CIS 433/533 - Computer and Network Security
Worms, DDoS, Web

Professor Kevin Butler
Winter 2011
Worms

- A worm is a self-propagating program.
- As relevant to this discussion
  1. Exploits some vulnerability on a target host …
  2. (often) embeds itself into a host …
  3. Searches for other vulnerable hosts …
  4. Goto (1)
The Danger

- What makes worms so dangerous is that infection grows at an exponential rate

  - A simple model:
    - \( s \) (search) is the time it takes to find vulnerable host
    - \( i \) (infect) is the time is take to infect a host

  - Assume that \( t=0 \) is the *worm outbreak*, the number of hosts at \( t=j \) is

\[
2^{(j/(s+i))}
\]

- For example, if \( s+i = 1 \), what # infected at time \( t=32 \)?
The result
The Morris Worm

- Robert Morris, doctoral student at Cornell (now MIT prof)
  - Wrote a small (99 line) program on Nov. 3, 1988
  - Disabled the Internet

- Worm operation
  - Reads /etc/password, they tries the obvious choices and dictionary, /usr/dict words
  - Used local /etc/hosts.equiv, .rhosts, .forward to identify hosts that are related
    - Tries cracked passwords at related hosts (if necessary)
    - Uses whatever services are available to compromise other hosts
  - Scanned local interfaces for network information
  - Covered its tracks (set is own process name to sh, prevented accurate cores, re-forked itself)
Code Red

- Anatomy of a worm: Maiffret (good reading)
- Exploited a Microsoft IIS web-server vulnerability
  - A vanilla buffer overflow (allows adversary to run code)
  - Scans for vulnerabilities over random IP addresses
  - Sometimes would deface the served website
- July 16th, 2001 - outbreak
  - CRv1 - contained bad randomness (fixed IPs searched)
  - CRv2 - fixed the randomness,
    - added DDOS of www.whitehouse.gov
    - Turned itself off and on (on 1st and 16th of month)
- August 4 - Code Red II
  - Different code base, same exploit
  - Added local scanning (biased randomness to local IPs)
  - Killed itself in October of 2001
Worms and infection

- The effectiveness of a worm is determined by how good it is at identifying vulnerable machines
  - Morris used local information at the host
  - Code Red used what?
- Multi-vector worms use lots of ways to infect
  - E.g., network, DFS partitions, email, drive by downloads …
  - Another worm, Nimda did this
- Lots of scanning strategies
  - **Signpost scanning** (using local information, e.g., Morris)
  - **Random IP** - good, but waste a lot of time scanning dark or unreachable addresses (e.g., Code Red)
  - **Local scanning** - biased randomness
  - **Permutation scanning** - instance is given part of IP space
Other scanning strategies

- The doomsday worm: a flash worm
  - Create a hit list of all vulnerable hosts
    - Staniford et al. argue this is feasible
    - Would contain a 48MB list
  - Do the infect and split approach
  - Use a zero-day vulnerability

- Result: saturate the Internet is less than 30 seconds!
Worms: Defense Strategies

- (Auto) patch your systems: most, if not all, large worm outbreaks have exploited known vulnerabilities (with patches)

- Heterogeneity: use more than one vendor for your networks

- Filtering: look for unnecessary or unusual communication patterns, then drop them on the floor
  - Shield (Wang et al., MSR): provides filtering for known vulnerabilities, such that they are protected immediately (analog to virus scanning)
  - SWORD (Li et al., UOregon): behavior-based worm detection (causal relationships, continuity)

- Auto-generation of worm signatures
  - Earlybird (UCSD), vulnerability-based sigs (CMU)
Denial of Service

- Intentional prevention of access to valued resource
  - CPU, memory, disk (system resources)
  - DNS, print queues, NIS (services)
  - Web server, database, media server (applications)

- This is an attack on availability (fidelity)

- Note: launching DOS attacks is easy

- Note: preventing DOS attacks is hard
  - Mitigation the path most frequently traveled
Request Flood

- Canonical DoS Attack: request flooding
  - Overwhelm some resource with legitimate requests
  - e.g., web server, phone system

- Note: unintentional flood is called a *flash crowd*
Example: SMURF Attacks

- This is one of the deadliest and simplest of the DOS attacks (called a *naturally amplified* attack)
  - Send a large number PING packet networks on the broadcast IP addresses (e.g., 192.168.27.254)
  - Set the source packet IP address to be your victim
  - All hosts will reflexively respond to the ping at your victim
  - … and it will be crushed under the load.
- Fraggle: UDP based SMURF
Distributed denial of service

- DDoS: Network oriented attacks aimed at preventing access to network, host or service
  - Saturate the target’s network with traffic
  - Consume all network resources (e.g., SYN)
  - Overload a service with requests
    - Use “expensive” requests (e.g., “sign this data”)
  - Can be extremely costly (e.g., Amazon)
- Result: service/host/network is unavailable
- Frequently distributed via other attack
- *Note*: IP is often hidden (spoofed)
DDoS

- Send a stream of packets/requests/whatever …
  - many PINGS, HTML requests, ...
- Send a few malformed packets
  - causing failures or expensive error handling
  - low-rate packet dropping (TCP congestion control)
  - “ping of death”
- Abuse legitimate access
  - Compromise service/host
  - Use its legitimate access rights to consume the rights for domain (e.g., local network)
  - E.g., First-year graduate student runs a recursive file operation on root of NFS partition
Adversary Network

(adversary) -> (masters) -> (zombies) -> (target)
Why DDoS

• What would motivate a DDoS?
  ‣ An axe to grind …
  ‣ Curiosity (script kiddies) …
  ‣ Blackmail
  ‣ Information warfare …

• Internet is an open system …
  ‣ Packets not authenticated, probably can’t be
    ‣ Would not solve the problem just move it (firewall)
  ‣ Too many end-points can be remote controlled
Why is DDoS possible?

- Interdependence - services dependent on each other
  - E.g., Web depends on TCP and DNS, which depends on routing and congestion control, …

- Limited resources (or rather resource imbalances)
  - Many times it takes few resources on the client side to consume lots of resources on the server side
  - E.g., SYN packets consume lots of internal resources
  - Difference in expected usage and design principles (e.g., hooking the mobile phone network up to the Internet)
DDoS and E2E argument

- E2E (a simplified version): We should design the network such that all the intelligence is at the edges.
  - So that the network can be more robust and scalable
  - Many think is the main reason why the Internet works

- Downside:
  - Also, no real ability to police the traffic/content
  - So, many security solutions break this E2E by cracking open packets (e.g., application level firewalls)
  - DDoS is real because of this …
Simple DDOS Mitigation

- Ingress/Egress Filtering
  - Helps spoofed sources, not much else
- Better Security
  - Limit availability of zombies, not feasible
  - Prevent compromise, viruses, …
- Quality of Service Guarantees (QOS)
  - Pre- or dynamically allocate bandwidth
  - E.g., diffserv, RSVP
  - Helps where such things are available …
- Content replication
  - E.g., CDS: Useful for static content
DOS Prevention - Reverse-Turing Tests

- **Turing test**: measures whether a human can tell the difference between a human or computer (AI)
- **Reverse Turning tests**: measures whether a user on the internet is a person, a bot, whatever?
- **CAPTCHA** - completely automated public Turing test to tell computers and humans apart
  - contorted image humans can read, computers can’t
  - image processing pressing SOA, making these harder

- Note: often used not just for DOS prevention, but for protecting “free” services (email accounts)
Problem with CAPTCHAs

- Accessibility
- Crowdsourcing
DoS Prevention - Puzzles

• Make the solver present evidence of “work” done
  ‣ If work is proven, then process request
  ‣ Note: only useful if request processing significantly more work than

• Puzzle design
  ‣ Must be hard to solve
  ‣ Easy to Verify

• Canonical Example
  ‣ Puzzle: given all but k-bits of r and h(r), where h is a cryptographic hash function
  ‣ Solution: Invert h(r)
  ‣ Q: Assume you are given all but 20 bits, how hard would it be to solve the puzzle?
**Traceback**

- Routers forward packet data to source
  - Include packets and previous hop …
  - At low frequency (1/20,000) …
- Targets reconstruct path to source (IP unreliable)
  - Use per-hop data to look at
  - Statistics say that the path will be exposed
- Enact standard
  - Add filters at routers along the path
Network vs. Web Security
What is the web?

- A collection of application-layer services used to distribute content
  - Web content (HTML)
  - Multimedia
  - Email
  - Instant messaging

- Many applications
  - News outlets, entertainment, education, research and technology, …
  - Commercial, consumer and B2B
Web security: the high bits

- The largest distributed system in existence
  - threats are as diverse as applications and users
  - But need to be thought out carefully …

- The stakeholders are …
  - Consumers (users, businesses, *agents*, …)
  - Providers (web-servers, IM services, …)

- Another way of seeing web security is
  - Securing the web *infrastructure* such that the *integrity*, *confidentiality*, and *availability* of content and user information is maintained
Secure Socket Layer (SSL/TLS)

- Used to authenticate servers
  - Uses certificates, “root” CAs
- **Can** authenticate clients
- Inclusive security protocol
- Security at the socket layer
  - Transport Layer Security (TLS)
  - Provides
    - authentication
    - confidentiality
    - integrity
SSL Handshake

1. Client Hello (algorithms, …)
2. Server Hello (alg. selection, …)
3. Server Certificate
4. ClientKeyRequest
5. ChangeCipherSuite
6. ChangeCipherSuite
7. Finished
8. Finished
**Simplified Protocol Detail**

*Participants:* Alice/A (client) and Bob/B (server)

*Crypto Elements:* Random R, Certificate C, $k_i^+$ Public Key (of i)

*Crypto Functions:* Hash function $H(x)$, Encryption $E(k, d)$, Decryption $D(k, d)$, Keyed MAC $HMAC(k, d)$

1. Alice $\rightarrow$ Bob $R_A$

2. Bob $\rightarrow$ Alice $R_B, C_B$
   - Alice: pick pre-master secret $S$
   - Alice: calculate master secret $K = H(S, R_A, R_B)$

3. Alice $\rightarrow$ Bob $E(k_B^+, S), HMAC(K', CLNT' + [\#1, \#2])$
   - Bob: recover pre-master secret $S = D(k_B^-, E(k_B^+, S))$
   - Bob: calculate master secret $K = H(S, R_A, R_B)$

4. Bob $\rightarrow$ Alice $HMAC(K', SRV R' + [\#1, \#2])$

*Note:* Alice and Bob: IV Keys, Encryption Keys, and Integrity Keys 6 keys, where each key $k_i = g_i(K, R_A, R_B)$, and $g_i$ is key generator function.
Advantages of SSL

- Confidential session
- Server authentication*
- GUI clues for users
- Built into every browser
- Easy to configure on the server
- Protocol has been analyzed like crazy
- Seems like you are getting security “for free”
Disadvantages of SSL

- Users don’t check certificates
  - most don’t know what they mean
- Too easy to obtain certificates
- Too many roots in the browsers
- Some settings are terrible
  - SSL v2 is on
    - totally insecure cipher suites are included
- very little use of client-side certificates
- performance!
  - early days had sites turning off
  - getting better (crypto coprocessors, etc.)
Reality of SSL

- SSL is here to stay no matter what
- credit card over SSL connection is probably safer than credit card to waiter
- biggest hurdles:
  - performance
  - user education (check those certificates)
  - too many trusted sites (edit your browser prefs)
  - misconfiguration (turn off bad ciphersuites)
  - can be used for many non-web applications
Cookies

- Cookies were designed to offload server state to browsers
  - Not initially part of web tools (Netscape)
  - Allows users to have cohesive experience
  - E.g., flow from page to page,
- Someone made a design choice
  - Use cookies to authenticate and authorize users
  - E.g. Amazon.com shopping cart, WSJ.com
Cookie Issues ...

- New design choice means
  - Cookies must be protected
    - Against forgery (integrity)
    - Against disclosure (confidentiality)
- Cookies not robust against web designer mistakes
  - Were never intended to be
  - Need the same scrutiny as any other tech.

Many security problems arise out of a technology built for one thing incorrectly applied to something else.
Cookie Design 1: mygorilla.com

- Requirement: authenticate users on site

- Design:
  1. use digest authentication to login user
  2. set cookie containing hashed username
  3. check cookie for hashed username

- Q: Is there anything wrong with this design?
Cookie Design 2: mygorilla.com

- Requirement: authenticate users on site

  mygorilla.com

- Design:
  1. use digest authentication to login user
  2. set cookie containing encrypted username
  3. check cookie for encrypted username

- Q: Is there anything wrong with this design?
Exercise: Cookie Design

- Design a secure cookie for mygorilla.com that meets the following requirements

- Requirements
  - Users must be authenticated (assume digest completed)
  - Time limited (to 24 hours)
  - Unforgeable (only server can create)
  - Privacy-protected (username not exposed)
  - Location safe (cannot be replayed by another host)
Dynamic Content

- Server generates content at run time
  - For time-sensitive information (stock ticker)
  - For user customization (Amazon.com)
  - Provide HTML interface to complex system (e.g., course management system)
Dynamic Content: CGI

- Common Gateway Interface (CGI)
  - Generic way to call external applications on the server
  - Passes URL to external program (e.g., form)
  - Result is captured and return to requestor

- Historically
  - “shell” scripts used to generate content
    - Very, very dangerous

- NOTE: server extensions are also dangerous (e.g., servlets)
DC: Embedded Scripting

- Program placed directly in content, run at during request time and output returned in content
  - MS active server pages (ASP)
  - PHP
  - mod_perl
  - server-side JavaScript
  - python, ....

- Nice at generating output
  - Dangerous if tied to user input
Warning: Cross-Site Scripting

- Note Assume the following is posted to a message board on your favorite website:

  Hello message board.

  \(<SCRIPT>\text{malicious code</SCRIPT>}

  This is the end of my message.

- Now a reasonable ASP (or some other dynamic content generator) uses the input to create a webpage (e.g., blogger nonsense).

- Now a malicious script is now running
  - Applet, ActiveX control, …
Dynamic Content Security

- Largely just applications
  - Inasmuch as application are secure
  - Command shells, interpreters, are dangerous
- Three things to prevent DC vulnerabilities
  - Validate input
    - Input often received as part of user supplied data
    - E.g., cookie
  - Limit program functionality
    - Don’t leave open ended-functionality
  - Execute with limited privileges
Web Content (client side)

- All providers serve up content …
- All sorts of technologies to improve content
  - Interactivity: Forms, CGI, Javascript, …
  - Web applications: Java, Flash, ActiveX…
  - Dynamic content: Servlets, Active Server Pages …
- However, these come with risks …
  - Both clients and servers must use complex and sometimes untried technologies …
  - … that have led to some nasty security problems.
Applications/Plugins

- A plugin is a simply a program used by a browser to process content
  - MIME type maps content to plugin
  - Like any old application (e.g., RealAudio)
  - Newer browsers have autoinstall features

- A kind of plug-in …
  - (1997) David.exe
  - “Free pornography …”

- Moral: beware of plugins
JavaScript

- Scripting Language used to improve the quality/experience
  - Create dialogs, forms, graphs, ...
  - Built upon API functions (lots of different flavors)
  - No ability to read local files, open connections …

- Security: No ability to read local files, open connections, but …
  - DOS – the “infinite popup” script
    - Often could not “break out” with restarting computer
  - Spoofing – easy to create “password” dialogs
Active X

- ActiveX is a MS windows technology
  - Really, just a way to run arbitrary code
  - Called controls (.OCX), just programs
  - Conforms to MS APIs to interact with web
- Extends user experience in lots of nice ways
  - Microsoft upgrade service
  - BIOS Upgrades
  - Lookup services
- **Massive** security hole ....
Is there a concern?

- Initially, MS thought that users would have no problem with ActiveX controls
  - Hey, you run programs you buy, right?
  - With traditional applications
    - You (generally) know who the software comes from
    - You (generally) have some recourse
  - On the Internet …
    - Neither of the above may be true
    - User not actually be involved/aware in execution
Java

- Platform and language for writing applets
  - Sun Microsystems platform for set-top boxes
  - Applets embedded in web pages (or native)
  - Language loosely resembling C++
  - Runs in a Java Virtual Machine (JVM)
    - Every platform has JVM
    - Platform runs arbitrary code (bytecode)
    - Hence: one application runs on a bunch of platforms
    - Great way to take advantage of the web
    - Slow for data/processing intensive applications
Drive by downloads

- Using a deceptive means to get someone to install something on their own (spyware/adware)
  - Once you have one, then it starts downloading lots of others, their friends, …
  - *Extortion-ware* (i.e., *crimeware*) – pay us $40 for our popup blocker, etc ….
    - The real gambit is that they demand $40 for the uninstall option
Spyware

- Definition: hidden software that uses local host to transmit user secrets
  - e.g., browsing habits, forms data
- Typically found in “free” software
  - Gnutella, game tools, demo software, MP3 tools ...
  - Implemented using spyware “engines” - gator
- Imbeds in local host to
  - Adds shared libraries (.dlls), adds to startup as TSR programs
  - Often difficult or impossible to remove
    - You are never really sure it is gone (advice: reinstall!!)
Malicious IFrame(s)

- An IFRAME is a HTML tag that create an embedded frame in the content of another page.
  - This is the attack vector de jour for adversaries attempting to delivery content that exploits browser vulnerabilities.
  - E.g., deliver crafted .jpg or malicious scripting
- The attack occurs when the adversary breaks into a webserver and places a IFRAME in legitimate content
  - e.g., by sniffing passwords, recursively adding IFRAMEs