the socket we'll be using.

SSL_set_fd( ssl, s );

// Establish an SSL connection.

int sslConnectResult = SSL_connect( ssl );
assert( sslConnectResult == 1 );
```

```c
#include <openssl/ssl.h>

int SSL_write(SSL *ssl, const void *buf, int num);
int SSL_read(SSL *ssl, void *buf, int num);
```

SSL_write() writes num bytes from the buffer buf into the specified ssl connection.
SSL_read() tries to read num bytes from the specified ssl into the buffer buf.

Here’s an example reading and writing.

```
while ( ( bytes = SSL_read( ssl, buffer,
      sizeof( buffer ) ) ) > 0 )
   write( 1, buffer, bytes );
```

Shutdown and free up resources at the end.

```
SSL_shutdown( ssl );
SSL_free( ssl );
SSL_CTX_free( ctx );
```

Agenda

1. Course details.
2. Correction on the Host: parameter.
3. Review of reading an HTTP webpage.
4. Read an HTTPS webpage.
5. TLS, SSL and the OpenSSL library.
6. LinuxGetSSL.

Here’s the entire main( ) for LinuxGetSsl, minus only all the code.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <openssl/ssl.h>
#include <netdb.h>

int main( int argc, char **argv )
   {
   // Parse the URL

   // Get the host address.

   // Create a TCP/IP socket.

   // Connect the socket to the host address.

   // Build an SSL layer and set it to read/write
   // to the socket we've connected.

   // Send a GET message.

   // Read from the SSL socket until there's no more data, copying it to
   // stdout.

   // Close the socket and free the address info structure.
   }
```