LAU
الجامعة اللبنانية الأميركية
Lebanese American University

CSC 634: Networks Programming

Lecture 06: Transport Layer Security

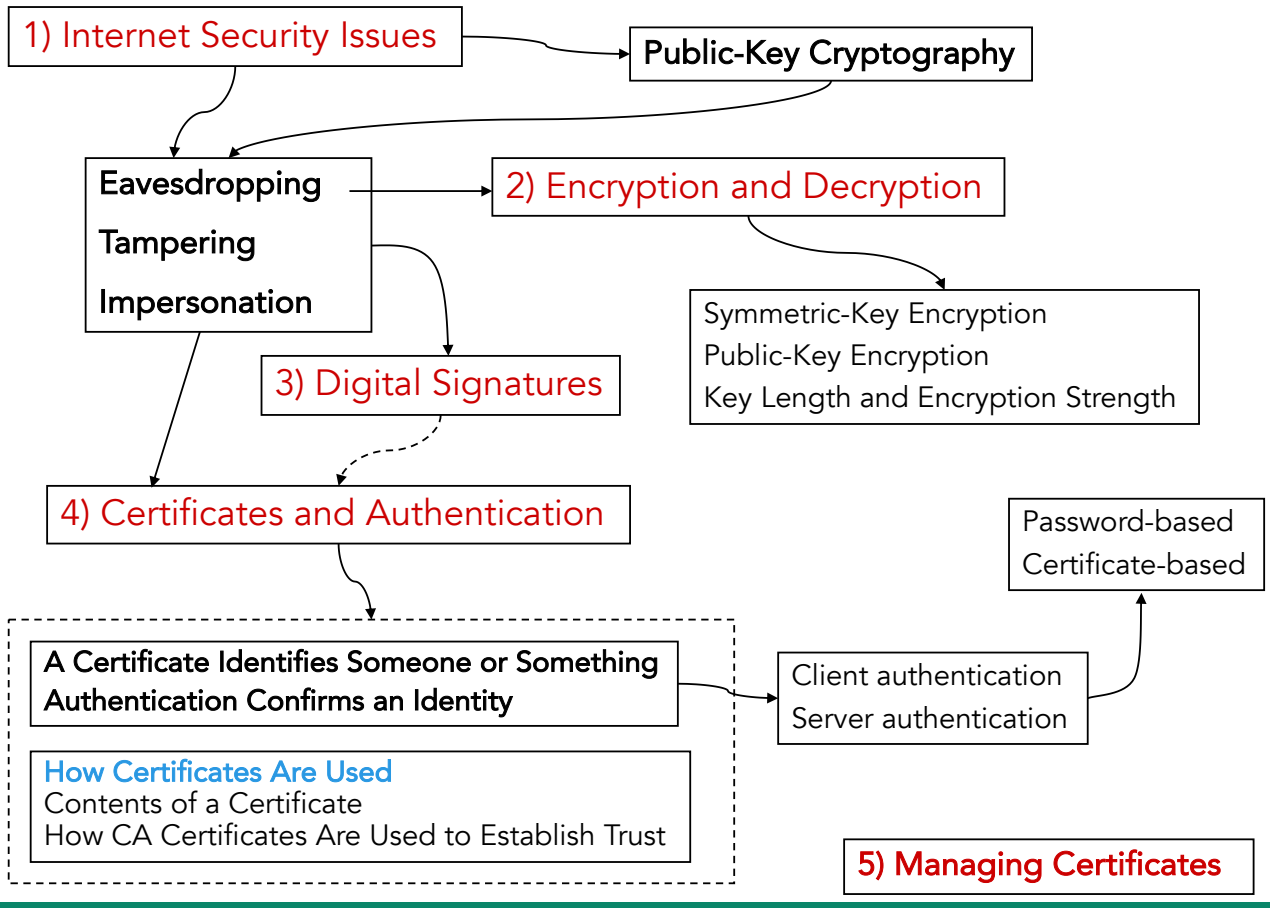
Instructor: Haidar M. Harmanani

Introduction

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

SSL and TLS

- TLS stands for Transport Layer Security (formerly known as SSL or Secure Sockets Layer) and is what enables encrypted communications between a web browser and a web server.



How Certificates Are Used

- Types of Certificates
- SSL Protocol
- Signed and Encrypted Email
- Form Signing
- Single Sign-On
- Object Signing

Types of Certificates

- Client SSL certificates
- Server SSL certificates
- S/MIME certificates
- Object-signing certificates
- CA certificates

Types of Certificates → Client SSL certificates

- Used to identify clients to servers via SSL (client authentication).
- Typically, the identity of the client is assumed to be the same as the identity of a human being, such as an employee in an enterprise.
- Examples:
  - A bank gives a customer a client SSL certificate that allows the bank's servers to identify that customer and authorize access to the customer's accounts.
  - A company might give a new employee a client SSL certificate that allows the company's servers to identify that employee and authorize access to the company's servers.

Types of Certificates → Server SSL certificates

- Used to identify servers to clients via SSL (server authentication).
- Server authentication may be used with or without client authentication.
- Example:
  - Internet sites that engage in electronic commerce (commonly known as e-commerce) usually support certificate-based server authentication, at a minimum, to establish an encrypted SSL session and to assure customers that they are dealing with a web site identified with a particular company. The encrypted SSL session ensures that personal information sent over the network, such as credit card numbers, cannot easily be intercepted.

Types of Certificates → S/MIME certificates

- Used for signed and encrypted email.
- A single certificate may be used as both an S/MIME certificate and an SSL certificate.
- Examples:
  - A company deploys combined S/MIME and SSL certificates solely for the purpose of authenticating employee identities, thus permitting signed email and client SSL authentication but not encrypted email.
  - Another company issues S/MIME certificates solely for the purpose of both signing and encrypting email that deals with sensitive financial or legal matters.

Types of Certificates → Object-signing certificates

- Used to identify signers of Java code, JavaScript scripts, or other signed files.
- Example:
  - A software company signs software distributed over the Internet to provide users with some assurance that the software is a legitimate product of that company. Using certificates and digital signatures in this manner can also make it possible for users to identify and control the kind of access downloaded software has to their computers.

Types of Certificates → CA certificates

- Used to identify CAs.
- Client and server software use CA certificates to determine what other certificates can be trusted.
- Example:
  - The CA certificates stored in Communicator determine what other certificates that copy of Communicator can authenticate. An administrator can implement some aspects of corporate security policies by controlling the CA certificates stored in each user's copy of Communicator.

SSL Protocol

LAU
Lebanese American University

Authentication protocols

- Basic Authentication
  - Supported by almost all web browsers, web servers and proxy servers
  - Used to protect access to resources

Proxy Servers

- Perform web retrievals on behalf of a web browser
- Most often used to speed up Internet access and reduce bandwidth by caching frequently used pages
- Libraries use proxy servers to make off-campus web clients look like on-campus ones
- Authenticated users allowed to relay requests through our IP address space

Basic Authentication: Basic Problem

- Sends usernames and passwords unencrypted
- Sends them with every page request

- Solution: Use SSL

Basic Authentication and SSL

- Secure Socket Layer (SSL)
- SSL works in combination with Basic Authentication to encrypt pages
- URLs show https:// rather than http://
- Netscape: padlock in the bottom left hand corner
- Internet Explorer, a yellow lock at the middle of the status bar

Basic Authentication and SSL

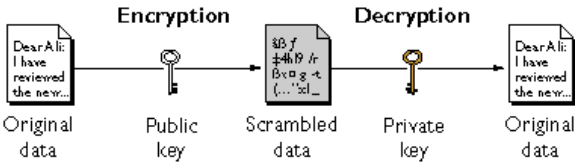
- Supported by:
  - Netscape Navigator (versions 2.0 and above),
  - Internet Explorer (version 3.0 and above)
  - AOL (AOL 3.0 and above)

The Secure Sockets Layer (SSL) Protocol

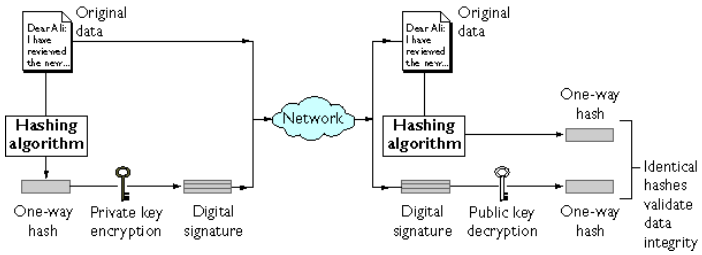
- Set of rules governing:
  - server authentication
  - client authentication
  - encrypted communication between servers and clients
- Widely used on the Internet, especially for interactions that involve exchanging confidential information such as credit card numbers.

SSL Protocol

- Requires a server SSL certificate, at a minimum
- Initial "handshake" process:
  - The server presents its certificate to the client to authenticate the server's identity;
- Authentication Process
  - Public-key encryption

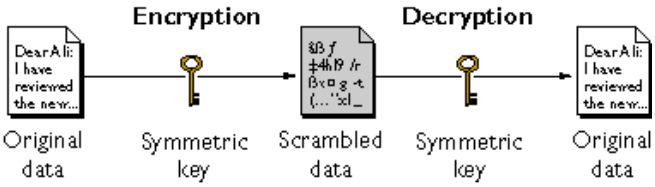


  - Digital signature



...once the server has been authenticated:

- The client and server use techniques of
- Symmetric-key encryption



  - Very fast
  - Encrypt all the information they exchange for the remainder of the session
  - Detect any tampering that may have occurred.

…optional

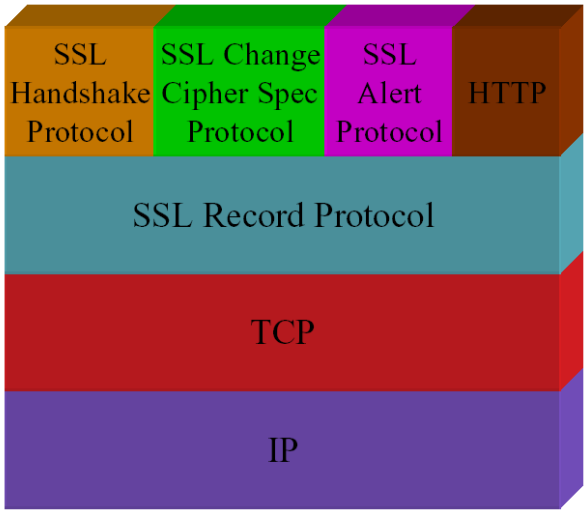
- Servers may be configured to require client authentication as well as server authentication.
- Server authentication is successfully completed
- Client must present its certificate to the server to authenticate the client's identity *before* the encrypted SSL session can be established.

More on SSL…

LAU
الجامعة اللبنانية الأمريكية
Lebanese American University

Secure Sockets Layer (SSL)

- Make use of TCP to provide reliable, secure end-to-end network service
- Two layers of protocol:
  - SSL Record Protocol: Transport
  - Higher-layer protocols for connection negotiation and alerts.
- SSL Connection: a secure stream within a session
- SSL Session: Association between client and server



SSL Shared Keys

- When a client and server establish an SSL connection for the first time they need to establish a shared key called the `master_secret`
- The `master_secret` is then used to create all the bulk encryption keys used to protect the traffic.

Eric Rescorla, *SSL and TLS: Designing and Building Secure Systems*, Addison-Wesley, 2001

SSL Connection and Session

- A connection represents one specific communications channel (typically mapped to a TCP connection), along with its keys, cipher choices, sequence number state, etc.
- A session is a virtual construct representing the negotiated algorithms and the master_secret .

Eric Rescorla, *SSL and TLS: Designing and Building Secure Systems*, Addison-Wesley, 2001

SSL Connection and Session

- A new session is created every time a given client and server go through a full key exchange and establish a new master_secret.
- Multiple connections can be associated with a given session.
- Although all connections in a given session share the same master_secret, each has its own encryption keys. This is absolutely necessary for security reasons because reuse of bulk keying material can be extremely dangerous.

Eric Rescorla, *SSL and TLS: Designing and Building Secure Systems*, Addison-Wesley, 2001

LAU
الجامعة اللبنانية الأمريكية
Lebanese American University

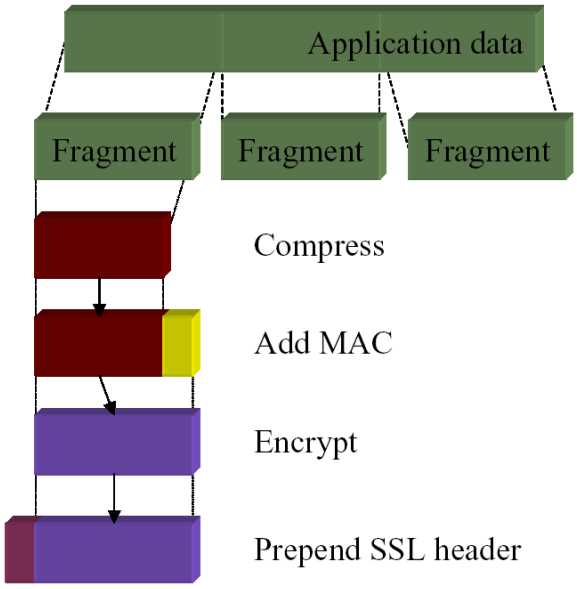
SSL Session State

Session identifier	Arbitrary byte sequence chosen by server to identify the session state.
Peer certificate	X509.v3 certificate of the peer (may be null).
Compression method	Algorithm used to compress data prior to encryption.
Cipher spec	Specifies the bulk data encryption algorithm (null, DES, etc.) and hash algorithm used for MAC calculation. Also includes other cryptographic attributes such as hash_size.
Master secret	48 byte secret shared between client and server.
Is resumable?	Flag indicating whether the session can be used to initiate new connections.

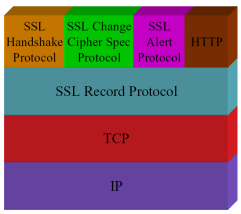
SSL Connection State

Server & client random	Byte sequences chosen by server & client for each connection.
Server write MAC secret	Key used in MAC operations over data sent by the server.
Client write MAC secret	Key used in MAC operations over data sent by the client.
Server write key	Encryption key for data sent by server.
Client write key	Key for data sent by client.
Initialization vectors	Used to initialize encryption for data sent in CBC mode.
Sequence numbers	Maintained by each party for transmitted and received information. May not exceed $2^{64}-1$.

SSL Record Protocol

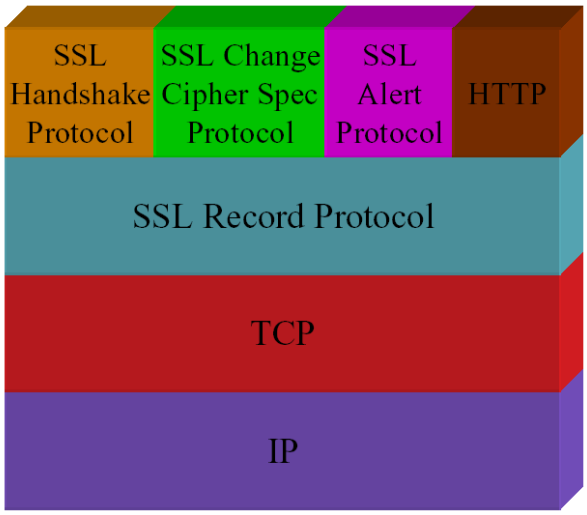


- Data fragmented into blocks of 2^{14} bytes or less.
- Compression applied (optionally).
- MAC calculated.
- Payload & MAC encrypted.
- Header prepended:
  - content type,
  - major & minor version,
  - compressed length.



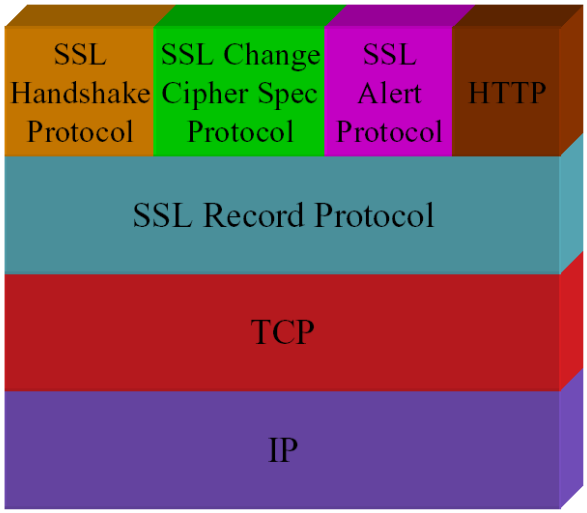
Change Cipher Spec Protocol

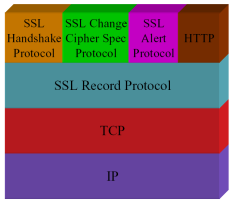
- A single message that contains a single byte
- A signal that the pending state should be copied into the current state
  - Updates the cipher information used by this connection
  - The state must have been set by the Handshake Protocol



SSL Alert Protocol

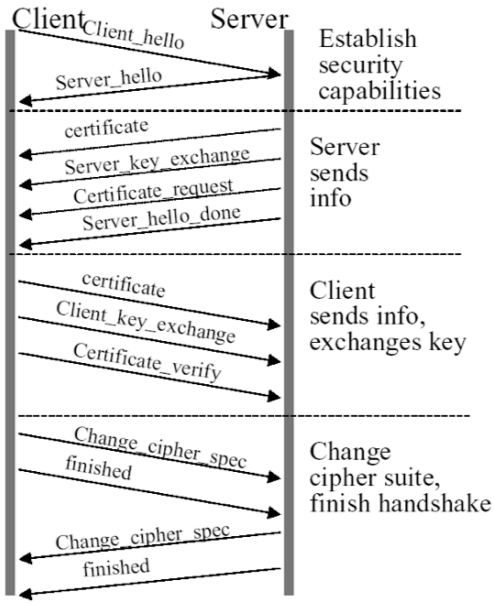
- Conveys SSL-related alerts (compressed and encrypted)
- Each message consists of exactly two bytes
  - Level: Severity of the alert (warning or fatal)
  - Alert Code: What kind of alert is this:
    - Unexpected message
    - Bad record MAC
    - Decompression failure
    - Handshake Failure





SSL Handshake Protocol

- First phase: Establish security Capabilities
  - Exchange information
  - Find Common ground for message exchange
- Second phase: Authenticate server and exchange key
- Third Phase: Authenticate client and exchange key
- Fourth phase: Finish



SSL: Does It Work?

- SSL has undergone lots of informal examination
- There have also been systematic but informal analysis of SSL and some partial proofs
- Now and then, some problematic issue is pointed out
- It is at the edge of what we can understand

SSL: Other Issues

- SSL slows down servers
- SSL ”breaks” caching and complicates virtual hosting
- SSL protects data in transit but not databases
- SSL is a two-way protocol (unlike SET)

An Introduction to OpenSSL Programming

Secure Sockets Layer (SSL)

Assignment #4
Due Date: 16. 4. 2002

This assignment demonstrates the Client/Server communication using SSL It is similar to the provious assignment.You are required to rewrite Assignment 3 using SSL. To simplify things make the server non-concurrent. To accomplish the requirement, you have to do the following steps:

1. Make yourself the Certificate Authority (CA) [1%]
2. Generate server certificate/key pair to be used by SSL [1%]
3. Write the SSL code, run and test [5.5%]

Submit the following:

1. Certificate Authority (CA) file.(CA.pem)
2. Server Certificate (server.pem)
3. Source code for client and server

"How to create CA cerificate" and SSL examples are in: ~ijirasek/examples/ssl

Also, see the Web pages listed in the course Web page

An Introduction to OpenSSL Programming

- SSL → The quickest and easiest way to secure a TCP-based network application
- OpenSSL
  - Best choice if you're working in C
  - API is vast and complicated
- Unfortunately, poor distributed documentation + sample code
- Manual pages are pretty good as a reference, not as a tutorial
- Article I → a simple Web client and server pair that demonstrates the basic features of OpenSSL.
- Article II → advanced features, such as session resumption and client authentication.

- Complete source code → http://www.rtfm.com/openssl-examples/

OpenSSL

- Free library providing cryptographic functions
  - it's not the only one, alternatives: Crypto++ and Cryptlib of Peter Guttman
- The important feature is the complete implementation of the protocols SSLv2,SSLv3 and TLSv1

- The relevant specifications are at:
  - SSLv2: http://www.netscape.com/eng/security/SSL_2.html
  - SSLv3: http://home.netscape.com/eng/ssl3/index.html
  - TLS (RFC 2246): http://www.ietf.org/rfc/rfc2246.txt
  - HTTPS (RFC 2818): http://www.ietf.org/rfc/rfc2818.txt

ssldump → SSLv3/TLS network protocol analyzer

- It identifies TCP connections on the chosen network interface and attempts to interpret them as SSLv3/TLS traffic.
- When it identifies SSLv3/TLS traffic, it decodes the records and displays them in a textual form to stdout.
- If provided with the appropriate keying material, it will also decrypt the connections and display the application data traffic.

ssldump → SSLv3/TLS network protocol analyzer

```
New TCP connection #3: localhost(3638) <-> localhost(4433)
3 1  0.0738 (0.0738)  C>S  Handshake      ClientHello
3 2  0.0743 (0.0004)  S>C  Handshake      ServerHello
3 3  0.0743 (0.0000)  S>C  Handshake      Certificate
3 4  0.0743 (0.0000)  S>C  Handshake      ServerHelloDone
3 5  0.0866 (0.0123)  C>S  Handshake      ClientKeyExchange
3 6  0.0866 (0.0000)  C>S  ChangeCipherSpec
3 7  0.0866 (0.0000)  C>S  Handshake      Finished
3 8  0.0909 (0.0043)  S>C  ChangeCipherSpec
3 9  0.0909 (0.0000)  S>C  Handshake      Finished
3 10 1.8652 (1.7742)  C>S  application_data
3 11 2.7539 (0.8887)  C>S  application_data
3 12 5.1861 (2.4321)  C>S  Alert          warning          close_notify
3    5.1868 (0.0007)  C>S  TCP FIN
3    5.1893 (0.0024)  S>C  TCP FIN
```

An Introduction to OpenSSL Programming: Article I

- Sample Programs (5 main points)
  - Context Initialization
  - Initialize the library and create the context
  - Load our own keys
  - Load root CA list
  - Load Randomness

An Introduction to OpenSSL Programming

Article I

The Client (11 main points)

- **Handshake**
- **Server Authentication**
- **Server Identity**
- **Chain Length**
- **When To Check Certificates**
- **Request**
- **Response**
- **Error Handling**
- **Closure**
- **Shutdown**
- **Cleanup**

An Introduction to OpenSSL Programming

Article I

The Server (6 main points)

- **Accept and Fork**
- **Server Accept**
- **Buffered I/O**
- **Request**
- **Response**
- **Shutdown**

- **What's Missing**
  - Better Certificate Checking
  - Better Error Handling

OpenSSL Programming:

Programs

A simple HTTPS (see RFC 2818) client:

- It initiates an SSL connection to the server and then transmits an HTTP request over that connection. It then waits for the response from the server and prints it to the screen.

A simple HTTPS server:

1. It waits for TCP connections from clients.
2. When it accepts one it negotiates an SSL connection.
3. Once the connection is negotiated, it reads the client's HTTP request.
4. It then transmits the HTTP response to the client.
5. Once the response is transmitted it closes the connection.

OpenSSL Programming:
Programs → Context Initialization

- Set up a context object (an SSL_CTX) → creates a new connection object for each new SSL connection.
- Connection objects → do SSL handshakes, reads, and writes.
- Advantages:
  - ✓ Create a new connection? simply point that connection to the context object.
  - ✓ Allows multiple SSL connections to share data.

OpenSSL Programming:
Programs → Initialize the library and create the context

SSL_library_init() → loads up the algorithms

SSL_load_error_strings() → If we want good reporting of errors

BIO object → allows the programmer to use the same functions for different kinds of I/O channels (sockets, terminal, memory buffers, etc.)

In this example: a BIO object attached to stderr to be used for printing errors.

SSL_library_init() → loads up the algorithms

Algorithms implemented

- Block ciphers: DES, 3DES, DESX, CAST, RC2, RC5, IDEA, Blowfish
- stream cipher: RC4
- hash: MD2, MD4, MD5, SHA-1, RIPEMD 160, MDC2
- asymmetric cryptosystems: RSA, DSA, DH
- MAC: HMAC

OpenSSL Programming:
Programs → Load our own keys

Load your own public/private key pair and the associated certificate.

SSL_CTX_use_certificate_chain_file()

SSL_CTX_use_PrivateKey_file() → load the private key.

The private key is usually encrypted under a password;

SSL_CTX_set_default_passwd_cb() →obtain the password

OpenSSL Programming:
Programs → Load root CA list

Host authentication

OpenSSL needs to know what certificate authorities (CAs) you trust.

SSL_CTX_load_verify_locations() → loads the CAs.

OpenSSL Programming:
Programs → Load Randomness

Good security →SSL needs a good source of strong random numbers.

OpenSSL automatically uses /dev/urandom

A system other than Linux → may get an error at some point because the random

number generator is unseeded.

OpenSSL's rand(3) manual page provides more information.

An Introduction to OpenSSL Programming

Article I

The Client

- **Handshake**
- **Server Authentication**
- **Server Identity**
- **Chain Length**
- **When To Check Certificates**
- **Request**
- **Response**
- **Error Handling**
- **Closure**
- **Shutdown**
- **Cleanup**

OpenSSL Programming:

Client

SSL context initialized by client → Ready to connect to the server.

OpenSSL requires us to create a TCP connection between client and server on our own and then use the TCP socket to create an SSL socket.

Once the TCP connection has been created, we create an SSL object to handle the connection.This object needs to be attached to the socket.

We don't directly attach the SSL object to the socket → we create a BIO object using the socket and then attach the SSL object to the BIO.

This abstraction layer allows you to use OpenSSL over channels other than sockets, provided you have an appropriate BIO.

OpenSSL Programming:
Client → Handshake

- The first step in an SSL connection.
- It authenticates the server (and optionally the client)
- Establishes the keying material that will be used to protect the rest of the traffic.
- SSL_connect()
- It will not return until the handshake is completed or an error has been detected.

OpenSSL Programming:
Client → Server Authentication

- OpenSSL does some of the checks
- Some of the checks are application specific
- Sample application → check the server identity with check_cert() function

OpenSSL Programming:
Client → Server Identity

- Verifies that the certificate you're looking at matches the identity that we expect the server to have.
- In most cases, this means that the server's DNS name appears in the certificate

Sample programs:

1. SSL_get_peer_certificate() → extract the server's certificate
2. Compare the common name to the host name we're connecting to.
3. If they don't match, something is wrong and we exit.

OpenSSL Programming:
Client → Chain Length

- Certificate I'm looking at → signed by the CA?
- X.509 version 3 contains a way for a CA to label certain certificates as other CAs.
- This permits a CA to have a single root that then certifies a bunch of subsidiary CAs.
- SSL_CTX_set_verify_depth() → check the chain length
- Modern versions of OpenSSL (0.9.5 and later) automatically check these extensions
- Highly advisable to upgrade to 0.9.6

OpenSSL Programming:
Client → When To Check Certificates

- Only after the handshake completes do we check the certificate and possibly terminate the connection.
- Advantage → it's easy to implement
- Disadvantage → if we don't like the server's certificate the server just sees a generic error rather than one indicating that the certificate was bad.

- Alternative → SSL_CTX_set_verify() : requires valid certificates during the handshake—if the server presents an invalid certificate the handshake will fail and the server will receive an error indicating that it had a bad certificate.

- You can also use SSL_CTX_set_verify() to set a callback to be called during certificate checking →extra checks or permit certificates which OpenSSL might otherwise not accept.

OpenSSL Programming:
Client → HTTP Request

Because the machine we're connecting to may change we do need to fill in the Host header.

SSL object instead of the file descriptor.

SSL_write() semantics are 'all-or-nothing': the call won't return until all the data is written or an error occurs, whereas write() may only write part of the data.

```
26      request_len=strlen(REQUEST_TEMPLATE)+
27        strlen(host)+6;
28      if(!(request=(char *)malloc(request_len)))
29        err_exit("Couldn't allocate request");
30      sprintf(request,REQUEST_TEMPLATE,
31        host,port);
32
33      /* Find the exact request_len */
34      request_len=strlen(request);
35
36      r=SSL_write(ssl,request,request_len);
37      switch(SSL_get_error(ssl,r)){
38        case SSL_ERROR_NONE:
39          if(request_len!=r)
40            err_exit("Incomplete write!");
41          break;
42          default:
43            berr_exit("SSL write problem");
44      }
```

OpenSSL Programming:
Client → Response

- Old-style HTTP/1.0 →the server transmits its response and then closes the connection.
- Later versions → persistent connections (multiple sequential transactions on the same connection)
- Sample programs do not use persistent connections (omit the header that allows them)
- •Operationally, this means that we can keep reading until we get an end of file,

```
48      while(1){
49        r=SSL_read(ssl,buf,BUFSIZZ);
50        switch(SSL_get_error(ssl,r)){
51          case SSL_ERROR_NONE:
52            len=r;
53            break;
54          case SSL_ERROR_ZERO_RETURN:
55            goto shutdown;
56          case SSL_ERROR_SYSCALL:
57            fprintf(stderr,
58              "SSL Error: Premature close0);
59            goto done;
60          default:
61            berr_exit("SSL read problem");
62        }
63
64        fwrite(buf,1,len,stdout);
65      }
```


OpenSSL Programming:
Client → Error Handling

- SSL_get_error()
- Two kinds of errors we're concerned with: ordinary errors and "premature closes".
- Error handling in our client is pretty primitive so with most errors we simply call berr_exit() to print an error message and exit.

A real application would, of course, be able to recognize errors and signal them to the user or some audit log rather than just exiting.

OpenSSL Programming:
Client → Closure

- TCP uses a FIN segment → sender has sent all of its data.
- SSL version 2 simply allowed either side to send a TCP FIN to terminate the SSL connection.
- This allowed for a "truncation attack":
  - ➢ The attacker could make it appear that a message was shorter than it was simply by forging a TCP FIN.
  - ➢ Unless the victim had some other way of knowing what message length to expect it would simply believe that it had received a shorter message.
- SSLv3 introduced a close_notify alert → an SSL message (and therefore secured) but not part of the data stream itself (not seen by the application).
- No data may be transmitted after the close_notify is sent.

OpenSSL Programming:
Client → Closure

- When SSL_read() returns 0 to indicate that the socket has been closed, this really means that the close_notify has been received.
- If the client receives a FIN before receiving a close_notify, SSL_read() will return with an error. This is called a "premature close".
- Unless you want to be reporting errors all the time you often have to ignore premature closes.
- Sample programs → report the premature close on stderr but doesn't exit with an error.

OpenSSL Programming:
Client → Shutdown

- If we read the response without any errors then we need to send our own close_notify to the server.

- SSL_shutdown()

- Since we've already received the server's close_notify, about the only thing that can go wrong is that we have trouble sending our close_notify.

OpenSSL Programming:
Client → Cleanup

Finally, we need to destroy the various objects we've allocated.

An Introduction to OpenSSL Programming

Article I

The Server

- **Accept and Fork**
- **Server Accept**
- **Buffered I/O**
- **Request**
- **Response**
- **Shutdown**

- **What's Missing**
  - Better Certificate Checking
  - Better Error Handling

Our web server is mainly a mirror of the client, but with a few twists:

1. We fork() in order to let the server handle multiple clients.
2. We use OpenSSL's BIO APIs to read the client's request one line at a time, as well as to do buffered writes to the client.
3. The server closure sequence is more complicated.

OpenSSL Programming:
Server → Accept and Fork

- Multiple clients?
  Create a new server process for each client that connects
  → fork() after accept() returns.

- Each new process executes independently and just exits when it's finished serving the client.

- This approach can be quite slow on busy web servers

OpenSSL Programming:
Server → Server Accept

- After forking and creating the SSL object, server calls SSL_accept() → causes OpenSSL to perform the server side of the SSL handshake.

- The only situation in which SSL_accept() will return is when the handshake has completed or an error has been detected.

OpenSSL Programming:
Server → Buffered I/O

BIO object → allows the programmer to use the same functions for different kinds of I/O channels (sockets, terminal, memory buffers, etc.)

BIO objects are stackable → we can wrap an SSL object in a BIO (the ssl_bio object) and then wrap that BIO in a buffered BIO object

```
13      io=BIO_new(BIO_f_buffer());
14      ssl_bio=BIO_new(BIO_f_ssl());
15      BIO_set_ssl(ssl_bio,ssl,BIO_CLOSE);
16      BIO_push(io,ssl_bio);
```

This allows us to perform buffered reads and writes on the SSL connection by using the BIO_* functions on the new io object.

OpenSSL Programming:
Server → Request

- An HTTP request → a request line + a bunch of header lines + an optional body.
- The end of the header lines is indicated by a blank line
- The most convenient way to read the request line and headers is to read one line at a time until you see a blank line. BIO_gets() → It takes an arbitrary size buffer and a length and reads a line from the SSL connection into that buffer.
- Note that we don't really DO anything with the HTTP request. We just read it and discard it.
- A real implementation would read the request line and the headers, figure out of there was a body and read that too.
- However, none of these things show anything interesting about SSL so they don't add anything to this demonstration.

OpenSSL Programming:
Server → Response

- The next step is to write the HTTP response and close the connection.

- BIO_puts() instead of SSL_write():
  - ➢ This allows us to write the response one line at a time but have the entire response written as a single SSL record.
  - ➢ Important because the cost of preparing an SSL record for transmission (computing the integrity check and encrypting it) is quite significant.

OpenSSL Programming:
Server → Response

Important aspects about using this kind of buffered write:

1. You need to flush the buffer before you close. The SSL object has no knowledge that you've layered a BIO on top of it, so if you destroy the SSL connection you'll just leave the last chunk of data sitting in the buffer. The BIO_flush() call takes care of this.

2. By default, OpenSSL uses a 1024-byte buffer size for buffered BIOs. Because SSL records can be up to 16K bytes long, using a 1024-byte buffer cause excessive fragmentation (and hence lower performance.) You can use the BIO_ctrl() API to increase the buffer size.

OpenSSL Programming:
Server → Shutdown

- Once we've finished transmitting the response we need to send our close_notify → SSL_shutdown().
- Things get a bit trickier when the server closes first.
- SSL_shutdown() returns immediately but with a value of 0, indicating that the closure sequence isn't finished. It's then the application's responsibility to call SSL_shutdown() again.

It's possible to have two attitudes here:

1. We've seen all of the HTTP request that we care about. we're not interested in anything else. Hence, we don't care whether the client sends a close_notify or not.
2. We strictly obey the protocol and expect others to as well. Thus, we require a close_notify.

OpenSSL Programming:
Server → Shutdown

```
51      r=SSL_shutdown(ssl);
52      if(!r){
53        /* If we called SSL_shutdown() first then
54           we always get return value of '0'. In
55           this case, try again, but first send a
56           TCP FIN to trigger the other side's
57           close_notify*/
58        shutdown(s,1);
59        r=SSL_shutdown(ssl);
60      }
61
62      switch(r){
63        case 1:
64          break; /* Success */
65        case 0:
66        case -1:
67        default:
68          berr_exit("Shutdown failed");
69      }
```

Session Resumption

SSL handshakes are expensive but SSL provides a feature called session resumption that allows you to bypass the full handshake for a peer you've communicated with before.

An Introduction to OpenSSL Programming

Article II

Multiplexed and Non-blocking I/O

We used blocking sockets for all of our reads and writes. This means that while the application is waiting for data from the peer it's completely stalled. This wouldn't be acceptable in a real GUI application like a browser. Instead, you need to use the select() call to multiplex between SSL and other sources of I/O.

An Introduction to OpenSSL Programming

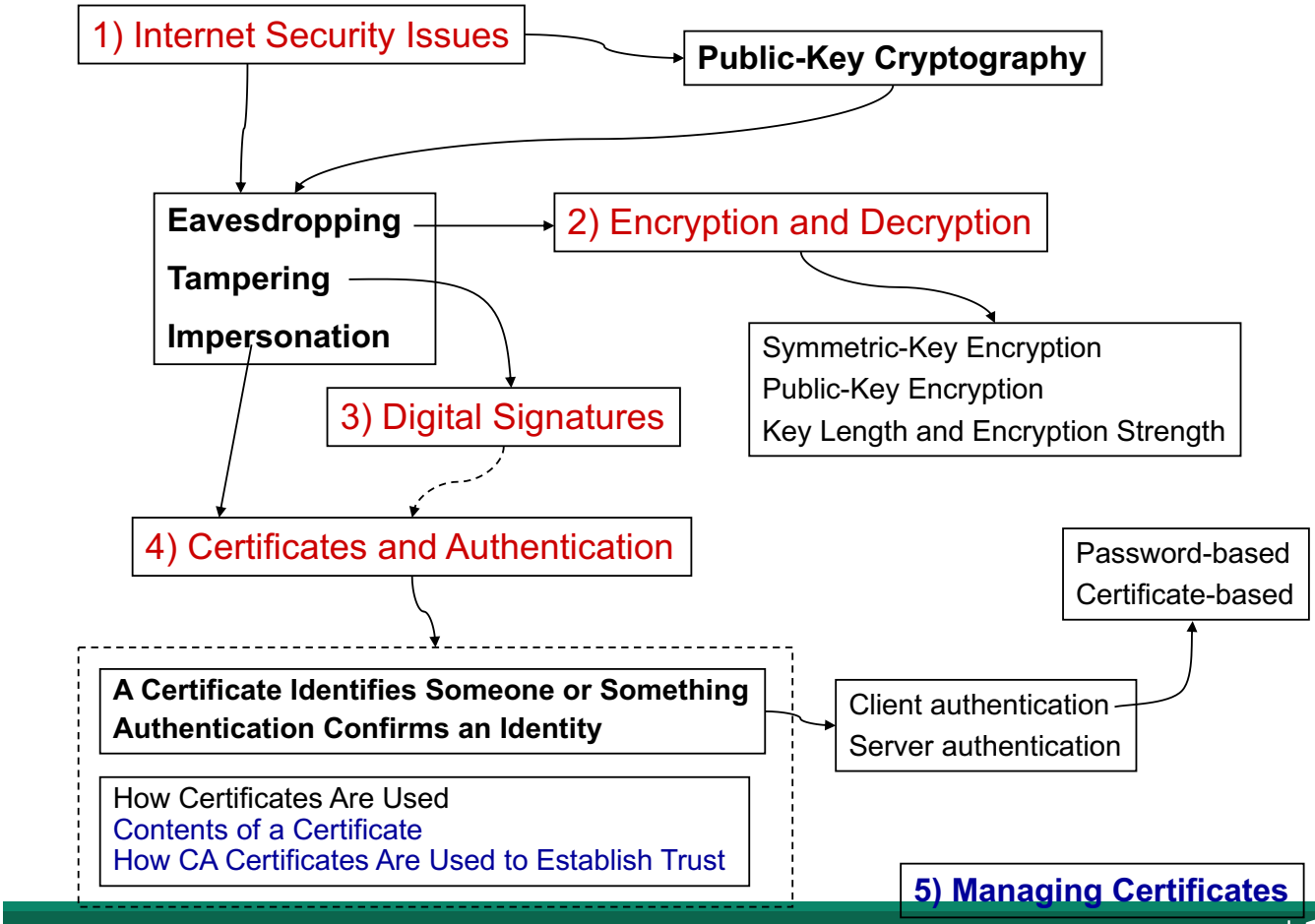
Article II

Client Authentication

Server authentication occurs as a part of almost every SSL connection. However, SSL also offers certificate-based client authentication, at the server's request.

How Certificates Are Used

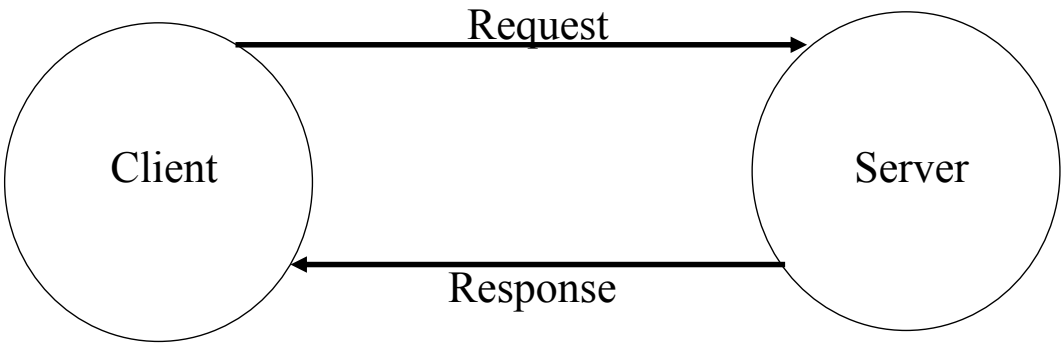
- Types of Certificates
- SSL Protocol
- Signed and Encrypted Email
- Form Signing
- Single Sign-On
- Object Signing



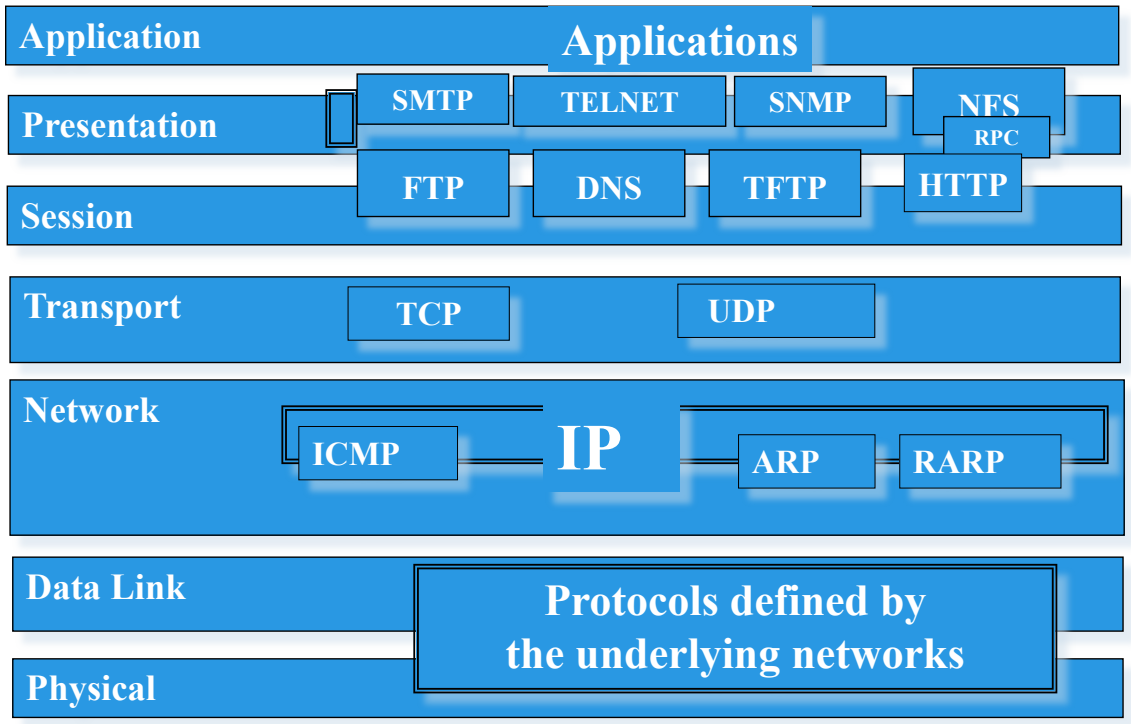
Web Concepts for E-Commerce

- Client/Server Applications
- Communication Channels
- TCP/IP

Client/Server Applications

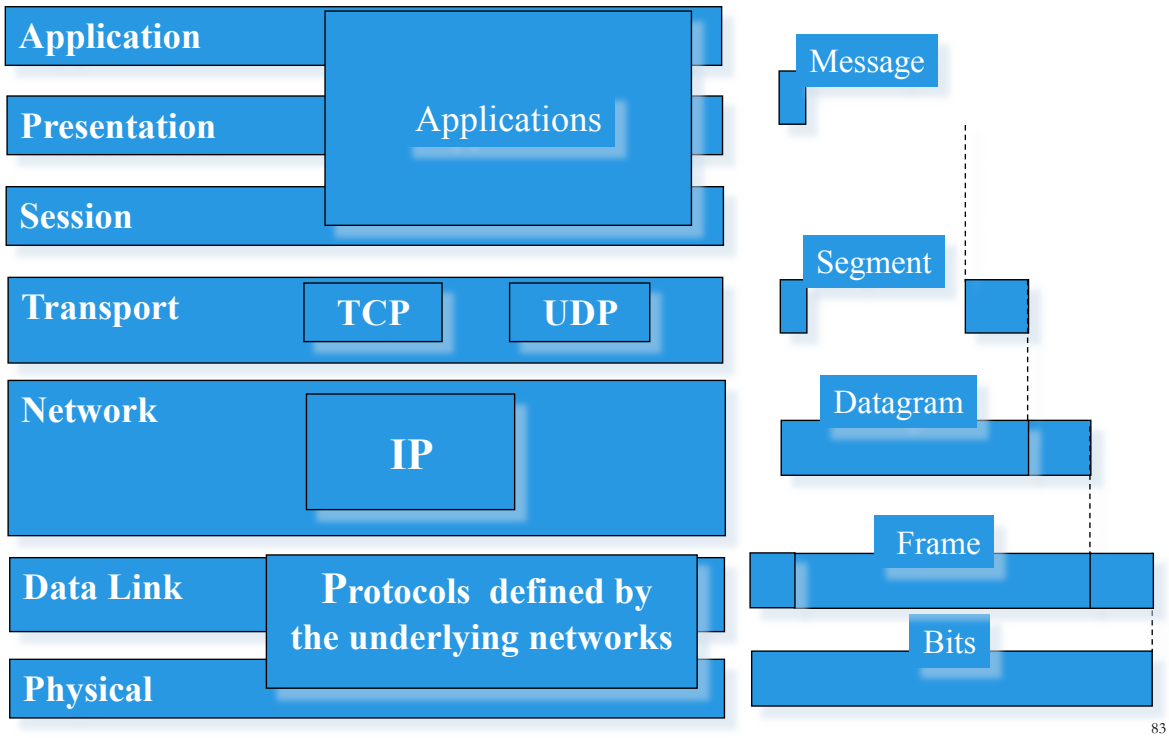


Review: TCP/IP and OSI Model

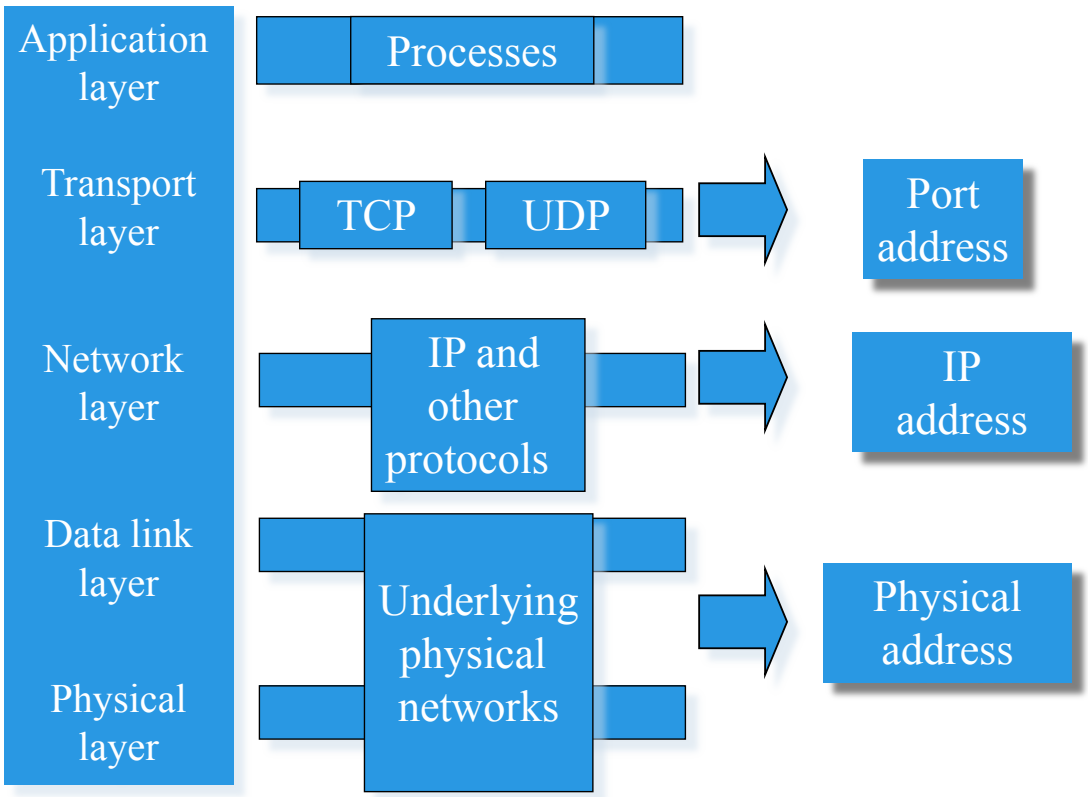


Review: TCP/IP and OSI Model

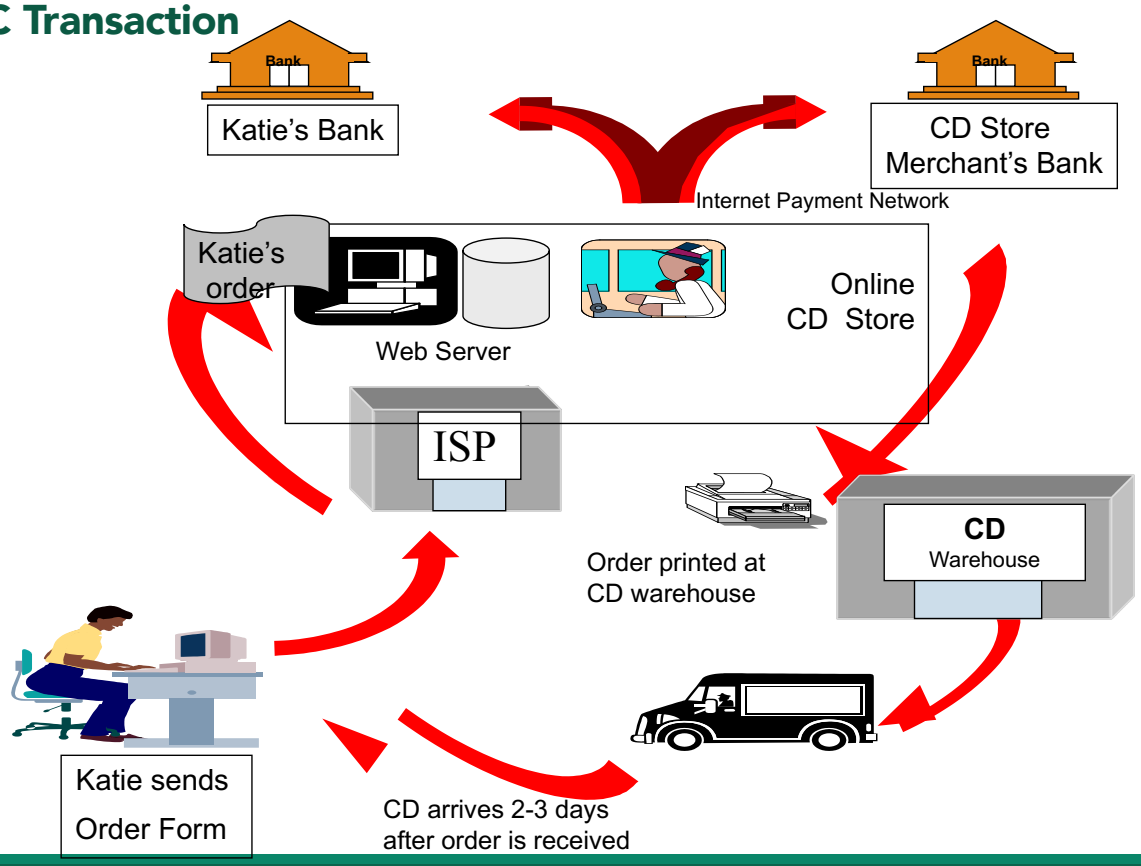
cont'd



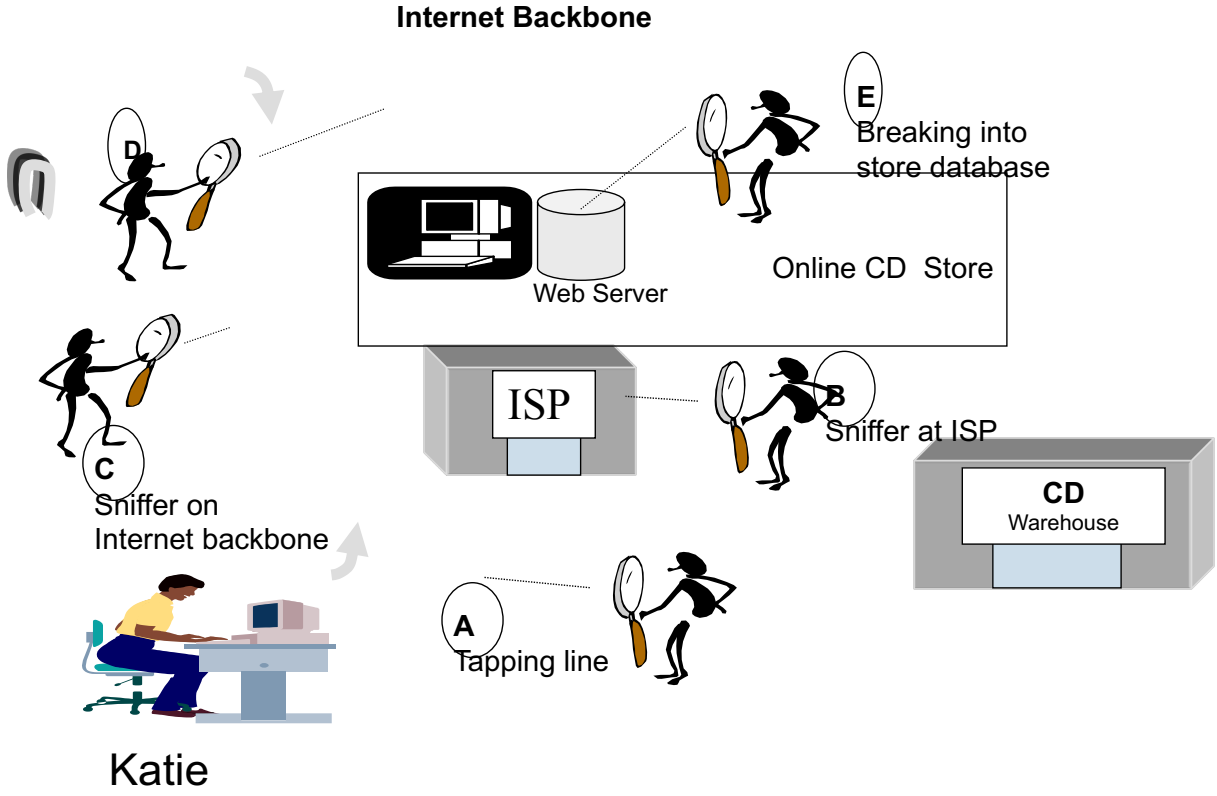
Review: TCP/IP and Addressing



Typical B2C Transaction



Web Security Threats in B2C



Why is Network Security Difficult?

- Open and interoperable protocols, while desirable, tend to work against security.
- Security is often sacrificed in return for gains in performance and scalability.
- Providing good security is expensive, so it can be difficult to get resources to support it.
- People tend to see security as a barrier to getting useful work done, and resist it.
- Information on circumventing security is widely available, as are software tools.
- Some people see circumventing security as a challenge and enjoy doing it.
- Most systems and networks were not designed with any security concerns in mind.

Security Planning, Policies, and Mechanisms

- A secure network does not come naturally.
  - First, a security plan must be developed.
  - Based on this plan, security policies must be developed.
  - To enforce these policies, the appropriate security mechanisms must be put in place.
- This is not a one time effort either.
  - Security plans must be reviewed and revised.
  - Security policies must be assessed and updated.
  - Security mechanisms must be patched, updated, or replaced as newer technologies become available.

Security Planning, Policies, and Mechanisms

- Security planning must do the following:
- Determine what your security needs are.
- Determine what your security threats and risks are.
- Identify who can be trusted to do what.

- Security policies are developed from this information, so be as specific as possible.

Security Planning, Policies, and Mechanisms

- Security policies that are developed must ensure appropriate levels of security for the activities performed in the network by:
- Making it clear what is protected and why.
- Clearly stating responsibility for providing that protection.
- Making it clear what users are allowed to do, and what they must or must not do.
- Providing grounds on how to interpret and resolve conflicts in policies later on.

Security Planning, Policies, and Mechanisms

- Security mechanisms are employed to ensure that security policies are being adequately enforced.
- Mechanisms can be based on people, software, hardware, or other physical means (for example, cameras, locks, cabling, and so on).
- Choosing the right mechanisms can be difficult.
  - Can these mechanisms be trusted?
- Determining if the mechanisms fully implement the desired security policies is even harder.
  - You can try a formal proof, but those are complex and very time consuming.

Security Threats

- Security threats A to D can be handled by providing secure transmission - cryptographic methods
- Threat E and similar types managed by access control methods
- Other types of security threats
  - Illegal access of server computing system (webjacking)
  - Illegal access client computing system
  - Unauthorized use of client information
  - Denial of Service

Information Security Threats

- Internet Cryptography Techniques
- Transport Layer Security
- Application Layer Security
- Server Proxies and Firewalls

Purpose of Cryptography

- Secure stored information - regardless if access obtained
- Secure transmitted information - regardless if transmission has been monitored

Services Provided by Cryptography

- Confidentiality
  - provides privacy for messages and stored data by hiding
- Message Integrity
  - provides assurance to all parties that a message remains unchanged
- Non-repudiation
  - Can prove a document came from X even if X' denies it
- Authentication
  - identifies the origin of a message
  - verifies the identity of person using a computer system

Cryptography

- Encryption Overview
  - Plain text is converted to cipher text by use of an algorithm and key.
    - Algorithm is publicly known
    - Key is held private
  - Three Main Categories
    - Secret Key
      - single key is used to encrypt and decrypt information
    - Public/Private Key
      - two keys are used: one for encryption (public key) and one for decryption (private key)
    - One-way Function
      - information is encrypted to produce a "digest" of the original information that can be used later to prove its authenticity

Encryption Techniques

- Secret Key (Symmetric)
  - Sender and receive have the same secret key that will encrypt and decrypt plain text
  - Strength of encryption technique depends on key length
  - Known symmetrical algorithms
    - Data Encryption Standard (DES)
      - 56 bit key
    - Triple DES, DESX, GDES, RDES
      - 168 bit key
    - RC2, RC4, RC5
      - variable length up to 2048 bits
    - IDEA - basis of PGP
      - 128 bit key
    - Blowfish
      - variable length up to 448 bits

Encryption Techniques (con't)

- Asymmetric Encryption (Public/Private Key)
  - user X has a pair of keys one public and one private
  - To encrypt a message to X use X's public key
  - X will decrypt encrypted message using X's private key that "matches" X's public key
  - Most common algorithm is the RSA (Rivest Shamir Adelman) algorithm with key lengths from 512 to 1024 bits.

Encryption Techniques (con't)

- One-Way Function
  - non-reversible "quick" encryption
  - produces a fixed length value called a hash or message digest
  - used to authenticate contents of a message
  - Common message digest functions
    - MD4 and MD5
      - produces 128 bit hashes
    - SHA
      - produces 160 bit hashes

Cryptographic Services Allow

- Digital Signatures
  - sign messages to validate source and integrity of the contents
- Digital Envelopes
  - secure delivery of secret keys
- Message Digests
  - short bit string hash of message
- Certificates (Digital Ids)
  - used to authenticate: users, web sites, public keys of public/private pair, and information in general
- Secure Channels
  - Encryption can be used to create secure channels over private or public networks

Digital Signatures

- Digital Signature
  - Encrypt sender's identity string with sender's private key
  - Concatenate the encrypted text and the identity string together
  - Encrypt this message with receiver's public key to create message
  - Receiver decrypts the encrypted text with their private key
  - the cypher text portion of the message is decrypted with sender's public key
  - The decrypted text can be compared with the normal text to checks its integrity

Digital Envelope

- Public/Private key encryption / decryption useful for internet
- Limitations
  - encryption / decryption slow
  - not reasonable for large documents
- Combine symmetric and asymmetric methods
  - sender creates and uses symmetric (session) key to create cipher text
  - sender uses receiver's public key to encrypt the symmetric key - digital envelope
  - sender transmits both cipher text and digital envelope to receiver

Message Digests

- How to create and use a message digest
  - sender uses message as input to digest function
  - "sign" (encrypt) output (hash) with sender's private key
  - send signed hash and original message (in plain text) to receiver
  - receiver decrypts hash with sender's public key
  - receiver runs plain text message through digest function to obtain a hash
  - if receiver's decrypted hash and computed hash match then message valid.

Digital Certificates (ID)

- Certification Authorities (CA)
  - used to distribute the public key of a public/private pair
  - guarantees the validity of the public key
    - does this by verifying the credentials of the entity associated with the public key
  - Some Case
    - Versign - http://www.versign.com
    - U.S. Post Office - http://www.ups.gov
    - CommerceNet - http//www.commerce.net
  - certificates contain
    - public key
    - e-mail
    - full name
- Digital certificates are secure
  - cannot be forged nor modified

Digital Certificates

- Process to create Digital Certificate
  - User generates public/private pair
  - User creates and sends a certificate request
    - contains: identifying information and user's public key
  - CA verifies this information
  - CA creates a certificate containing user's public key and information
  - CA creates message digest from certificate and signs it with CA's private key
  - This a signed certificate

Digital Certificates

- Using a Digital Certificate
  - before sending a secure message sender request a signed certificate from receiver
  - sender decrypts signed certificate with CA's known public key to obtain message digest of info and public key provided to CA by receiver
  - sender creates a message digest of public key and info provided by the receiver for sender's use
  - sender compare the message digests if they match then receiver is validated.

Digital Certificates

- Types of Digital Certificates
  - site certificates
    - used to authenticate web servers
  - personal certificates
    - used to authenticate individual users
  - software publishers certificates
    - used to authenticate executables
  - CA certificates
    - used to authenticate CA's public keys
  - All certificates have the common format standard of X.509v3

Secure Channels

- Encrypted Traffic may use
  - Symmetric Key
  - Public/Private Key
- Negotiated Secure Session
  - Secure Socket Layer (SSL)
  - Transport Layer Security (TLS)
  - SSL or TLS provides these services
    - Authenticate users and servers
    - Encryption to hide transmitted data - symmetric or asymmetric
    - Integrity to provide assurance that data has not been altered during transmission
  - SSL or TLS require certificates to be issued by a CA

Secure Channels (con't)

- Internet Tunnels
  - virtual network circuit across the Internet between specified remote sites
  - uses an encrypting router that automatically encrypts all traffic that traverses the links of the virtual circuit
- Tunneling Protocols
  - PPTP by Microsoft - http://www.microsoft.com
  - Layer 2 Forwarding (L2F) by Cisco - http://www.cisco.com
  - L2TP (combines PPTP and L2F) - http://www.ietf.com

Secure Sockets Layer

- SSL History
  - Competitor to S-HTTP
  - S-HTTP an extension of HTTP
  - General purpose encryption system using symmetric encryption
  - S-HTTP only encrypts Web protocols
  - Three versions v1.0, v2.0 and v3.0
    - SSL v3.0 implemented in Netscape 3.0 and Internet Explorer 3.0 and higher
    - SSL v3.0 supports Diffie-Hellman anonymous key exchange and Fortezza smart card

Secure Sockets Layer

- SSL Characteristics
  - Operates at the TCP/IP transport layer
  - Encrypts (decrypts) input from application (transport) layer
  - Any program using TCP can be modified to use SSL connections
  - SSL connection uses a dedicated TCP/IP socket (e.g. port 443 for https or port 465 for ssmtp)

Secure Sockets Layer

- SSL Characteristics
  - SSL is flexible in choice of which symmetric encryption, message digest, and authentication algorithms can be used
  - When SSL client makes contact with SSL server they try to pick strongest encryption methods they have in common.
  - SSL provides built in data compression
    - compress first then encrypt

Secure Sockets Layer

- SSL Characteristics
  - When SSL connection established browser-to-server and server-to-browser communications are encrypted. This includes:
    - URL of requested document
    - Contents of the document
    - Contents of browser forms
    - Cookies sent from browser to server
    - Cookies sent from server to browser
    - Contents of HTTP header
    - But NOT particular browser to particular server
      - socket addresses not encrypted
      - can use proxy server for privacy

Secure Sockets Layer

- Establishing an SSL Connection
  - The client (browser) opens a connection to server port
  - Browser sends “client hello” message. Client hello message contains:
    - version of SSL browser uses
    - ciphers and data compression methods it supports
  - The Server responds with a “server hello” message. Server hello message contains
    - session id
    - the chosen versions for ciphers and data compression methods.

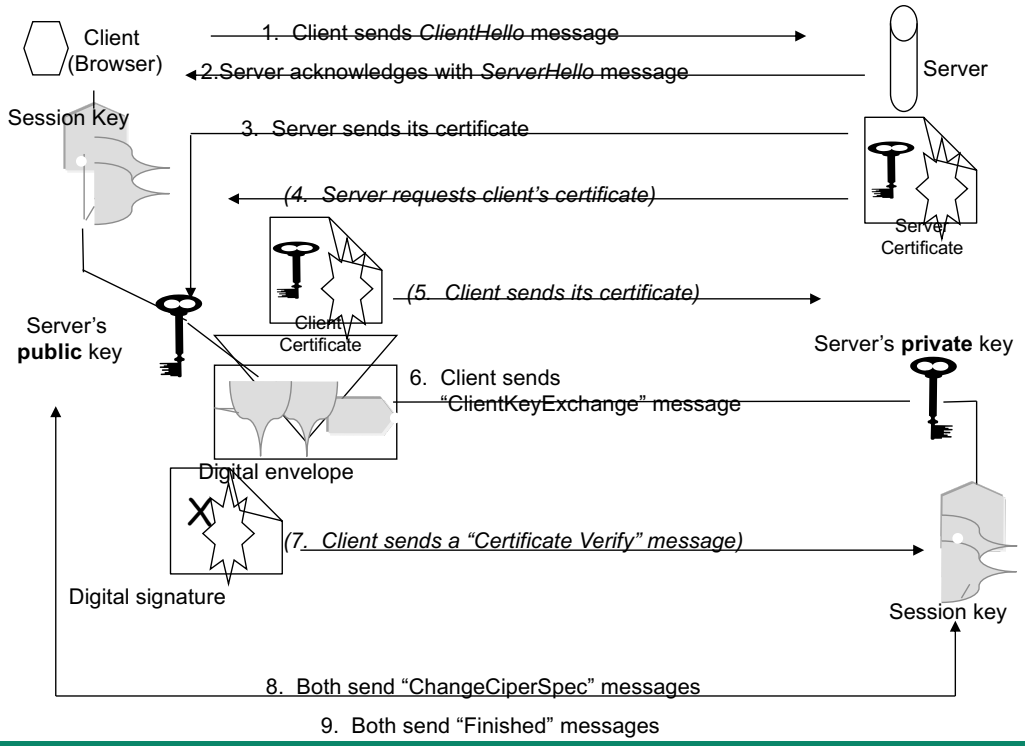
Secure Sockets Layer

- Establishing an SSL Connection (con't.)
  - The server sends its certificate
    - used to authenticate server to client
  - Optionally the server may request client's certificate
  - If requested, client will send its certificate of authentication
    - if client has no certificate then connection failure
  - Client sends a "ClientKeyExchange" message
    - symmetric session key chosen
    - digital envelope is created using server's public key and contains the symmetric session key

Secure Sockets Layer

- Establishing an SSL Connection (con't.)
  - Optionally, if client authentication is used the client will send a certificate verify message.
  - Server and client send "ChangeCipherSpec" message indicating they are ready to begin encrypted transmission.
  - Client and server send "Finished" messages to each other
    - These are a message digest of their entire conversation up to this point.
    - If the digests match then messages were received without interference.

SSL Connection Setup



Transport Layer Security (TLS)

- IETF (Internet Engineering Task Force) Standard for secure connection
- Derivative of SSLv3.0
- Uses different digest functions and different set of encryption algorithms
- see TLS URL for more details
  - http://www.consensus.com/ietf-tls/
- see SSL URL for more details
  - WBSRV = home.netscape.com/
    - WBSRV/newsref/std/SSL.html
    - WBSER/ref/internet-security.html

Making Secure Client/Server Interaction Using Java

- The Java Secure Socket Extension (JSSE) provides a framework and a Java implementation of the SSL and TLS protocols.
- It provides mechanisms for data encryption, server authentication, message integrity, and optional client authentication

Making Secure Client/Server Interaction Using Java

- Steps
  - Generate server certificate/key pair to be used by SSL.
    - Create a Keystore.
    - Create an RSA key for the Keystore (Use genkey option in keytool) that are valid for 120 days.
    - Export the RSA certificate (without the private key) using the keytool
    - Import the RSA certificate into a new file
  - Write the SSL code, run and test.
    - Use Java JSSE Socket Factories (SSLSocketFactory , SSLServerSocketFactory in javax.net.ssl and java.security)

Making Existing Servers Secure

```
import java.io.*;
import javax.net.ssl.*;
public class Server {
        int port = portNumber;
        SSLServerSocket server;
        try {
                SSLServerSocketFactory factory =
                (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
                server = (SSLServerSocket)
                factory.createServerSocket(portNumber);
                SSLSocket client = (SSLSocket) server.accept();

        //  Create input and output streams as usual
        //  send secure messages to client through the
        //  output stream
        //  receive secure messages from client through
        //  the input stream
        } catch(Exception e) {
                }
}
```


Make an Existing Client Secure

```
import java.io.*;
import javax.net.ssl.*;
public class Client {
...
try {
    SSLSocketFactory factory = (SSLSocketFactory)
    SSLSocketFactory.getDefault();
    SSLSocket client = (SSLSOcket) factory.createSocket(serverHost, port);
    //  Create input and output streams as usual
    //  send secure messages to server through the
    //  output stream receive secure
    //  messages from server through the input stream
    } catch(Exception e) {
        }

}
```

Making Secure Client/Server Interaction Using C

- C does not have built-in functions like Java
  - Think 1970!
- Need to use OpenSSL library
  - A free implementation of SSL/TLS based on Eric Young's SSLeay package

Making Secure Client/Server Interaction Using C

- Steps
  - Initialize the SSL library with SSL_library_init()
  - load the error strings SSL_load_error_strings()
  - Create an object to be used as an error-printing context
  - Load your own public/private key pair and the associated certificate using SSL_CTX_use_certificate_chain_file()
  - Use SSL_CTX_use_PrivateKey_file() to load the private key
  - For security reasons, the private key is encrypted using a password.
    - If it is, the password callback (set using SSL_CTX_set_default_passwd_cb()) will be called to obtain the password.
  - If you're going to be authenticating the host to which you're connected, OpenSSL needs to know what CAs you trust.
    - The SSL_CTX_load_verify_locations() call is used to load the CAs.
  - SSL needs a good source of strong random numbers and uses /dev/urandom to seed the RNG (Linux)

Client

- Create a TCP connection between client and server on our own and then use the TCP socket to create
- Create an SSL object to handle the connection.
- Attach the SSL object to the socket
  - Use a BIO object using the socket and then attach the SSL object to the BIO.
  - This abstraction layer allows the user to use OpenSSL over channels other than sockets

```
/* Connect the TCP socket*/
sock=tcp_connect(host,port);

/* Connect the SSL socket */
ssl=SSL_new(ctx);
sbio=BIO_new_socket(sock,BIO_NOCLOSE);
SSL_set_bio(ssl,sbio,sbio);

if(SSL_connect(ssl)<=0)
  berr_exit("SSL connect error");
if(require_server_auth)
  check_cert(ssl,host);
```


Server

- Verify that the certificate you're looking at matches the identity that you expect the server to have
  - extract the server's certificate using SSL_get_peer_certificate() and then compare the common name to the hostname we're connecting to

```
request_len=strlen(REQUEST_TEMPLATE)+strlen(host)+6;
if(!(request=(char *)malloc(request_len)))
  err_exit("Couldn't allocate request");
sprintf(request,REQUEST_TEMPLATE, host,port);/* Find the exact request_len */
request_len=strlen(request);
r=SSL_write(ssl,request,request_len);
switch(SSL_get_error(ssl,r)){
  case SSL_ERROR_NONE:
    if(request_len!=r)
      err_exit("Incomplete write!");
    break;
    default:
      berr_exit("SSL write problem");
}
```

```
void InitializeSSL()
{
    SSL_load_error_strings();
    SSL_library_init();
    OpenSSL_add_all_algorithms();
}
void DestroySSL()
{
    ERR_free_strings();
    EVP_cleanup();
}
void ShutdownSSL()
{
        SSL_shutdown(ssl);
        SSL_free(ssl);
}
```



```
int sockfd, newsockfd;
SSL_CTX *sslctx;
SSL *cSSL;

InitializeSSL();
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd< 0)
{
    //Log and Error
    return;
}
struct sockaddr_in saiServerAddress;
bzero((char *) &saiServerAddress, sizeof(saiServerAddress));
saiServerAddress.sin_family = AF_INET;
saiServerAddress.sin_addr.s_addr = serv_addr;
saiServerAddress.sin_port = htons(aPortNumber);

bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr));

listen(sockfd,5);
newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);

sslctx = SSL_CTX_new( SSLv23_server_method());
SSL_CTX_set_options(sslctx, SSL_OP_SINGLE_DH_USE);
int use_cert = SSL_CTX_use_certificate_file(sslctx, "/serverCertificate.pem" , SSL_FILETYPE_PEM);

int use_prv = SSL_CTX_use_PrivateKey_file(sslctx, "/serverCertificate.pem", SSL_FILETYPE_PEM);

cSSL = SSL_new(sslctx);
SSL_set_fd(cSSL, newsockfd );
//Here is the SSL Accept portion.  Now all reads and writes must use SSL
ssl_err = SSL_accept(cSSL);
if(ssl_err <= 0)
{
    //Error occurred, log and close down ssl
    ShutdownSSL();
}
```

Read and Write

```
SSL_read(cSSL, (char *)charBuffer, nBytesToRead);
SSL_write(cSSL, "Hi :3\n", 6);
```


Application Layer Security

- Secure Electronic Transactions
  - SET
- Digital Payment Systems
  - First Virtual
  - CyberCash
  - DigiCash
  - Millicent
- Pretty Good Privacy
  - PGP: used to secure e-mail
- These are the applications sender/receiver use to give secure communication

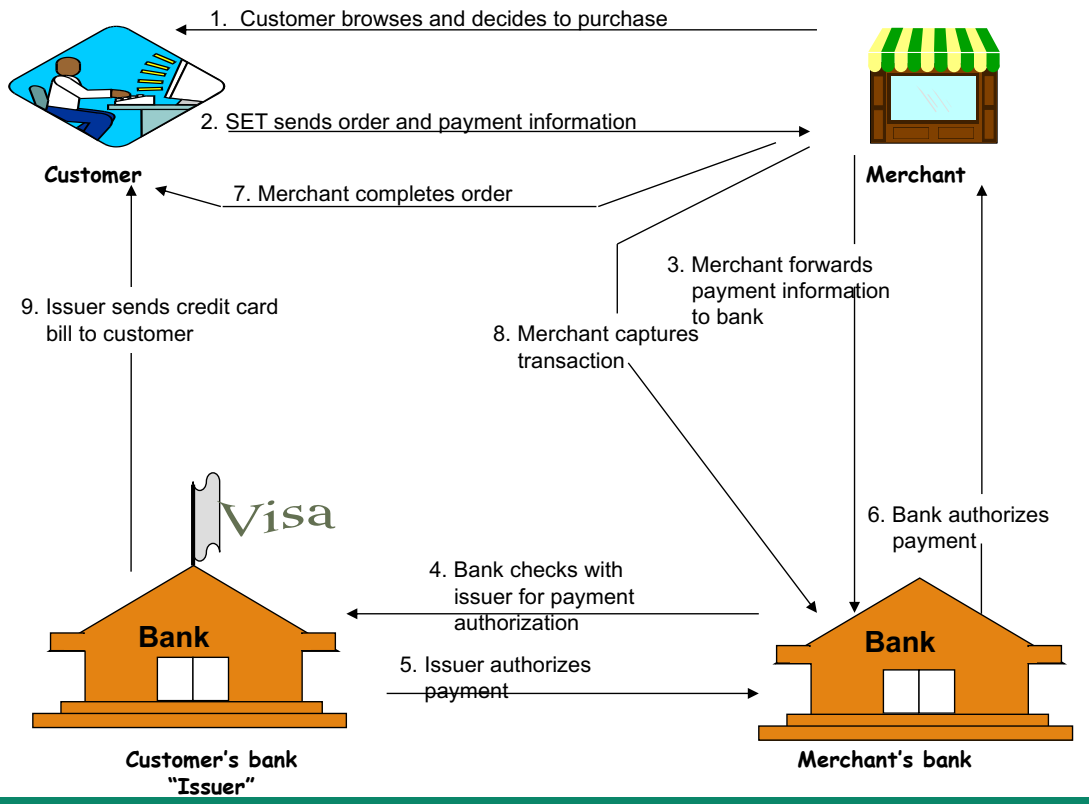
Secure Electronic Transactions

- Cryptographic protocol
- Developed by Visa, Mastercard, Netscape, and Microsoft
- Used for credit card transactions on the Web
- Provides
  - Authentication of all parties in transaction
  - Confidentiality: transaction is encrypted to foil eavesdroppers
  - Message integrity: not possible to alter account number or transaction amount
  - Linkage: attachments can only be read by 3rd party if necessary

Secure Electronic Transactions

- SET protocol supports all features of credit card system
  - Cardholder registration
  - Merchant registration
  - Purchase requests
  - Payment authorizations
  - Funds transfer (payment capture)
  - Chargebacks (refuns)
  - Credits
  - Credit reversals
  - Debit card transactions
- SET can manage
  - real-time & batch transactions
  - installment payments

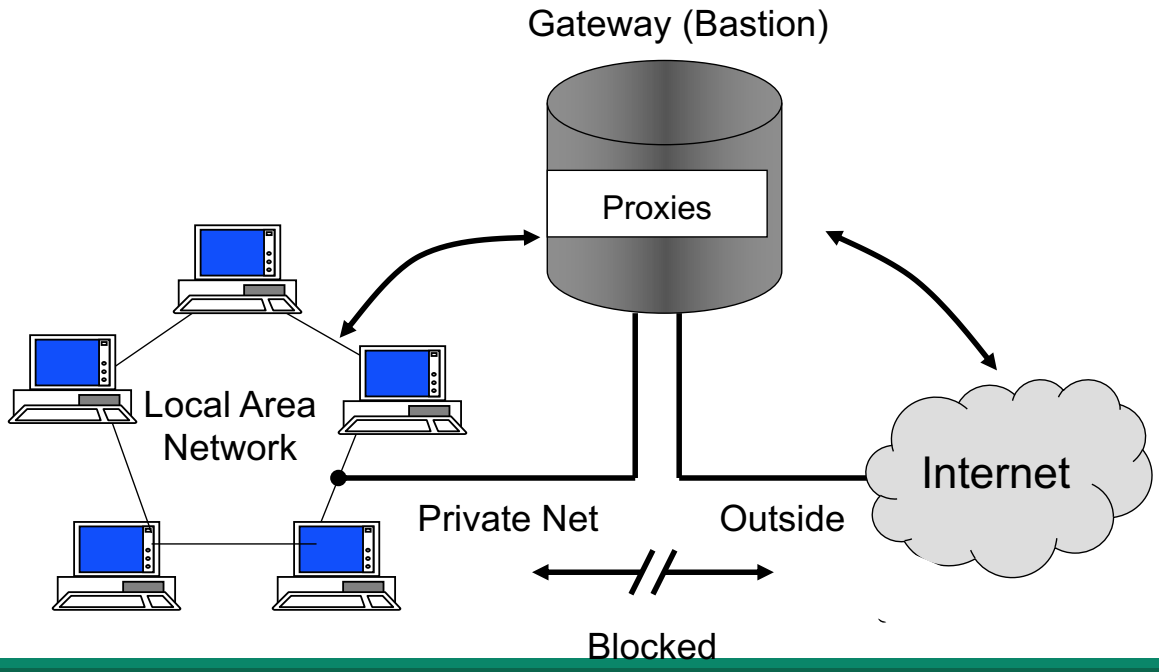
Secure Electronic Transaction



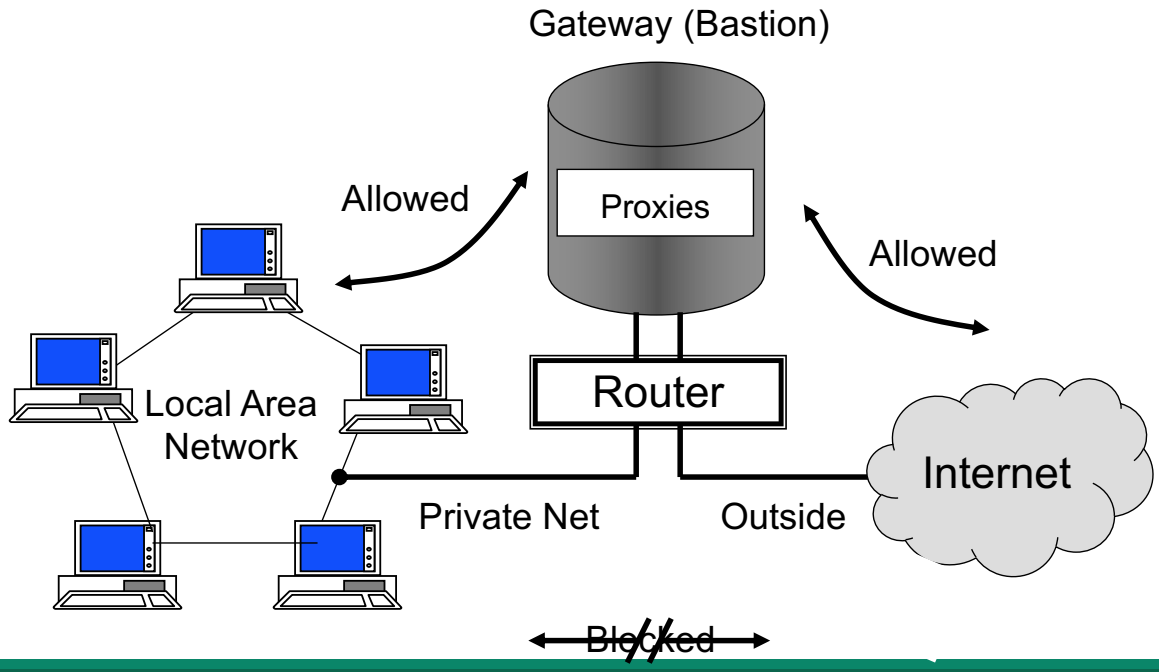
Securing Private Networks

- Minimize external access to LAN
- Done by means of firewalls and proxy servers
- Firewalls provide a secure interface between an “inner” trusted network and “outer” untrusted network
- every packet to and from inner and outer network is “processed”
- Firewalls require hardware and software to implement
- Three main hardware architectures
  - dual-homed host
  - screened gateway
  - screened subnet gateway

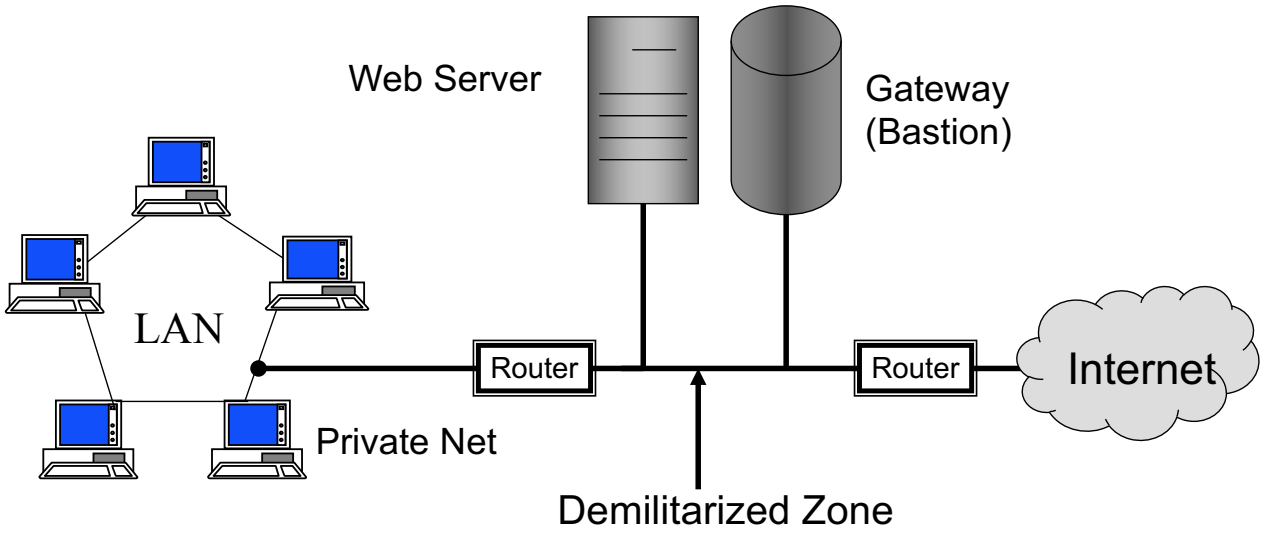
Dual Homed Gateway



Screened Host Gateway



Screened Subnet Gateway



Securing Private Networks

- Software that is used are proxies and filters that allow or deny network traffic access to either network
- Proxy programs
  - application-level
  - circuit-level
- Filters
  - packet filtering

Securing Private Networks

- Application level proxies
  - written for each particular protocol
    - e.g. HTTP or FTP or SMTP
  - regardless of protocol its function is to forward or not forward messages across firewall
  - they decide based on TCP/IP information
    - e.g. source and destination ports and IP addresses
  - they decide based on content of message
    - e.g. do not forward on and message containing VB executable or ActiveX components

Securing Private Networks

- Circuit level proxies
  - software's function is to forward or not forward packets across firewall
  - decides only on basis of header information in the packet
    - i.e. source and destination IP addresses and port numbers
  - they cannot peek into packet
  - advantage
    - very fast - less computation required
    - very general - handle many protocols
  - SOCKS
    - freeware circuit level proxy
  - SMLI
    - stateful multilayer inspection gateway
    - correlates incoming and outgoing packets

Securing Private Networks

- Packet Filtering
  - technically not software
  - used with screen host or screened subnet host architecture
  - uses router's routing table to decide which packets to forward or not forward
  - if bastion does not have proxy for a given service (e.g. TFTP) then packet filter can be configured to bypass firewall

Access Security Threats

- Access Control
  - Threats
    - Webjacking: site vandalism
  - Countermeasures
    - User Authentication
    - User Authorization
- Denial of Service
  - Threat
    - Unable to user server resources
    - Type of DOS Attacks
  - Counter Measures (limited)
    - Firewalls
    - System Configuration

Access Control

- User authentication
  - process used to identify user who accesses a web server
  - determines legitimate user
  - Generally referred to as access control
- User authorization
  - once user authenticated specifies what server resources that user may access
  - resources are: files, scripts, and directories

User Authentication

- Several type of access control
  - Based on IP address
    - validates web browser based on its host's IP address
  - Based on Domain Name
    - validates web browser based on its host's domain name
  - Based on user name and password
    - User of browser is validated on basis of user ID and its associated password
  - Based on client certificates
    - remote user is issued a secure certificate to use as a digital signature
  - Based on network security protocols
    - solves validation problems associated with accessing via LAN and WAN
    - e.g. Kerberos and DCE

Authentication based on host IP address and/or DNS name

- Screen browsers based on their source IP address, Domain Name, network,or subnetworks
- Advantages
  - easy to set up
  - not likely to be incorrectly configured
- Disadvantages
  - difficult to grant access to users who migrate
  - difficult hand DHCP protocol and Web proxies
  - security issues of
    - DNS spoofing
    - IP spoofing

Countermeasures to DNS Spoofing

- DNS Spoofing
  - Attacker assumes control if DNS host/name lookup system
- Counter by
  - Paranoid DNS checking
    - Upon receiving packet from browser server uses that source IP address to make two DNS requests
    - First resolves IP address to get a Domain Name
    - Returned domain name used to find its IP address
    - if domain name correlates with IP address then legitimate remote host
  - Use a firewall's DNS lookup

Countermeasures to IP Spoofing

- IP spoofing requires technical expertise
- Uses source routing protocol
  - appears as if request originates from within LAN
  - can be used to insert CGI script or modify OS
- Prevented by
  - configuring routers and firewalls to reject connections using source routing protocol
  - configure the server's operating system to reject connections using source routing

Authentication Based on User ID and Password

- Requires user to provide protected information in order to be authenticated
- Advantages
  - Authenticates users not hosts
  - Users can migrate from host to host
  - No problems with Web proxies or DHCP
- Disadvantages
  - Users share passwords, forget passwords, do not keep passwords private, or choose poor passwords
  - passwords can be "sniffed" if transmitted over a network

Authentication Based on User ID and Password

- Countermeasures to disadvantages
  - Users share passwords, forget passwords, do not keep passwords private, or they choose poor passwords
    - User education
    - Chose hard passwords but easy to remember

Authentication Based on User ID and Password

- Countermeasures to disadvantages
  - passwords can be “sniffed” if transmitted over a network
    - Basic authentication is carryout in plain text but coded in Base 64 MIME - HTTP/1.0
    - Can be intercepted and decoded
    - Since HTTP protocol stateless every access to protected resource needs to be authenticated
    - Basic Authentication process occurs frequently hence more opportunity to be sniffed.
  - Use secure transmissions
    - HTTP/1.1 uses Digest Authentication process
    - Use encrypted communications e.g. SSL connection

Client Based Certificate System

- Certificates
  - when user logs on (presents their certificate) the authentication server verifies the certificate is valid by opening it with the CA's public key
  - certificate contains users public key and personal information.
  - Server sends a challenge to the user - a one-time value the user signs with their private key
  - Server then signs the same value with its copy of the user's private key
  - If the signatures match then user is authenticated

Other Forms of Access Control

- Kerberos authentication model
  - Uses a secure "key server"
  - Once user authenticated free to use any resources of the system
  - All transmissions are encrypted
- Distributed Computing Environment
  - DCE is designed by Open Software Foundation
  - Similar to Kerberos authentication model
- Two Factor Authentication
  - need something you have - ATM card
  - need something you know - PIN number

Other Forms of Access Control

- Smart Card Type
  - token access device that has information that is in sync with server information (e.g. counter, time, random number generator, etc.)
  - "One time pad" of user name and password

Denial of Service

- Some Types of Attack
  - TCP/IP SYN attack
    - To set TCP/IP connection use a three step "handshake" protocol
      - client requests
      - server acknowledges and waits
      - client acknowledges
    - if no client acknowledgement or many client requests then server overwhelmed.
  - PING of Death
    - many clients "ping" server
  - Flood server with URL requests
    - either one client or many in parallel
    - DDOS attack

Denial of Service

- Countermeasures to DOS
  - Minimal counter measures after attack has started
    - DOS attacks require client(s) to carry requests
    - locate source(s) of requests and terminate those processes
  - Countermeasures prior to attack
    - prevent attacks by making sure all hosts a going to be used legitimately
    - requires securing all remote hosts - not likely
    - e.g. DDOS: number of freeware programs that when run will create SYN flooding attack make sure remote host does not run this program.