Announcements

• Plan for Today:
  – Return briefly to finish up attacker reconnaissance
  – Access Control

• Project 2 reminder
  – Due: Friday, March 6th (right before Spring Break)
Midterm 1 Statistics

(Out of 80)
Max: 77
Min: 29
Avg: 62
Std Dev: 8
Detecting Attacks

- Attacks (against computer systems) usually consist of several stages:
  - Finding software vulnerabilities
  - Exploiting them
  - Hiding/cleaning up the exploit

- Attackers care about finding vulnerabilities:
  - What machines are available?
  - What OS / version / patch level are the machines running?
  - What additional software is running?
  - What is the network topology?

- Attackers care about not getting caught:
  - How detectible will the attack be?
  - How can the attacker cover her tracks?

- Programs can automate the process of finding/exploiting vulnerabilities.
  - Same tools that sys. admins. use to audit their systems…
  - A worm is just an automatic vulnerability finder/exploiter…
Attacker Reconnaissance

- Network Scanning
  - Existence of machines at IP addresses
  - Attempt to determine network topology
  - ping, tracert

- Port scanners
  - Try to detect what processes are running on which ports, which ports are open to connections.
  - Typical machine on the internet gets 10-20 port scans per day!
  - Can be used to find hit lists for flash worms

- Web services
  - Use a browser to search for CGI scripts, Javascript, etc.
Determining OS information

- Gives a lot of information that can help an attacker carry out exploits
  - Exact version of OS code can be correlated with vulnerability databases
- Sadly, often simple to obtain this information:
  - Just try telnet

```
playground~> telnet hpux.u-aizu.ac.jp
Trying 163.143.103.12 ...
Connected to hpux.u-aizu.ac.jp.
Escape character is '^]'.
HP-UX hpux B.10.01 A 9000/715 (ttyp2)
login:
```
Determining OS

- Or ftp:

```
$ ftp ftp.netscape.com 21
Connected to ftp.gftp.netscape.com.
220-36
220 ftpnscp.newaol.com FTP server (SunOS 5.8) ready.
Name (ftp.netscape.com:stevez):
331 Password required for stevez.
Password:
530 Login incorrect.
ftp: Login failed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> system
215 UNIX Type: L8 Version: SUNOS
ftp>
```
Determining OS

- Exploit different implementations of protocols
  - Different OS’s have different behavior in some cases
- Consider TCP protocol, there are many flags and options, and some unspecified behavior
  - Reply to bogus FIN request for TCP port (should not reply, but some OS’s do)
  - Handling of invalid flags in TCP packets (some OS’s keep the invalid flags set in reply)
  - Initial values for RWS, pattern in random sequence numbers, etc.
  - Can narrow down the possible OS based on the combination of implementation features
- Tools can automate this process
Auditing: Remote auditing tools

- Several utilities available to “attack” or gather information about services/daemons on a system.
  - SATAN (early 1990’s): 
    Security Administrator Tool for Analyzing Networks
  - SAINT - Based on SATAN utility
  - SARA - Also based on SATAN
  - Nessus - Open source vulnerability scanner
    - http://www.nessus.org
  - Nmap

- Commercial:
  - ISS scanner
  - Cybercop
Nmap screen shot

http://www.insecure.org/nmap
http://www.insecure.org/nmap/nmap-fingerprinting-article.html
Today's Plan

• We've seen how worms and viruses spread.
• What can we do about it?
  – Proactive:
    • Produce good software (eliminate vulnerabilities)
    • Limit the damages that can be done
  – Reactive: install filtering configure firewalls to drop packets

• Restrict access to OS resources?
  – If one could prevent a worm or virus from tampering with the file system or restrict their access to other functionality, the damage they can do is limited.

• Today: access control more generally
Authorization

- A principal is an entity that has a bearing on the security properties of a system.
  - Example principals: Users, Hosts, Processes, “the Attacker”, etc.

- Authorization is the process of determining whether a principal is permitted to perform a particular action.

- Access control is necessary at many levels of abstraction in a computing system:
  - Firewalls are one example of an access control mechanism.
  - Others?
The “Gold” Standard

• Authentication
  – Identify which principals take which actions

• Authorization
  – Determine what actions are permissible

• Audit
  – Recording the security relevant actions

• We discussed auditing in one context – there’s more to say about that later.

• This rest of this lecture is about authorization.

• We'll get to authentication in a few lectures.
Policy vs. Mechanism

- **Access control policy is a specification**
  - Given in terms of a model of the system
  - Subjects: do things (i.e. a process writes to files)
  - Objects: are passive (i.e. the file itself)
  - Actions: what the subjects do (i.e. read a string from a file)
  - Rights: describe authority (i.e. read or write permission)

- **Mechanisms are used to implement a policy**
  - Example: access control bits in Unix file system & OS checks
  - Mechanism should be general; ideally should not constrain the possible policies.
  - Complete mediation: every access must be checked
Access Control Matrices

<table>
<thead>
<tr>
<th>A[s][o]</th>
<th>Obj₁</th>
<th>Obj₂</th>
<th>...</th>
<th>Objₙ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subj₁</td>
<td>{r,w,x}</td>
<td>{r,w}</td>
<td>...</td>
<td>{}</td>
</tr>
<tr>
<td>Subj₂</td>
<td>{w,x}</td>
<td>{}</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Subjₘ</td>
<td>{x}</td>
<td>{r,w,x}</td>
<td>...</td>
<td>{r,w,x}</td>
</tr>
</tbody>
</table>

Each entry contains a set of rights.
Access Control Checks

• Suppose subject \( s \) wants to perform action that requires right \( r \) on object \( o \):

• If \( r \in A[s][o] \) then perform action
  else access is denied
Rights

• Besides read, write, execute rights there are many others:
• Ownership
• Creation
  – New subjects (i.e. in Unix add a user)
  – New objects (i.e. create a new file)
  – New rights: Grant right r to subject s with respect to object o (sometimes called delegation)
• Deletion of
  – Subjects
  – Objects
  – Rights (sometimes called revocation)
Access Control Examples

• Assume OS is a subject with all rights
• To create a file f owned by Alice:
  – Create object f
  – Grant own to Alice with respect to f
  – Grant read to Alice with respect to f
  – Grant write to Alice with respect to f
• To start a login for Alice
  – Input and check password
  – Create a shell process p
  – Grant own_process to Alice with respect to p
Reference Monitors

Subject -> Request (Action, Object) -> Monitor

Monitor consults policy to make decision

- Granted
- Denied
Reference Monitors

• Criteria
  – Correctness
  – Complete mediation (all avenues of access must be protected)
  – Expressiveness (what policies are admitted)
  – How large/complex is the mechanism?

• Trusted Computing Base (TCB)
  – The set of components that must be trusted to enforce a given security policy
  – Would like to simplify/minimize the TCB to improve assurance of correctness
Software Mechanisms

- **Interpreters**
  - Check the execution of every instruction
  - Hard to mediate high-level abstractions

- **Wrappers**
  - Only “interpret” some of the instructions
  - What do you wrap?
  - Where do you wrap? (link-time?)

- **Operating Systems**
  - Level of granularity?
  - Context switching overheads?

- **Example**
  - Java and C# runtime systems
Hardware Mechanisms

- Multiple modes of operation
  - User mode (problem state)
  - Kernel mode (supervisor state)

- Specialized hardware
  - Virtual memory support (TLB’s, etc.)
  - Interrupts
Protecting Reference Monitors

- It must not be possible to circumvent the reference monitor by corrupting it
- Mechanisms
  - Type checking
  - Sandboxing: run processes in isolation
  - Software fault isolation: rewrite memory access instructions to perform bounds checking
  - User/Kernel modes
  - Segmentation of memory (OS resources aren’t part of virtual memory system)
  - Physical configuration (e.g. network topology)
Implementing Access Control

• Access control matrices
  – Subjects >> #users (say 1000s)
  – Objects >> #files (say 1,000,000s)
  – To specify “all users read f”
    • Change O(users) entries

• Matrix is typically sparse
  – Store only non-empty entries

• Special consideration for groups of users
Access Control Lists

For each object, store a list of (Subject x Rights) pairs.

<table>
<thead>
<tr>
<th>A[s][o]</th>
<th>Obj₁</th>
<th>Obj₂</th>
<th>...</th>
<th>Objₙ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subj₁</td>
<td>{r,w,x}</td>
<td>{r,w}</td>
<td>...</td>
<td>{}</td>
</tr>
<tr>
<td>Subj₂</td>
<td>{w,x}</td>
<td>{}</td>
<td>...</td>
<td>{r}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Subjₘ</td>
<td>{x}</td>
<td>{r,w,x}</td>
<td>...</td>
<td>{r,w,x}</td>
</tr>
</tbody>
</table>
Access Control Lists

- Resolving queries is linear in length of the list
- Revocation w.r.t. a single object is easy
- “Who can access this object?” is easy
  - Useful for auditing
- Lists could be long
  - Factor into groups (lists of subjects)
  - Give permissions based on group
  - Introduces consistency question w.r.t. groups
- Authentication critical
  - When does it take place? Every access would be expensive.
Representational Completeness

- **Access Control Lists**
  - Can represent any access control matrix
  - Potentially very large
  - Used in windows file system, NTFS

- **Unix file permissions (next topic)**
  - Fixed size
  - Can't naturally express some access control policies/matrices
Unix file security

• Each file has owner and group
• Permissions set by owner
  – Read, write, execute
  – Owner, group, other
  – Represented by vector of four octal values
• Only owner, root can change permissions
  – This privilege cannot be delegated or shared
• Setid bits – Discuss in a few slides
Question

• "owner" can have fewer privileges than "other"
  – What happens?
    • User gets access?
    • User does not?

• Prioritized resolution of differences
  if user = owner then owner permission
  else if user in group then group permission
  else other permission
Unix Policies Interact

/home/jeff/        jeff  jeff  -rwx  ---  ---
/home/jeff/.bashrc jeff  jeff  -rwx  r--  r--

• stevez cannot read /home/jeff/.bashrc
  – The confidentiality/availability of an object depends on policies other than it's own.
  – Such interactions make specifying policies hard.
  – Problem is not limited to unix (or file systems).
Setid bits on executable Unix file

• Three setid bits
  – Sticky
    • Off: if user has write permission on directory, can rename or remove files, even if not owner
    • On: only file owner, directory owner, and root can rename or remove file in the directory
  – Setuid – set EUID of process to ID of file owner
    – passwd owned by root and setuid is true
    – Jeff executes passwd: “passwd runs as root”
  – Setgid – set EGID of process to GID of file
Effective User ID (EUID)

- Each process has three user IDs (more in Linux)
  - Real user ID (RUID)
    - same as the user ID of parent (unless changed)
    - used to determine which user started the process
  - Effective user ID (EUID)
    - from set user ID bit on program file, or system call
    - determines the permissions for process
      - file access and port binding
  - Saved user ID (SUID)
    - So previous EUID can be restored

- Real group ID, effective group ID, used similarly
Process Operations and IDs

- **Root**
  - ID=0 for superuser root; can access any file

- **Fork and Exec**
  - Inherit three IDs, except when executing a file with setuid bit on.

- **Setuid system calls**
  - seteuid(newid) can set EUID to
    - Real ID or saved ID, regardless of current EUID
    - Any ID, if EUID=0

- **Details are actually more complicated**
  - Several different calls: setuid, seteuid, setruid
Example

```
...;
...;
exec( );
```

```
RUID 25
Owner 18
-rw-r--r--
file
read/write
```

```
EUID 18
i=getruid()
setuid(i);
...;
...;
```

```
RUID 25
EUID 25
Owner 25
-rw-r--r--
file
read/write
```

```
RUID 25
Owner 18
SetUID
program
```
Setuid programming

- Can do anything that owner of file is allowed to do
- Be Careful!
  - Root can do anything; don’t get tricked (no middle ground)
  - Principle of least privilege – change EUID when root privileges no longer needed
  - Be sure not to
    - Take action for untrusted user
    - Return secret data to untrusted user

- Setuid scripts
  - This is a bad idea
  - Historically, race conditions
    - Begin executing setuid program; change contents of program before it loads and is executed
Unix summary

• We’re all very used to this …
  – So probably seems pretty good
  – We overlook ways it might be better

• Good things
  – Some protection from most users
  – Flexible enough to make things possible

• Main bad thing
  – Too tempting to use root privileges
  – No way to assume some root privileges without all root privileges
Capabilities Lists

For each subject, store a list of (Object x Rights) pairs.
Capabilities

- A capability is a (Object, Rights) pair
  - Used like a movie ticket e.g.:
    (“Cloverfield”, {admit one, 7:00pm show})

- Should be unforgeable
  - Otherwise, subjects could get illegal access

- Authentication takes place when the capabilities are granted (not needed at use)

- Harder to do revocation (must find all tickets)

- Easy to audit a subject, hard to audit an object
Implementing Capabilities

• Must be able to name objects
• Unique identifiers
  – Must keep map of UIDs to objects
  – Must protect integrity of the map
  – Extra level of indirection to use the object
  – Generating UIDs can be difficult
• Pointers
  – Name changes when the object moves
  – Remote pointers in distributed setting
  – Aliasing possible
Unforgeability of Capabilities

- Special hardware: tagged words in memory
  - Can’t copy/modify tagged words
- Store the capabilities in protected address space
- Could use static scoping mechanism of safe programming languages.
  - Java’s “private” fields
- Could use cryptographic techniques
  - OS kernel could sign (Object, Rights) pairs using a private key
  - Any process can verify the capability
  - Example: Kerberos