CSE3231 Final Exam Study Guide

Grant Butler, Tyler Zars

gbutler2020@my.fit.edu | tzars2019@my.fit.edu

Transport Layer

UDP and TCP

Differences:

TCP (Transport Control Protocol)

Advantages

- Connection oriented transport
  - Sender must establish connection before transmission
  - Sender notified of delivery or of error
- Byte-stream service
  - Data transmission and reception are similar to file I/O
- Reliable delivery
  - Garuntees packets are assembled in order

Connection Management

- Connection
  - Three way connection increases probability that both endpoints know that connection was acccepted.
- Termination
  - Four way handshake requires two FIN and two ACK to complete. Ensures proper termination of connection occurs.

UDP (User Datagram Protocol)

- Connectionless
  - Destination address and port number are added to transport segment's header and the segment is sent to the destination.
  - No confirmation of error or delivery (ACK) is returned.
    - Unreliable because of no ACK
- Advantages
  - Applications pass **directly** to transport layer.
  - Data is transmitted *immediately*. Will either reach the receiver or not at all.
  - Less to manage:
    - No *congestion-control* or retransmission mechanisms.
    - No Connection Establishment
    - No connection state
    - Results in small packet header
  - Messages can be sent in broadcast or multicast mode.
    - one to multiple receivers (multicast) or all nodes (broadcast)

TCP Header Fields

Field	Size (bits)	Description
Source Port	16	Identifies source port number (sender's TCP port)
Destination Port	16	Identifies destination port number (receiving port)
Sequence Number	32	Used to number TCP segments. If SYN = 0, each byte is assigned a sequence number.[*]
Acknowledgement Number	32	Indicates next sequence number that sending device is expecting.
Offset (Header Length)	4	Shows number of 32 bit words in header. Minimum size of 5 words (`0101` in binary).
Reserved	4 (6)	Always set to 0.
TCP Flags	8	Flags are: URG, ACK, PSH, RST, SYN, FIN
Window	16	The size of the receive window, which is the number of bytes beyond sequence number in Acknowledgement field that the receiver is willing to receive.
Checksum	16	Used for error checking of header and data.
Urgent Pointer	16	Shows the end of urgent data so interrupted data streams can be continued.[+]
TCP Options	variable	0 → End of Options List, 1 → No Operations (NOP, Pad), 2 → Maximum segment size, 3 → Window Scale, 4 → Selective ACK ok, 8 → Timestamp

Size: 4 bits → nibble, 8 bits → byte, 32 bits → word
Sequence Number: If SYN = 1, then this is the initial sequence number. The sequence number of the first byte of data will then be this number + 1. *i.e.*: Let first byte of data have this be 300. Then if a packet has 10 bytes, then the next packet sent will have a sequence number of `300 + 10 + 1 = 311`.
Urgent Pointer: When URG is set, the data is given priority.

TCP Flags Explained

Flag	Description
URG	Urgent Pointer.
ACK	Acknowledgement
PSH	Push function. TCP allows an application to specify that data is to be pushed immediately.
RST	Reset connection. Receiver must respond immediately terminating the connection. Transfer of data ceases, so data in transit is lost. Used for abnormal close of TCP connection, unlike FIN.
SYN	Indicates synchronized sequence numbers. Source is beginning a new sequence.
FIN	Set when no more data is to come from sender. Used for good closing of TCP connection, unlike RST.

TCP Flow Control

TCP needs to control amount of data a sender transmits to avoid overwhelming the receiver.

Congestion Control vs Flow Control

Congestion Control

- focuses on preventing too much data in network.
- uses Retransmission Timeout (RTO) and Round Trip Time (RTT)
    - RTT is different for each path a packet takes.
- A router might only be able to handle 100 Mb/s total, but two senders could send more than that.

Flow Control

- Tries to prevent senders from overrunning capacity of receivers.
    - Can't prevent congestion at routers.
- Uses sliding window to control traffic in transit.
    - Uses AdvertisedWindow to indicate how much data it can handle.
    - Measures in bytes, not packets.
    - Limits how many unacknowledged bytes can be in transit at a time.
    - TCP vs Data-Link Sliding Windows
        - Data-Link layer controls transmission of frames over links between adjacent nodes.
            - one sender at a time
            - always arrive in order sent (unless frames are lost)
        - TCP deals with end-to-end flow
            - each receiver can have multiple senders
            - each packet can follow a different path
    - Header uses these fields to manage flow control:
        - *SequenceNum*
        - *Acknowledgement*
        - *AdvertisedWindow*

TCP Congestion Control: Additive Increase and Multiplicative Decrease (AIMD)

TCP Source sets the CongestionWindow based on level of congestion it *perceives* in the network.

- Involves decreasing congestion window when congestion goes up and increasing the congestion window when level of congestion goes down.
- This is called *Additive Increase / Multiplicative Decrease* (AIMD).

Additive Increase

- Every successful send from source that is a *CongestionWindow*'s worth of packets adds the equivalent of 1 to CongestionWindow.
  - Success is measured as one ACK per RTT.
- Increase is slower than decrease and avoids too rapid an increase in transmission rate.

Multiplicative Decrease

- Easier to understand in terms of packets, despite CongestionWindow being measured in bytes.
  - *e.g.:*
    - CongestionWindow is 16 packets
    - If a loss is detected, CongestionWindow is set to 8.
    - Additional losses go → 4, 2, 1.

Slow Start

1. Source starts *CongestionWindow* at one packet.
2. Sends one packet.
3. ACK arrives → *CongestionWindow* += 1.
4. Two packets are sent.
5. Two ACKs → *CongestionWindow* += 2.

Trend: TCP effectively **doubles** every RTT.

1. **Slow Start** begins by doubling *CongestionWindow* size.
2. When threshold is reached, switches to *additive increase*.
3. When packet is lost, *CongestionWindow* goes to 1 and slow start repeats.

TCP Timeout and RTT

Timeout

Timeout period must be long enough to allow longer paths. If a packet is lost, multiple packets can be sent out before timeout expires. Receiver can't ACK because missing packet caused a gap in *SequenceNum*. Sender can reach *CongestionWindow* limit while waiting for timeout.

Fast Retransmission and *Duplicate Acknowledgements*
Receiver sends ACK for later packets, but with ACK number of last packet before lost packet--*i.e.* duplicate acknowledgements.

- Tells sender that at least one packet hasn't arrived, but later ACK's indicate some later packets arrived.
- Duplicate ACK number tells sender which packet wasn't received.

Sender can *resend* missing packet without waiting for timeout to expire. This is called *fast retransmission* and can trigger transmission of lost packets sooner than regular timeout.

- Not triggered until *three duplicate ACK's* arrive.
- Sender knows packed was lost, and halves *slow start threshold* and goes into slow start.

Fast Recovery

- Lost packed decreases *CongestionWindow* to one and starts slow start.
- Fast retransmission signals congestion, and instead of lower *CongestionWindow*, fast recovery uses ACKs in transit to trigger sending of new packets.
- Removes slow start phase when fast retransmit detects lost packet.

Round Trip Time (RTT)

Retransmission TimeOut (RTO) is based on Round Trip Time (RTT) for a given connection.

- At connection, sender and receiver determine RTT and sender uses that for RTO.
- Sender calulates an average RTT to deal with delays.

Determining RTT:

- Sender and receiver both need RTT, so they put timestamps in options field to track send and receive times.
- A *Smoothed RTT* (SRTT) is calculated based on the SRTT averaged over time and the most recent RTT.

$$SRTT = \alpha * SRTT + (1 - \alpha)RTT \text{ where } \alpha = 0.9$$

- Smoothed RTT calculation was revised to include *variance* in RTT.
  - **Variance** measures how much RTT changes over time.

$$VarRTT = \beta * VarRTT + (1 - \beta) * |SRTT - RTT| \text{ where } \beta = 0.75$$

- Retransmission TimeOut (RTO) is calculated as follows:

$$RTO = SRTT + 4^{++} * VarRTT$$

++(multiplying by 4 is based on experimentation)

IP Checksum

Header Checksum: 16 bits

- A checksum on the header only. Since some header fields change, it is *recomputed* and verified each time the header is processed.
- Algorithm:
  - 16 bit one's complement of the one's complement sum of all 16 bit words in the header. The value of the checksum field is zero.

IP Checksum Example:

1. break sequence into 16-bit words
2. Add 16-bit values. Each carry-out produced is added to the LSb.
3. Invert all bits to get one's complement.

Header to check:

```
1000 0110 0101 1110
1010 1100 0110 0000
0111 0001 0010 1010
1000 0001 1011 0101
```

Add 16-bit values 2 at a time and convert to one's complement:

$$
\begin{array}{rl}
1000\ 0110\ 0101\ 1110 & first\ val \\
+\ 1010\ 1100\ 0110\ 0000 & second\ val \\
\hline
1\ 0011\ 0010\ 1011\ 1110 & carry-out \\
+\ 0000\ 0000\ 0000\ 0001 & add\ to\ LSb \\
\hline
0011\ 0010\ 11011\ 1111 & \\
+\ 0111\ 0001\ 0010\ 1010 & third\ val \\
\hline
0\ 1010\ 0011\ 1110\ 1001 & no\ carry-out \\
+\ 1000\ 0001\ 1011\ 0101 & fourth\ val \\
\hline
1\ 0010\ 0101\ 1001\ 1110 & carry-out \\
+\ 0000\ 0000\ 0000\ 0001 & add\ to\ LSb \\
\hline
0010\ 0101\ 1001\ 1111 & one^{\prime}s\ complement\ sum \\
 & flip\ bits \\
1101\ 1010\ 0110\ 0000 & one^{\prime}s\ complement
\end{array}
$$

Thus, the 16 bit checksum is `1101 1010 0110 0000`.

Applications

Application layer interfaces with transport layer, isolating applications from the details of packet delivery. Applications can use either UDP or TCP. Some use UDP but add features like acknowledgements on their own to get reliable delivery without TCP's overhead.

Protocols

Usually, an application that supports interaction and data transfer have a *protocol* for communication between nodes. One may be a *server*, collecting and delivering data, while another may be a client, requesting and providing data. Application Protocols describe how endpoints will interact to accomplish tasks.

Internet RFC (Request For Comment)

A set of application protocols that standardize actions across vendors. Examples include:

- Simple Mail Transfer Protocol (SMTP) *RFC 5321*
- Hypertext Transfer Protocol (HTTP) *RFC 2616/7540*
- File Transfer Protocol (FTP) *RFC 959*

Clear application protocols allow developers to create servers and clients that interact in established and predictable ways.

- Some inconsistencies happen when features are added that are not part of the protocol.

Port Numbers

Servers typically listen on well known ports for connections. A range of UDP and TCP ports are assigned to specific protocols, while others are not reserved.

- Ports 0-1023 are reserved by the Internet Assigned Numbers Authority (IANA).

Common ports and their services:

Service	Port
SSH	22
HTTP	80
NTP (Network Time Protocol)	123
IMAP (email)	143 or 220
LDAP (authentication protocol)	289
HTTPS (using TLS/SSL)	443

Request/Reply Protocols

Messages are transmitted by client and server to manage and exchange data. Often consist of text commands.

- *Stateful* protocols require client and server to keep track of current state of exchange.
- *Stateless* protocols might have a server not keep a record of exchanges and close connection after each message.

Publish-Subscribe Protocol (PubSub)

- **Publishers** make data available for *subscribers* who register to receive types of messages.
    - *Loosely coupled* to subscribers and produce whether or not it is used.
    - Has better scalability because publishers don't manage connections.

Examples:

- Apache Kafka
- Google Cloud Pub/Sub
- Data Distribution Service (DDS)

Message Queueing

Similar to PubSub, these protocols don't require servers to wait for a client to request. Important aspects include:

- A *queue manager* is implemented and announced to users.
- Applications *register* to be notified when messages arrive, and then can download those when ready.
- Applications can add messages to a queue.
- Queueing *decouples* senders and receivers, so senders don't wait before they add to the queue.

Examples:

- Apache ActiveMQ
- Microsoft Message Queueing
- Java Message Service

Peer to Peer

Each node can exchange messages with any other node.

- a node can be a client **and** a server
- data is *decentralized* and passed between peers
- not all nodes have same capabilities, so some may perform special tasks to help other nodes
- peer networks can be *structured* with set topology or *unstructured* and allow rapid changes to adapt

Examples:

- Bitcoin and other Cryptocurrency
- BitTorrent
- Gnutella

HyperText Transfer Protocol (HTTP)

Used for connections between clients (browsers) and servers in the World Wide Web. Provides request/response interaction between server and multiple clients.

- Web browser acts as *User Agent*
- Communications are based on TCP
- *Stateless* protocol
  - keep-alive feature was added in v1.1
  - retaining state information was solved with variables in messages or web cookies on client's host

Messages:

```
START_LINE <CRLF>
MESSAGE_HEADER <CRLF>
<CRLF>
MESSAGE_BODY <CRLF>
```

`START_LINE` indicates request or response message. Each section ends with CRLF.

Cookies

Coockies are used to support *stateful* client/server interactions

- server sends cookies (state) with response
- client stores them locally
  - client sends cookie with a new request to the server
- server extracts state information from cookie

Examples:

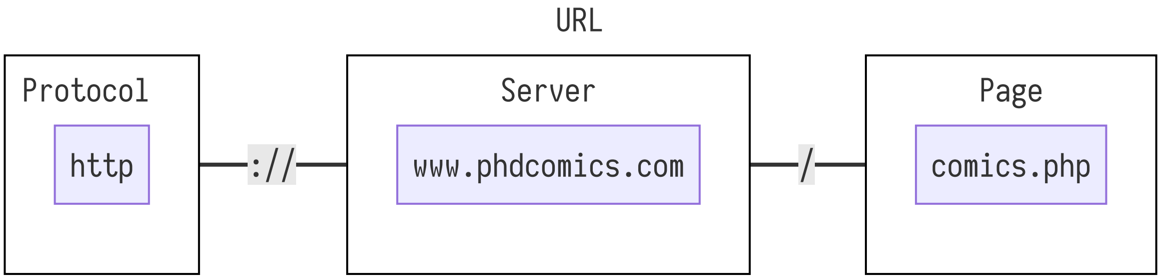
Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=297793521	15-10-10 17:00	Yes
jills-store.com	/	Cart=1-00501;1-07031;2-13721	11-1-11 14:22	No
aportal.com	/	Prefs=Stk:CSCO+ORCL;Spt:Jets	31-13-20 23:59	No
sneaky.com	/	UserID=4627239101	31-12-19 23:59	No

URLs

Identification of servers and hyperlinks is based on *Uniform Resource Locators* (URLs), which contain information to access a target server and document on that server.

- HTTP headers are text based and use standard header fields to manage connections and data exchange
- HTTP connections are not encrypted, but *HTTPS* creates an encrypted connection before data is exchanged
    - protects data, but does not authenticate users

Example:



- begins with protocol it will connect to
- specifies *domain name* (server) and *specific file* on the server
    - also can specify email accounts, local files, links to FTP servers, and other sources

History:

- links information in documents, presented by **Vannevar Bush** in 1945
- **Ted Nelson** coined hypertext in 1963 and helped create a system with hyperlinks
- **Douglas Engelbart** demonstrated a user interface with different tools and documents in 1968
- **Tim Berners-Lee** created a hypertext sharing system called the *World Wide Web* in 1990

HTTP Headers

Function	Example Headers
Browser capabilities (client → server)	User-Agent, Accept, Accept-Charset, Accept-Encoding, Accept-Language
Caching related (mixed directions)	If-Modified-Since, If-None-Match, Date, Last-Modified, Expires, Cache-Control, ETag
Browser context (client → server)	Cookie, Referrer, Authorization, Host
Content delivery (server → client)	Content-Encoding, Content-Length, Content-Type, Content-Language, Content-Range, Set-Cookie

HTTP Requests

Methods

Operation	Description
OPTIONS	Request information about available options
GET	Retrieve document in URL
HEAD	Retrieve metainfo about document in URL
POST	Give information to server
PUT	Store document under URL
DELETE	Delete URL
TRACE	Loopback request message
CONNECT	For use by proxies

Example:

```
GET /download.html HTTP/1.1
Host: www.ethereal.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1;
en-US; rv:1.6) Gecko/20040113
Accept:
text/xml,application/xml,application/xhtml+xml,text/htm
l;q=0.9,text/plain;q=0.8,image/png,image/jpeg,image/gif
;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.ethereal.com/development.html
```

HTTP Response

Steps server takes to serve pages:

1. Accept TCP connection
2. Receive page request and map it to requested resource

- May be static or requires execution of a program

3. Send reply to client

- May include links to other resources that client's browser has to access

4. Release idle TCP connections

Responses begin with a `START_LINE`, just like requests. The line specifies:

- The version of HTTP being used
- A three digit code indicating whether request was successful
- A text string giving reason for response.

Response Codes:

Code	Meaning	Examples
`1XX`	Information	`100` = server agrees to handle request
`2XX`	Success	`200` = request succeeded; `204` = no content present
`3XX`	Redirection	`301` = page moved; `304` = cached page still valid
`4XX`	Client error	`403` = forbidden page; `404` page not found
`5XX`	Server error	`500` = internal server error; `503` = try again later

Example:

```
HTTP/1.1 200 OK
Date: Thu, 13 May 2004 10:17:12 GMT
Server: Apache
Last-Modified: Tue, 20 Apr 2004 13:17:00 GMT ETag: "9a01a-4696-7e354b00"
Accept-Ranges: bytes
Content-Length: 18070
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1
```

Static vs. Dynamic HTTP Pages

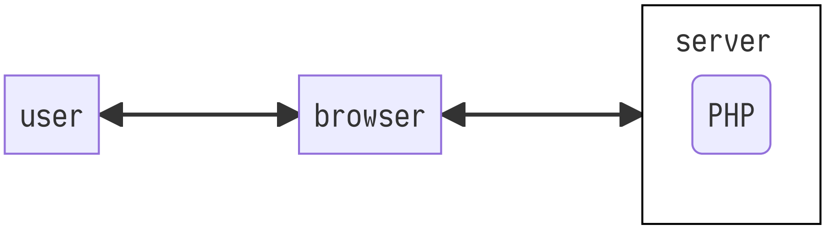
Static

- pre-built files
- content doesn't change between viewing, unless edited by administrator
- often written in HTML or similar languages
- can still have interactive parts, containing text and images
- style sheets are used to customize presentation

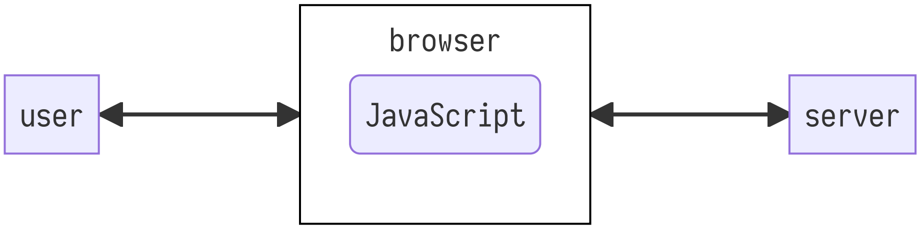
Dynamic

- generated by programs running on client/server
- created on demand by software
- sometimes generated based on user input or data from another database
- written in languages like PHP, JavaScript/TypeScript, Ruby, Python, etc.

Server-side scripting with PHP



Client-side scripting with JavaScript



DOM (Document Object Model)

Representation as a tree that web pages can alter. Dynamic pages use this to change their structure by altering attributes in elements on the page.

XML is a markup language that focuses on document structure and not page presentation.

Email

RFC 822 defined email to have a *header* and a *body*. Originally only ASCII text, but extended with the Multipurpose Internet Mail Extensions (MIME) protocol, allowing the email *body* to carry all types of data

Message Formats

email message header is lines of text terminated by *CarriageReturnLineFeed* (CRLF)

- each line has a type and value seperated by a colon (:)
- header is separated from message by a blank line
- *RFC 822* was extended by MIME to allow different types of data

Header Fields

message transport

Header	Meaning
To:	Email address of primary recipient
Cc:	address of secondary recipient
Bcc:	address of blind carbon copy
From:	Person who created message
Sender:	address of sender
Received:	line added by transfer agents along route
Return-Path:	Used to identify path back to sender

user agents

Header	Meaning
Date:	Date and time the message was sent
Reply-To:	address where replies should be sent
Message-Id:	Unique number for message
In-Reply-To:	`Message-Id` of message to which this is a reply
References:	Other relevant `Message-Id`s
Keywords:	User-chosen keywords
Subject:	Short summary of message for one-line display

ESMTP Example:

```
220 my.fit.edu ESMTP
EHLO gbutler2020
250-my.fit.edu
MAIL FROM:<gbutler2020@my.fit.edu>
250 2.1.0 Ok
RCPT TO:<tzars2019@my.fit.edu>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
Date: Tue, 19 Jan 2016 18:00:38 -0500 (EST)
From: worker0-0@mail0.company.com0
To: worker2-2@mail0.company.com0
Message-ID: <1338576172.94.1453244438408@worker0-0> Subject: Requirements
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii Content-Transfer-Encoding: 7bit

Words words words.
.  <——————————————————————————— indicates end of text
250 2.0.0 Ok: queued as 5BE21209C5
QUIT
221 2.0.0 Bye
```

Multipurpose Internet Mail Extensions (MIME)

Encodes binary data into the range of ASCII characters

- allows attatchments and images
- **doesn't** change text based nature of email

MIME consists of three components:

1. a set of header lines used to extend *RFC 822*

- `MIME-Version`: version of MIME being used
- `Content-Description`: what is in the message
- `Content-Type`: type of data
- `Content-Id`: a unique id for content
- `Content-Transfer-Encoding`: how data is encoded

2. definitions for *content types*

- MIME defines image types: **image/gif** and **image/jpeg** for both types

3. a way to *encode* various data types to include in an ASCII email message

MIME supported data types

Type	Subtype Examples
text	plain, html, xml, css
image	gif, jpeg, tiff
audio	basic, mpeg, mp3
video	mpeg, mp4, quicktime
model	vrml
application	octet-stream, pdf, javascipt, zip
message	http, rfc822
multipart	mixed, alternative, parallel, digest

Example MIME message:

```
From: Some One <someone@example.com>
MIME-Version: 1.0
Content-Type: multipart/mixed;
        boundary="XXXXboundary text"

This is a multipart message in MIME format.

--XXXXboundary text
Content-Type: text/plain

this is the body text

--XXXXboundary text
Content-Type: text/plain;
Content-Disposition: attachment;
        filename="test.txt"

this is the attachment text

--XXXXboundary text--
```

Purpose of MIME

- created for non-text data in text-only emails
- SMIME extends MIME, providing encryption and signing of documents

Application of MIME

- SMTP expects ASCII, so MIME encodes to BASE64 to look like ASCII
    - Encoding methos can be specified in MIME

SMTP (Simple Mail Transfer Protocol)

Moves email between servers (Transfer Agents) to deliver to a user's local email server

- Done over TCP, providing reliable delivery
- Not usually used to deliver directly to a user's computer, several other user agents do that
- A program running (as a daemon) on the email server handles the transfers, called *Message Transfer Agent* (MTA)

Compression

The purpose of compression is to reduce bandwidth for data transfer.

Applications

- Lossy: data is discarded to reduce the size, sometimes with less noticeable parts
  - algorithms will allow for a variable amount of detail to be lost
  - *e.g.*: jpeg
- Lossless: compressed data contains all original data
  - necessary for code, data, executables, and other sources that require original data
  - typically done with Lempel-Ziv algorithm
- applications are typically centered around video/audio/image transfer

Cryptography

Terminology

Term	Definition	Symbol
plaintext	original message (sequence of characters/bytes)	$P =< p_1, p_2, ..., p_n >$
ciphertext	encrypted version of plaintext	$C =< c_1, c_2, ..., c_n >$
encryption/decryption algorithms	steps to go between plaintext and ciphertext	$C = E(P)$, $P = D(C)$ | E → encryption, D → decryption
key	information or device used to encrypt/decrypt	K
symmetric encryption	the same key is used for encryption and decryption	$C = E(K, P)$, $P = D(K, C)$
assymetric encryption	different keys used for encryption and deceryption	$C = E(K_E, P)$, $P = D(K_D, C)$

Symmetric and Asymmetric Encryption

Symmetric

- Same key is used for encryptiona **and** decryption
- Advantages:
  - fast
  - keys are smaller
- Disadvantages:
  - less secure
  - key must be protected from access by others

Asymmetric

- Different keys used for encryption and decryption
- Advantages:
  - provides confidentiality and integrity
- Disadvantages
  - poor performance compared to symmetric's single key

How Encryption Works

Symmetric

Public key provides a solution. Sender S and receiver R both have public keys. KX is symmetric key we want to exchange.

1. S wants to send KX to R
2. S encrypts KX with $E(K_{S_{private}}, KX)$
3. S encrypte using R's public key → $E(K_{R_{public}}, E(K_{S_{private}}, KX))$
4. R uses $K_{R_{private}}$ to decrypt the outer layer, then uses $K_{S_{public}}$ to decrypt inner layer

- Note that only S could send KX, and only R could read it.

Single Sign On (SSO)

A process that allows a user to log into a central authority and access other sites and services for which they have credentials for.

Benefits

- Fewer username and password combinations for users to remember and manage, reducing risks to privacy and security
- More convenient for users who frequently access multiple machines/systems
- Supports centralized management of password compliance and reporting by IT

Cryptographic Hash Functions

A function that provides an apparently random output for the same input.

- provides security and diffusion, avoids collisions
  - Not every hash function is a *cryptographic* hash function, but every cryptographic hash function is a hash function

A good *cryptgraphic* hash function should not allow:

- finding a message with a given hash
- modification of a message without changing the hash
- finding two messages with the same hash

Hashes are used to verify integrity in Software Distribution

Secure Communications

Network Encryption

Link Encryption

- at lowest network layers
- encrypts Ethernet payload, but not the header
- encryption across direct links between nodes
- requires compatible hardware at each end
- key shared by network interfaces at nodes

Benefits

- Data is encrypted in transit
- Hardware does encryption/decryption

Limits

- Data is not encrypted on hosts, only in transit

End to End

- at higher network layers
- only data is encrypted, but routing is unaffected
- application encrypts, while the network transmits
- users/applications control keys

Benefits

- Each applications handles its own encryption/decryption
- Data is encrypted while in transit and while on hosts.

Limits

- Only applications that can encrypt will protect their data.
- Client and server must share a key.

VPN

- connects to a remote computer to a secure network by creating an encrypted tunnel
- remote host must be authenticated by the secure network
- protects data in transit to/from network
- applications on host send data through tunnel, but don't have to add encryption

Benefits

- VPN software and/or hardware handles encryption/decryption at endpoints
- applications don't have to provide encryption keys
- data is encrypted aloing entire path from client to server

Limits

- VPNs lamost always slow connection speed
- all traffic goes through the tunnel

SSH (Secure Shell)

The Secure Shell (SSH) protocol was developed to provide a secure remote login service by creating encrypted connections before transeferring any data or commands.

- SSH can be used to create a secure tunnel between hosts
- replaced by telnet and rlogin

SSL (Secure Sockets Layer)

SSL supports secure end-toend network connections for TCP applications

- first used in the Netscape browser in 1995
- updated over time to fix flaws and vulnerabilities
- SSL provides a layer between the application and the TCP layer, encrypting all data in the TCP packet using a shared session key
- the approach the protocol was based on was good, but it relied on obsolete or vulnerable components (e.g., MD5 and SHA-1)
- replaced by the Transport Layer Security (TLS) protocol in the 2000’s and deprecated in 2015

TLS (Transport Layer Security)

Widely deployed security protocol above transport layer

- supported by browsers, web servers, etc.

Provides:

- authentication established by public key certificates
- confidentiality through shared session key exchanged during configuration stage
- integrity through cryptographic hash values included with messages

Built on:

- handshake: Alice, Bob use public key certificated and share private keys to authenticate each other and exchange/create shared secret key
- key derivation: Alice, Bob use shared secret to derive session keys
- data transfer: stream data transfer, data is seen as series of records, not just one-time transactions
- connection closure: special messages are used to securely close the connection

TLS provides an Application Programming Interface (API) that any application can use (added to HTTP)

Network Security

Network Recon

Checks what app is running and if it is vulnerable

- port scans discover active applications
  - checks if the applications are vulnerable to an attack

OS/Application Fingerprinting

- looks for identifying characteristics
- find vulnerable OS or apps before attacking
- protocol specifications are interpreted by OS designers and each OS may respond slightly differently to requests
- using those differences, we can find out which OS is running

Details and specifications are found in published documentation

- bug reports, source code, RFCs

Firewalls

General Information

- isolate organizations internal network from internet, allowing some packets to pass and blocking others
- firewall sits between internal network and internet, examining packets and makes decisions about forwarding or dropping packets based on:
  - source or destination IP address
  - application (port #), protocol (TCP/UDP)
  - contents of one or more packets
  - known malicious traffic or behaviors
- firewall based network security depends on the firewall being the **only** path into the local network from outside
  - there should be no way to bypass the firewall via another gateway or wireless connection

Purpose

- prevent denial of service attacks:
  - SYN flooding: attacker establishes many bogus TCP connections, so no resources are left for 'real' connections
- prevent illegal modification/access of internal data
  - *e.g.*: attacker compromises company database and downloads/modifies private data
- allow only authorized access to the network
  - pre-determined authenticated users/hosts

Three Types of Firewalls

1. stateless packet filters

- firewall decides whether a packet should proceed
  - decision is made on per-packet basis
- uses packet header for inspection
- filtering rules based on binary pattern-matching

2. stateful packet filters

- based on context of connection
  - if it is a new connection, then check against security policy
  - if existing, look it up in state transition table and update table if necessary
    - *e.g*: only allow incoming traffic to high-numbered port if there is an established connection
- Challenges: UDP and ICMP
- default filter denies everything not explicitly permitted
- filtering rules (ACL) have same format as stateless filtering
  - filters can be bypassed with a VPN
- application gateways
  - filter packets on type of application data
    - also specific IP addresses or TCP, UDP ports
  - *e.g.*: allow select internal users to sftp to outside network

Limitations:

- IP spoofing, when a firewall doesn't know the data *really* comes from claimed source
- Doesn't prevent application-specific attacks
  - no payload inspection
  - *e.g.*: firewall will not block an attack string with URL buffer overflow
- cannot determine if:
  - user has been compromised and will continue to pass that user's traffic
  - a machine is being used to pass information in/out of the network for malicious purposes
    - outgoing connections are usually allowed

IP Packet Information

Header Fields

Field	Size (bits)	Description
Version	4	most often v4
HLen	4	number of 32-bit words in header
TOS	8	type of service (not widely used)
Length	16	number of bytes in datagram
Ident	16	support for fragmentation
Flags/Offset	16	^^^^^^^^^
TTL	8	number of hops this datagram has travelled
Protocol	8	demultiplex key (TCP=6, UDP=17)
Checksum	16	for header only
SrcAffr	32	source address
DestAddr	32	destination address
Options	variable	
Pad	variable	

IP Header Fields Explained

Version

all IPv4 packets have value `0x4` (`0100`)

What are the two active version of the Internet Protocol (IP)?

- IPv4 and IPv6

TTL

number of hops this datagram can pass through (prevents infinite loops)

What is the purpose?

- To prevent infinite loops by defining how many hops the datagram can pass through.

Source Address

How many bits in the IP header are allocated for this field?

- 32 bits

HLen

This field shows the number of *4-byte words* the header uses. Default is `0x5` (`0101`) because default header is 20 bytes.

> `0x5 * 4 bytes`

How large (in bytes) is the header of a packet when the value in this field is 0x6?

> `0x6 * 4 bytes = 24 bytes`

Checksum

What areas of an IP datagram are included when the checksum is calculated?

- only the header of the IP datagram

Length

What is the maximum length of an IP packet (in bytes)?

- minimum of 20, maximum of $2^16 = 65,535$ bytes.