Security

15-441

With slides from: Debabrata Dash, Nick Feamster, Vyas Sekar, and others

Our “Narrow” Focus

- Yes:
  - Protecting network resources and limiting connectivity (Part I)
  - Creating a “secure channel” for communication (Part II)

- No:
  - Preventing software vulnerabilities & malware, or “social engineering”.

Flashback .. Internet design goals

1. Interconnection
2. Failure resilience
3. Multiple types of service
4. Variety of networks
5. Management of resources
6. Cost-effective
7. Low entry-cost
8. Accountability for resources

Where is security?

Why did they leave it out?

- Designed for connectivity
- Network designed with implicit trust
  - No “bad” guys
- Can’t security be provided at the edge?
  - Encryption, Authentication etc
  - End-to-end arguments in system design
Security Vulnerabilities

- At every layer in the protocol stack!
- Network-layer attacks
  - IP-level vulnerabilities
  - Routing attacks
- Transport-layer attacks
  - TCP vulnerabilities
- Application-layer attacks

IP-level vulnerabilities

- IP addresses are provided by the source
  - Spoofing attacks
- Using IP address for authentication
  - e.g., login with .rhosts
- Some “features” that have been exploited
  - Fragmentation
  - Broadcast for traffic amplification

Security Flaws in IP

- The IP addresses are filled in by the originating host
  - Address spoofing
- Using source address for authentication
  - r-utilities (rlogin, rsh, rhosts etc..)

Smurf Attack

- Can A claim it is B to the server S?
  - ARP Spoofing
- Can C claim it is B to the server S?
  - Source Routing

Attacking System

Broadcast Enabled Network

Victim System
ICMP Attacks

- No authentication
- ICMP redirect message
  - Can cause the host to switch gateways
  - Benefit of doing this?
    - Man in the middle attack, sniffing
- ICMP destination unreachable
  - Can cause the host to drop connection
- ICMP echo request/reply
- Many more...

Routing attacks

- Divert traffic to malicious nodes
  - Black-hole
  - Eavesdropping
- How to implement routing attacks?
  - Distance-Vector:
    - Link-state:
  - BGP vulnerabilities

Routing attacks

- Divert traffic to malicious nodes
  - Black-hole
  - Eavesdropping
- How to implement routing attacks?
  - Distance-Vector:
    - Link-state:
  - BGP vulnerabilities

TCP-level attacks

- SYN-Floods
  - Implementations create state at servers before connection is fully established
- Session hijack
  - Pretend to be a trusted host
  - Sequence number guessing
- Session resets
  - Close a legitimate connection
**Session Hijack**

1. First send a legitimate SYN to server
2. Using ISN_S1 from earlier connection guess ISN_S2!

**TCP Layer Attacks**

- **TCP SYN Flooding**
  - Exploit state allocated at server after initial SYN packet
  - Send a SYN and don’t reply with ACK
  - Server will wait for 511 seconds for ACK
  - Finite queue size for incomplete connections (1024)
  - Once the queue is full it doesn’t accept requests

- **TCP Session Poisoning**
  - Send RST packet
    - Will tear down connection
  - Do you have to guess the exact sequence number?
    - Anywhere in window is fine
    - For 64k window it takes 64k packets to reset
    - About 15 seconds for a T1
An Example

- Finger @S
- showmount -e
- Send 20 SYN packets to S

Finger

- Attack when no one is around
- What other systems it trusts?
- Determine ISN behavior

Mitnick

- Send 20 SYN packets to S
- SYN flood T
- T won’t respond to packets

Mitnick

• Send SYN to S spoofing as T
• Send ACK to S with a guessed number
• S assumes that it has a session with T

Mitnick

• Send "echo ++ > ~/.rhosts"

Mitnick

15-411: security
Where do the problems come from?

- Protocol-level vulnerabilities
  - Implicit trust assumptions in design

- Implementation vulnerabilities
  - Both on routers and end-hosts

- Incomplete specifications
  - Often left to the imagination of programmers

Outline – Part I

- Security Vulnerabilities

  - **Denial of Service**

  - Worms

  - Countermeasures: Firewalls/IDS

---

Denial of Service

- Make a service unusable/unavailable

- Disrupt service by taking down hosts
  - E.g., ping-of-death

- Consume host-level resources
  - E.g., SYN-floods

- Consume network resources
  - E.g., UDP/ICMP floods

Simple DoS

- Attacker usually spoofs source address to hide origin

- Aside: Backscatter Analysis
  - When attack traffic results in replies from the victim
  - E.g. TCP SYN, ICMP ECHO

  ![Diagram](Attacker -> Lots of traffic -> Victim)
Backscatter Analysis

- Attacker sends spoofed TCP SYN packets to www.haplessvictim.com
  - With spoofed addresses chosen at random
- My network sees TCP SYN-ACKs from www.haplessvictim.com at rate R
- What is the rate of the attack?
  - Assuming addresses chosen are uniform
  - \((2^{32}/ \text{Network Address space}) \times R\)

Reflector Attack

Unsolicited traffic at victim from legitimate hosts

Distributed DoS

- Handlers are usually high volume servers
  - Easy to hide the attack packets
- Agents are usually home users with DSL/Cable
  - Already infected and the agent installed
- Very difficult to track down the attacker
  - Multiple levels of indirection!
- Aside: How to distinguish DDos from flash crowd?
Outline – Part I

- Security, Vulnerabilities
- Denial of Service
- Worms
- Countermeasures: Firewalls/IDS

Worm Overview

- Self-propagate through network
- Typical Steps in worm propagation
  - Probe host for vulnerable software
  - Exploit the vulnerability (e.g., buffer overflow)
    - Attacker gains privileges of the vulnerable program
  - Launch copy on compromised host
- Spread at exponential rate
  - 10M hosts in < 5 minutes
  - Hard to deal with manual intervention

Scanning Techniques

- Random
- Local subnet
- Routing Worm
- Hitlist
- Topological

Random Scanning

- 32-bit randomly generated IP address
  - E.g., Slammer and Code Red I
  - What about IPv6?
- Hits black-holed IP space frequently
  - Only 28.6% of IP space is allocated
  - Detect worms by monitoring unused addresses
    - Honeypots/Honeynet
**Subnet Scanning**

- Generate last 1, 2, or 3 bytes of IP address randomly
- Code Red II and Blaster
- Some scans must be completely random to infect whole internet

**Routing Worm**

- BGP information can tell which IP address blocks are allocated
- This information is publicly available
  - http://www.routeviews.org/
  - http://www.ripe.net/ris/

**Hit List**

- List of vulnerable hosts sent with payload
  - Determined before worm launch by scanning
- Boosts worm growth in the slow start phase
- Can evade common detection techniques

**Topological**

- Uses info on the infected host to find the next target
  - Morris Worm used /etc/hosts, .rhosts
  - Email address books
  - P2P software usually store info about peers that each host connects to
Some proposals for countermeasures

- Better software safeguards
  - Static analysis and array bounds checking (lint/e-fence)
  - Safe versions of library calls
    - `gets(buf) -> fgets(buf, size, ...)`
    - `sprintf(buf, ...) -> snprintf(buf, size, ...)`
- Host-diversity
  - Avoid same exploit on multiple machines
- Network-level: IP address space randomization
- Host-level solutions
  - E.g., Memory randomization, Stack guard
- Rate-limiting: Contain the rate of spread
- Content-based filtering: signatures in packet payloads

Outline – Part I

- Security, Vulnerabilities
- Denial of Service
- Worms
- Countermeasures: Firewalls/IDS

Countermeasure Overview

- High level basic approaches
  - Prevention
  - Detection
  - Resilience
- Requirements
  - Security: soundness / completeness (false positive / negative)
  - Overhead
  - Usability

Design questions ..

- Why is it so easy to send unwanted traffic?
  - Worm, DDoS, virus, spam, phishing etc
- Where to place functionality for stopping unwanted traffic?
  - Edge vs. Core
  - Routers vs. Middleboxes
- Redesign Internet architecture to detect and prevent unwanted traffic?
Firewalls

- Block/filter/modify traffic at network-level
  - Limit access to the network
  - Installed at perimeter of the network

- Why network-level?
  - Vulnerabilities on many hosts in network
  - Users don’t keep systems up to date
  - Lots of patches to keep track of
  - Zero-day exploits

Firewalls (cont’d…)

- Firewall inspects traffic through it
- Allows traffic specified in the policy
- Drops everything else
- Two Types
  - Packet Filters, Proxies

Packet Filters

- Selectively passes packets from one network interface to another

- Usually done within a router between external and internal network

- What/How to filter?
  - Packet Header Fields
    - IP source and destination addresses
    - Application port numbers
    - ICMP message types/Protocol options etc.
  - Packet contents (payloads)

Packet Filters: Possible Actions

- Allow the packet to go through
- Drop the packet (Notify Sender/Drop Silently)
- Alter the packet (NAT?)
- Log information about the packet
Some examples

- Block all packets from outside except for SMTP servers
- Block all traffic to/from a list of domains
- Ingress filtering
  - Drop pkt from outside with addresses inside the network
- Egress filtering
  - Drop pkt from inside with addresses outside the network

Typical Firewall Configuration

- Internal hosts can access DMZ and Internet
- External hosts can access DMZ only, not Intranet
- DMZ hosts can access Internet only
- Advantages?
  - If a service gets compromised in DMZ it cannot affect internal hosts

Firewall implementation

- Stateless packet filtering firewall

- Rule → (Condition, Action)

- Rules are processed in top-down order
  - If a condition satisfied – action is taken

Sample Firewall Rule

Allow SSH from external hosts to internal hosts

Two rules
- Inbound and outbound

How to know a packet is for SSH?
- Inbound: src-port>1023, dst-port=22
- Outbound: src-port=22, dst-port>1023
- Protocol=TCP

Ack Set?

<table>
<thead>
<tr>
<th>Rule</th>
<th>Dir</th>
<th>Src Addr</th>
<th>Src Port</th>
<th>Dst Addr</th>
<th>Dst Port</th>
<th>Proto</th>
<th>Ack Set?</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH-1</td>
<td>In</td>
<td>Ext</td>
<td>&gt; 1023</td>
<td>Int</td>
<td>22</td>
<td>TCP</td>
<td>Any</td>
<td>Allow</td>
</tr>
<tr>
<td>SSH-2</td>
<td>Out</td>
<td>Int</td>
<td>22</td>
<td>Ext</td>
<td>&gt; 1023</td>
<td>TCP</td>
<td>Yes</td>
<td>Allow</td>
</tr>
</tbody>
</table>
Default Firewall Rules

- Egress Filtering
  - Outbound traffic from external address → Drop
  - Benefits?
- Ingress Filtering
  - Inbound Traffic from internal address → Drop
  - Benefits?
- Default Deny
  - Why?

<table>
<thead>
<tr>
<th>Rule</th>
<th>Dir</th>
<th>Src Addr</th>
<th>Src Port</th>
<th>Dst Addr</th>
<th>Dst Port</th>
<th>Proto</th>
<th>Ack Set?</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Egress</td>
<td>Out</td>
<td>Ext</td>
<td>Any</td>
<td>Ext</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>Deny</td>
</tr>
<tr>
<td>Ingress</td>
<td>In</td>
<td>Int</td>
<td>Any</td>
<td>Int</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>Deny</td>
</tr>
<tr>
<td>Default</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>Deny</td>
</tr>
</tbody>
</table>

Packet Filters

- Advantages
  - Transparent to application/user
  - Simple packet filters can be efficient

- Disadvantages
  - Usually fail open
  - Very hard to configure the rules
  - May only have coarse-grained information?
    - Does port 22 always mean SSH?
    - Who is the user accessing the SSH?

Alternatives

- Stateful packet filters
  - Keep the connection states
  - Easier to specify rules
  - Problems?
    - State explosion
    - State for UDP/ICMP?
- Proxy Firewalls
  - Two connections instead of one
  - Either at transport level
    - SOCKS proxy
  - Or at application level
    - HTTP proxy

Proxy Firewall

- Data Available
  - Application level information
  - User information

- Advantages?
  - Better policy enforcement
  - Better logging
  - Fail closed

- Disadvantages?
  - Doesn’t perform as well
  - One proxy for each application
  - Client modification
Intrusion Detection Systems

- Firewalls allow traffic only to legitimate hosts and services
- Traffic to the legitimate hosts/services can have attacks
- Solution?
  - Intrusion Detection Systems
  - Monitor data and behavior
  - Report when identify attacks

Classes of IDS

- What type of analysis?
  - Signature-based
  - Anomaly-based
- Where is it operating?
  - Network-based
  - Host-based

Signature-based IDS

- Characteristics
  - Uses known pattern matching to signify attack
- Advantages?
  - Widely available
  - Fairly fast
  - Easy to implement
  - Easy to update
- Disadvantages?
  - Cannot detect attacks for which it has no signature

Anomaly-based IDS

- Characteristics
  - Uses statistical model or machine learning engine to characterize normal usage behaviors
  - Recognizes departures from normal as potential intrusions
- Advantages?
  - Can detect attempts to exploit new and unforeseen vulnerabilities
  - Can recognize authorized usage that falls outside the normal pattern
- Disadvantages?
  - Generally slower, more resource intensive compared to signature-based IDS
  - Greater complexity, difficult to configure
  - Higher percentages of false alerts
Network-based IDS

- Characteristics
  - NIDS examine raw packets in the network passively and triggers alerts

- Advantages?
  - Easy deployment
  - Unobtrusive
  - Difficult to evade if done at low level of network operation

- Disadvantages?
  - Fail Open
  - Different hosts process packets differently
  - NIDS needs to create traffic seen at the end host
  - Need to have the complete network topology and complete host behavior

Host-based IDS

- Characteristics
  - Runs on single host
  - Can analyze audit-trails, logs, integrity of files and directories, etc.

- Advantages
  - More accurate than NIDS
  - Less volume of traffic so less overhead

- Disadvantages
  - Deployment is expensive
  - What happens when host get compromised?

Summary – Part I

- Security vulnerabilities are real!
  - Protocol or implementation or bad specs
  - Poor programming practices
  - At all layers in protocol stack

- DoS/DDoS
  - Resource utilization attacks

- Worm/Malware
  - Exploit vulnerable services
  - Exponential spread

- Countermeasures: Firewall/IDS

Our “Narrow” Focus

- Yes:
  - Protecting network resources and limiting connectivity (Part I)
  - Creating a “secure channel” for communication (Part II)

- No:
  - Preventing software vulnerabilities & malware, or “social engineering”
Internet Design Decisions and Security

- Origin as a small and cooperative network
  (=> largely trusted infrastructure)
- Global Addressing
  (=> every sociopath is your next-door neighbor*)
- Connection-less datagram service
  (=> can’t verify source, hard to protect bandwidth)

* Dan Geer

Secure Communication with an Untrusted Infrastructure

- Anyone can connect
  (=> ANYONE can connect)
- Millions of hosts run nearly identical software
  (=> single exploit can create epidemic)
- Most Internet users know about as much as Senator Stevens aka “the tubes guy”
  (=> God help us all…)

6215-411: security
Secure Communication with an Untrusted Infrastructure

Alice

Hello, I'm “Bob”

What do we need for a secure communication channel?

- Authentication (Who am I talking to?)
- Confidentiality (Is my data hidden?)
- Integrity (Has my data been modified?)
- Availability (Can I reach the destination?)

What is cryptography?

"cryptography is about communication in the presence of adversaries."
- Ron Rivest

“cryptography is using math and other crazy tricks to approximate magic”
- Unknown 441 TA

Tools to help us build secure communication channels that provide:

1) Authentication
2) Integrity
3) Confidentiality
Cryptography As a Tool

• Using cryptography securely is not simple
• Designing cryptographic schemes correctly is near impossible.

Today we want to give you an idea of what can be done with cryptography. Take a security course if you think you may use it in the future.

The Great Divide

<table>
<thead>
<tr>
<th>Symmetric Crypto (Private key)</th>
<th>Asymmetric Crypto (Public key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(E.g., AES)</td>
<td>(E.g., RSA)</td>
</tr>
</tbody>
</table>

- Shared secret between parties? Yes No
- Speed of crypto operations Fast Slow

Symmetric Key: Confidentiality

Motivating Example:
You and a friend share a key \( K \) of \( L \) random bits, and want to secretly share message \( M \) also \( L \) bits long.

Scheme:
You send her the \( \text{xor}(M,K) \) and then she “decrypts” using \( \text{xor}(M,K) \) again.

1) Do you get the right message to your friend?
2) Can an adversary recover the message \( M \)?
3) Can adversary recover the key \( K \)?

Symmetric Key: Confidentiality

• One-time Pad (OTP) is secure but usually impractical
  - Key is as long as the message
  - Keys cannot be reused (why?)

In practice, two types of ciphers are used that require constant length keys:

Stream Ciphers: Ex: RC4, A5
Block Ciphers: Ex: DES, AES, Blowfish
Symmetric Key: Confidentiality

- **Stream Ciphers (ex: RC4)**

Alice: $K_{A-B}$ → PRNG → Pseudo-Random stream of $L$ bits → XOR → Message of Length $L$ bits → Encrypted Ciphertext

Bob uses $K_{A-B}$ as PRNG seed, and XORs encrypted text to get the message back (just like OTP).

Symmetric Key: Confidentiality

- **Block Ciphers (ex: AES)**

Block 1 → Block 2 → Block 3 → Block 4

(fixed block size, e.g. 128 bits)

Bob breaks the ciphertext into blocks, feeds it through the decryption engine using $K_{A-B}$ to recover the message.

Cryptographic Hash Functions

- Consistent
  hash(X) always yields same result
- One-way
  given Y, can’t find X s.t. hash(X) = Y
- Collision resistant
  given hash(W) = Z, can’t find X such that hash(X) = Z

Message of arbitrary length → Hash Fn → Fixed Size Hash

Symmetric Key: Integrity

- **Hash Message Authentication Code (HMAC)**

Step #1:
Alice creates MAC

Step #2
Alice Transmits Message & MAC

Step #3
Bob computes MAC with message and $K_{A-B}$ to verify.

Why is this secure? How do properties of a hash function help us?
Symmetric Key: Authentication

- You already know how to do this!
  (hint: think about how we showed integrity)

Alice receives the hash, computes a hash with \( K_{A-B} \), and she knows the sender is Bob.

whoops!

Symmetric Key: Authentication

- A “Nonce”
  - A random bitstring used only once. Alice sends nonce to Bob as a “challenge”. Bob replies with “fresh” MAC result.

Alice sends nonce to Bob as a “challenge”. Bob replies with “fresh” MAC result.

whoops!

Symmetric Key: Authentication

- A “Nonce”
  - A random bitstring used only once. Alice sends nonce to Bob as a “challenge”. Bob replies with “fresh” MAC result.

Alice sends Mallory a nonce, she cannot compute the corresponding MAC without \( K_{A-B} \).
Symmetric Key Crypto Review

- Confidentiality: Stream & Block Ciphers
- Integrity: HMAC
- Authentication: HMAC and Nonce

Questions??
Are we done? Not Really:
1) Number of keys scales as $O(n^2)$
2) How to securely share keys in the first place?

Diffie-Hellman key exchange

- An early (1976) way to create a shared secret.
- Everyone knows a prime, $p$, and a generator, $g$.
- Alice and Bob want to share a secret, but only have internet to communicate over.

DH key exchange

Everyone: large prime $p$ and generator $g$

Create secret: $a$

Send Bob: $g^a \mod p$

Create secret: $b$

Send Alice: $g^b \mod p$

Alice

Bob

Compute: $(g^b \mod p)^a$

Compute: $(g^a \mod p)^b$

Voila: They both know $g^{ab}$ which is secret!

DH key exchange & Man-In-The-Middle

- $g^a \mod p$
- $g^b \mod p$
- $g^c \mod p$

Asymmetric Key Crypto:

- Instead of shared keys, each person has a “key pair”
  - $K_B$ Bob’s public key
  - $K_B^{-1}$ Bob’s private key

- The keys are inverses, so: $K_B^{-1}(K_B(m)) = m$

Asymmetric Key Crypto:

- It is believed to be computationally unfeasible to derive $K_B^{-1}$ from $K_B$ or to find any way to get $M$ from $K_B(M)$ other than using $K_B^{-1}$.

  $\Rightarrow$ $K_B$ can safely be made public.

  Note: We will not explain the computation that $K_B(m)$ entails, but rather treat these functions as black boxes with the desired properties.

Asymmetric Key: Confidentiality

- If we are given a message $M$, and a value $S$ such that $K_B(S) = M$, what can we conclude?

  - The message must be from Bob, because it must be the case that $S = K_B^{-1}(M)$, and only Bob has $K_B^{-1}$!

  - This gives us two primitives:
    - $\text{Sign} (M) = K_B^{-1}(M) = \text{Signature } S$
    - $\text{Verify} (S, M) = \text{test}(K_B(S) == M)$
Asymmetric Key: Integrity & Authentication

• We can use Sign() and Verify() in a similar manner as our HMAC in symmetric schemes.

![Diagram of Integrity: S = Sign(M)](message)

Receiver must only check Verify(M, S)

![Diagram of Authentication: Nonce](message)

S = Sign(Nonce)

One last “little detail”…

How do I get these keys in the first place??
Remember:

• Symmetric key primitives assumed Alice and Bob had already shared a key.
• Asymmetric key primitives assumed Alice knew Bob’s public key.

This may work with friends, but when was the last time you saw Amazon.com walking down the street?

Asymmetric Key Review:

• Confidentiality: Encrypt with Public Key of Receiver
• Integrity: Sign message with private key of the sender
• Authentication: Entity being authenticated signs a nonce with private key, signature is then verified with the public key

But, these operations are computationally expensive*

Symmetric Key Distribution

• How does Andrew do this?

Andrew Uses Kerberos, which relies on a Key Distribution Center (KDC) to establish shared symmetric keys.
**Key Distribution Center (KDC)**

- Alice, Bob need shared symmetric key.
- **KDC**: server shares different secret key with each registered user (many users)
- Alice, Bob know own symmetric keys, $K_{A,KDC} - K_{B,KDC}$, for communicating with KDC.

**Q:** How does KDC allow Bob, Alice to determine shared symmetric secret key to communicate with each other?

**KDC** generates $R_1$  

KDC  

Alice knows $R_1$  

Bob knows to use $R_1$ to communicate with Alice  

Alice and Bob communicate: using $R_1$ as session key for shared symmetric encryption

**How Useful is a KDC?**

- Must always be online to support secure communication
- KDC can expose our session keys to others!
- Centralized trust and point of failure.

In practice, the KDC model is mostly used within single organizations (e.g. Kerberos) but not more widely.

**Certification Authorities**

- **Certification authority (CA)**: binds public key to particular entity, E.
- An entity E registers its public key with CA.  
  - E provides “proof of identity” to CA.  
  - CA creates certificate binding E to its public key.  
  - Certificate contains E’s public key AND the CA’s signature of E’s public key.

Bob's public key  

CA generates $S = \text{Sign}(K_{CA})$  

certificate = Bob’s public key and signature by CA
Certification Authorities

- When Alice wants Bob's public key:
  - Gets Bob’s certificate (Bob or elsewhere).
  - Use CA’s public key to verify the signature within Bob’s certificate, then accepts public key.

\[ \text{Verify}(S, K_B) \]

If signature is valid, use \( K_B \)

\[ \text{CA public key} \]

\[ K_{CA} \]

Certificate Contents

- info algorithm and key value itself (not shown)

- Cert owner
- Cert issuer
- Valid dates
- Fingerprint of signature

Which Authority Should You Trust?

- Today: many authorities
- What about a shared Public Key Infrastructure (PKI)?
  - A system in which “roots of trust” authoritatively bind public keys to real-world identities
  - So far it has not been very successful

Transport Layer Security (TLS) aka Secure Socket Layer (SSL)

- Used for protocols like HTTPS
- Special TLS socket layer between application and TCP (small changes to application).
- Handles confidentiality, integrity, and authentication.
- Uses “hybrid” cryptography.
Setup Channel with TLS “Handshake”

Handshake Steps:
1) Client and server negotiate exact cryptographic protocols
2) Client validates public key certificate with CA public key.
3) Client encrypts secret random value with server’s key, and sends it as a challenge.
4) Server decrypts, proving it has the corresponding private key.
5) This value is used to derive symmetric session keys for encryption & MACs.

How TLS Handles Data

1) Data arrives as a stream from the application via the TLS Socket
2) The data is segmented by TLS into chunks
3) A session key is used to encrypt and MAC each chunk to form a TLS “record”, which includes a short header and data that is encrypted, as well as a MAC.
4) Records form a byte stream that is fed to a TCP socket for transmission.

Summary – Part II

• Internet design and growth => security challenges
• Symmetric (pre-shared key, fast) and asymmetric (key pairs, slow) primitives provide:
  ▪ Confidentiality
  ▪ Integrity
  ▪ Authentication
• “Hybrid Encryption” leverages strengths of both.
• Great complexity exists in securely acquiring keys.
• Crypto is hard to get right, so use tools from others, don’t design your own (e.g. TLS).

Resources

• Textbook: 8.1 – 8.3
• Wikipedia for overview of Symmetric/Asymmetric primitives and Hash functions.
• OpenSSL (www.openssl.org): top-rate open source code for SSL and primitive functions.
• “Handbook of Applied Cryptography” available free online: www.cacr.math.uwaterloo.ca/hac/