CSci 4271W
Development of Secure Software Systems
Day 23: Networking and security
Stephen McCamant
University of Minnesota, Computer Science & Engineering

Outline
- Brief introduction to networking
- Announcements intermission
- Some classic network attacks
- The web from a security perspective
- SQL injection

The Internet
- A bunch of computer networks voluntarily interconnected
- Capitalized because there’s really only one
- No centralized network-level management
  - But technical collaboration, DNS, etc.

Layered model (OSI)
- 7. Application (HTTP)
- 6. Presentation (MIME?)
- 5. Session (SSL?)
- 4. Transport (TCP)
- 3. Network (IP)
- 2. Data-link (PPP)
- 1. Physical (10BASE-T)

Layered model: TCP/IP
- Application protocol (e.g., HTTP) → Application
- TCP or UDP → Transport
- IP → Network
- 802.11 (Wi-Fi) → Link

IP(v4) addressing
- Interfaces (hosts or routers) identified by 32-bit addresses
  - Written as four decimal bytes, e.g., 192.168.10.2
- First k bits identify network, 32 – k host within network
  - Can’t (anymore) tell k from the bits
- We’ll run out any year now

Packet wrapping
- application data → segments → TCP data → IP data → packet
- frames → IP data → TCP data

IP and ICMP
- Internet Protocol (IP) forwards individual packets
- Packets have source and destination addresses, other options
- Automatic fragmentation (usually avoided)
- ICMP (I Control Message P) adds errors, ping packets, etc.
UDP

User Datagram Protocol: thin wrapper around IP
- Adds source and destination port numbers (each 16-bit)
- Still connectionless, unreliable
- OK for some small messages

TCP

Transmission Control Protocol: provides reliable bidirectional stream abstraction
- Packets have sequence numbers, acknowledged in order
- Missed packets resent later

Flow and congestion control

- Flow control: match speed to slowest link
  - "Window" limits number of packets sent but not ACKed
- Congestion control: avoid traffic jams
  - Lost packets signal congestion
  - Additive increase, multiplicative decrease of rate

Routing

- Where do I send this packet next?
  - Table from address ranges to next hops
  - Core Internet routers need big tables
- Maintained by complex, insecure, cooperative protocols
  - Internet-level algorithm: BGP (Border Gateway Protocol)

Below IP: ARP

- Address Resolution Protocol maps IP addresses to lower-level address
  - E.g., 48-bit Ethernet MAC address
- Based on local-network broadcast packets
- Complex Ethernets also need their own routing (but called switches)

DNS

- Domain Name System: map more memorable and stable string names to IP addresses
- Hierarchically administered namespace
  - Like Unix paths, but backwards
- .edu server delegates to .umn.edu server, etc.

DNS caching and reverse DNS

- To be practical, DNS requires caching
  - Of positive and negative results
- But, cache lifetime limited for freshness
- Also, reverse IP to name mapping
  - Based on special top-level domain, IP address written backwards

Classic application: remote login

- Killer app of early Internet: access supercomputers at another university
- Telnet: works cross-OS
  - Send character stream, run regular login program
- rlogin: BSD Unix
  - Can authenticate based on trusting computer connection comes from
  - (Also rsh, rcp)
Outline
Brief introduction to networking
Announcements intermission
Some classic network attacks
The web from a security perspective
SQL injection

Last day for Wheeler reading quiz

If you were putting this quiz off until the last day, that's today

Outline
Brief introduction to networking
Announcements intermission
Some classic network attacks
The web from a security perspective
SQL injection

Packet sniffing

Watch other people's traffic as it goes by on network
Easiest on:
- Old-style broadcast (thin, "hub") Ethernet
- Wireless
- Or if you own the router

Forging packet sources

Source IP address not involved in routing, often not checked
Change it to something else!
Might already be enough to fool a naive UDP protocol

TCP spoofing

Forging source address only lets you talk, not listen
Old attack: wait until connection established, then DoS one participant and send packets in their place
Frustrated by making TCP initial sequence numbers unpredictable
Fancier attacks modern attacks are "off-path"

ARP spoofing

Impersonate other hosts on local network level
Typical ARP implementations stateless, don’t mind changes
Now you get victim's traffic, can read, modify, resend

rlogin and reverse DNS

rlogin uses reverse DNS to see if originating host is on whitelist
How can you attack this mechanism with an honest source IP address?
**rlogin and reverse DNS**
- rlogin uses reverse DNS to see if originating host is on whitelist
- How can you attack this mechanism with an honest source IP address?
- Remember, ownership of reverse-DNS is by IP address

**Outline**
- Brief introduction to networking
- Announcements intermission
- Some classic network attacks
- The web from a security perspective
- SQL injection

**Once upon a time: the static web**
- HTTP: stateless file download protocol
  - TCP, usually using port 80
- HTML: markup language for text with formatting and links
- All pages public, so no need for authentication or encryption

**Web applications**
- The modern web depends heavily on active software
- Static pages have ads, paywalls, or “Edit” buttons
- Many web sites are primarily forms or storefronts
- Web hosted versions of desktop apps like word processing

**Server programs**
- Could be anything that outputs HTML
- In practice, heavy use of databases and frameworks
- Wide variety of commercial, open-source, and custom-written
- Flexible scripting languages for ease of development
  - PHP, Ruby, Perl, etc.

**Client-side programming**
- Java: nice language, mostly moved to other uses
- ActiveX: Windows-only binaries, no sandboxing
  - Glad to see it on the way out
- Flash and Silverlight: last important use was DRM-ed video
- Core language: JavaScript

**JavaScript and the DOM**
- JavaScript (JS) is a dynamically-typed prototype-OO language
  - No real similarity with Java
- Document Object Model (DOM): lets JS interact with pages and the browser
- Extensive security checks for untrusted-code model

**Same-origin policy**
- Origin is a tuple (scheme, host, port)
  - E.g., (http, www.umn.edu, 80)
- Basic JS rule: interaction is allowed only with the same origin
- Different sites are (mostly) isolated applications
GET, POST, and cookies

- GET request loads a URL, may have parameters delimited with ?, &,
- Standard: should not have side-effects
- POST request originally for forms
- Can be larger, more hidden, have side-effects
- Cookie: small token chosen by server, sent back on subsequent requests to same domain

User and attack models

- "Web attacker" owns their own site (www.attacker.com)
- And users sometimes visit it
- Realistic reasons: ads, SEO
- "Network attacker" can view and sniff unencrypted data
- Unprotected coffee shop WiFi

Outline

- Brief introduction to networking
- Announcements intermission
- Some classic network attacks
- The web from a security perspective
- SQL injection

Relational model and SQL

- Relational databases have tables with rows and single-typed columns
- Used in web sites (and elsewhere) to provide scalable persistent storage
- Allow complex queries in a declarative language SQL

Example SQL queries

- SELECT name, grade FROM Students WHERE grade < 60 ORDER BY name;
- UPDATE Votes SET count = count + 1 WHERE candidate = 'John';

Template: injection attacks

- Your program interacts with an interpreted language
- Untrusted data can be passed to the interpreter
- Attack data can break parsing assumptions and execute arbitrary commands

SQL + injection

- Why is this named most critical web app. risk?
- Easy mistake to make systematically
- Can be easy to exploit
- Database often has high-impact contents
  - Eg, logins or credit cards on commerce site

Strings do not respect syntax

- Key problem: assembling commands as strings
- "WHERE name = '$name';"
- Looks like $name is a string
- Try $name = "me’ OR grade > 80; --"
Using tautologies

- Tautology: formula that's always true
- Often convenient for attacker to see a whole table
- Classic: OR 1=1

Non-string interfaces

- Best fix: avoid constructing queries as strings
- SQL mechanism: prepared statement
  - Original motivation was performance
- Web languages/frameworks often provide other syntax

Retain functionality: escape

- Sanitizing data is transforming it to prevent an attack
- Escaped data is encoded to match language rules for literal
  - E.g., " and \n in C
- But many pitfalls for the unwary:
  - Differences in escape syntax between servers
  - Must use right escape for context: not everything's a string

Lazy sanitization: allow-listing

- Allow only things you know to be safe/intended
- Error or delete anything else
- Short allow-list is easy and relatively easy to secure
  - E.g., digits only for non-negative integer
- But, tends to break benign functionality

Poor idea: deny-listing

- Space of possible attacks is endless, don't try to think of them all
- Want to guess how many more comment formats SQL has?
- Particularly silly: deny 1=1

Attacking without the program

- Often web attacks don't get to see the program
  - Not even binary, it's on the server
- Surmountable obstacle:
  - Guess natural names for columns
  - Harvest information from error messages

Blind SQL injection

- Attacking with almost no feedback
- Common: only "error" or "no error"
- One bit channel you can make yourself: if (x) delay 10 seconds
- Trick to remember: go one character at a time