Secure Architecture Principles

• Isolation and Least Privilege
• Access Control Concepts
• Operating Systems
• Browser Isolation and Least Privilege

Original slides were created by Prof. John Mitchel
Secure Architecture Principles

Isolation and Least Privilege
Principles of Secure Design

“Every battle is won BEFORE it is fought.”

“Victorious warriors win first and then go to war, while defeated warriors go to war first and then seek to win.”

- Sun Tzu
Principles of Secure Design

• Compartmentalization
  – Isolation
  – Principle of least privilege

• Defense in depth
  – Use more than one security mechanism
  – Secure the weakest link
  – Fail securely

• Keep it simple
Principle of Least Privilege

• What’s a privilege?
  – Ability to access or modify a resource

• Assume compartmentalization and isolation
  – Separate the system into isolated compartments
  – Limit interaction between compartments

• Principle of Least Privilege
  – A system module should only have the minimal privileges needed for its intended purposes
Monolithic design

Network
User input
File system

System

Network
User device
File system
Monolithic design

System

Network
User input
File system

Network
User device
File system
Monolithic design

Network
User input
File system

Network
User display
File system
Component design

Network
User input
File system

Network
User display
File system
Component design
Component design
Principle of Least Privilege

• What’s a privilege?
  – Ability to access or modify a resource
• Assume compartmentalization and isolation
  – Separate the system into isolated compartments
  – Limit interaction between compartments
• Principle of Least Privilege
  – A system module should only have the minimal privileges needed for its intended purposes
Example: Mail Agent

• Requirements
  – Receive and send email over external network
  – Place incoming email into local user inbox files

• Sendmail
  – Traditional Unix
  – Monolithic design
  – Historical source of many vulnerabilities

• Qmail
  – Compartmentalized design
OS Basics (before examples)

• Isolation between processes
  – Each process has a UID
    • Two processes with same UID have same permissions
  – A process may access files, network sockets, ....
    • Permission granted according to UID

• Relation to previous terminology
  – Compartment defined by UID
  – Privileges defined by actions allowed on system resources
Qmail design

• Isolation based on OS isolation
  – Separate modules run as separate “users”
  – Each user only has access to specific resources

• Least privilege
  – Minimal privileges for each UID
  – Only one “setuid” program
    • setuid allows a program to run as different users
  – Only one “root” program
    • root program has all privileges
Structure of qmail

- qmail-smtpd
- qmail-queue
- qmail-send
  - qmail-rspawn
  - qmail-remote
- qmail-inject
  - qmail-lspawn
  - qmail-local

Incoming external mail

Incoming internal mail
Isolation by Unix UIDs

qmailq – user who is allowed to read/write mail queue

user

root

setuid user

user
Android process isolation

- Android application sandbox
  - Isolation: Each application runs with its own UID in own VM
    - Provides memory protection
    - Communication limited to using Unix domain sockets
    - Only ping, zygote (spawn another process) run as root
  - Interaction: reference monitor checks permissions on inter-component communication
  - Least Privilege: Applications announces permission
    - User grants access at install time
Secure Architecture Principles

Access Control Concepts
Access control

• Assumptions
  – System knows who the user is
    • Authentication via name and password, other credential
  – Access requests pass through gatekeeper (reference monitor)
    • System must not allow monitor to be bypassed
# Access control matrix

[Lampson]

<table>
<thead>
<tr>
<th>Subjects</th>
<th>File 1</th>
<th>File 2</th>
<th>File 3</th>
<th>...</th>
<th>File n</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>read</td>
<td>write</td>
<td>-</td>
<td>-</td>
<td>read</td>
</tr>
<tr>
<td>User 2</td>
<td>write</td>
<td>write</td>
<td>write</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>User 3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>read</td>
<td>read</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User m</td>
<td>read</td>
<td>write</td>
<td>read</td>
<td>write</td>
<td>read</td>
</tr>
</tbody>
</table>
Implementation concepts

- Access control list (ACL)
  - Store column of matrix with the resource
- Capability
  - User holds a “ticket” for each resource
  - Two variations
    - store row of matrix with user, under OS control
    - unforgeable ticket in user space

Access control lists are widely used, often with groups
Some aspects of capability concept are used in many systems
ACL vs Capabilities

• Access control list
  – Associate list with each object
  – Check user/group against list
  – Relies on authentication: need to know user

• Capabilities
  – Capability is unforgeable ticket
    • Random bit sequence, or managed by OS
    • Can be passed from one process to another
  – Reference monitor checks ticket
    • Does not need to know identify of user/process
ACL vs Capabilities

User U
Process P

User U
Process Q

User U
Process R

Capability c,d,e
Process P

Capability c,e
Process Q

Capability c
Process R
ACL vs Capabilities

• Delegation
  – Cap: Process can pass capability at run time
  – ACL: Try to get owner to add permission to list?
    • More common: let other process act under current user

• Revocation
  – ACL: Remove user or group from list
  – Cap: Try to get capability back from process?
    • Possible in some systems if appropriate bookkeeping
      – OS knows which data is capability
      – If capability is used for multiple resources, have to revoke all or none ...
    • Indirection: capability points to pointer to resource
      – If C → P → R, then revoke capability C by setting P=0
Roles (aka Groups)

- **Role** = set of users
  - Administrator, PowerUser, User, Guest
  - Assign permissions to roles; each user gets permission

- **Role hierarchy**
  - Partial order of roles
  - Each role gets permissions of roles below
  - List only new permissions given to each role

```
Administrator
     |        |
     v        v
PowerUser
     |        |
     v        v
User
     |        |
     v        v
Guest
```
Role-Based Access Control

Advantage: users change more frequently than roles
Access control summary

• Access control involves reference monitor
  – Check permissions: \langle\text{user info}, \text{action}\rangle \rightarrow \text{yes/no}
  – Important: no way around this check

• Access control matrix
  – Access control lists vs capabilities
  – Advantages and disadvantages of each

• Role-based access control
  – Use group as “user info”; use group hierarchies
Secure Architecture Principles

Operating Systems
Unix access control

- Process has user id
  - Inherit from creating process
  - Process can change id
    - Restricted set of options
    - Special “root” id
      - All access allowed
- File has access control list (ACL)
  - Grants permission to user ids
  - Owner, group, other

<table>
<thead>
<tr>
<th></th>
<th>File 1</th>
<th>File 2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>read</td>
<td>write</td>
<td>-</td>
</tr>
<tr>
<td>User 2</td>
<td>write</td>
<td>write</td>
<td>-</td>
</tr>
<tr>
<td>User 3</td>
<td>-</td>
<td>-</td>
<td>read</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User m</td>
<td>Read</td>
<td>write</td>
<td>write</td>
</tr>
</tbody>
</table>
Unix file access control list

• 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
Process effective user id (EUID)

- Each process has three IDs (+ more under 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 the file being executed, or sys 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 exec of file with setuid bit

• Setuid system call
  – 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, setreuid
Setid bits on executable Unix file

• Three setid bits
  – Setuid – set EUID of process to ID of file owner
  – Setgid – set EGID of process to GID of file
  – 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
Example

```
...;
...;
exec(
		);
RUID	25
SetUID

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

```
Owner 18
SetUID
program

...;
...;
RUID	25
EUID	18
RUID	25
EUID	25

Owner 18
-rw-r--r--
file

Owner 25
-rw-r--r--
file

Owner 18
-rw-r--r--
file

Owner 25
-rw-r--r--
file

RUID 25
EUID 18
RUID 25
EUID 25
```
Another example

• Why do we need the setuid bit?
  – Some programs need to do privileged operations on behalf of unprivileged users
    • /usr/bin/ping should be able to create raw sockets (needs root)
    • An unprivileged user should be able to run ping
    • Solution: /usr/bin/ping in Linux is owned by root with setuid bit set
SetUID for least privilege: OpenSSH
Unix summary

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

• Main limitation
  – Too tempting to use root privileges
  – No way to assume some root privileges without all root privileges
Weakness in isolation, privileges

• Network-facing Daemons
  – Root processes with network ports open to all remote parties, e.g., sshd, ftpd, sendmail, ...
    • How can you solve this?

• Rootkits
  – System extension via dynamically loaded kernel modules

• Environment Variables
  – System variables such as LD_LIBRARY_PATH that are shared state across applications. An attacker can change LD_LIBRARY_PATH to load an attacker-provided file as a dynamic library
Weakness in isolation, privileges

• Shared Resources
  – Since any process can create files in /tmp directory, an untrusted process may create files that are used by arbitrary system processes

• Time-of-Check-to-Time-of-Use (TOCTTOU)
  – Typically, a root process uses system call to determine if initiating user has permission to a particular file, e.g. /tmp/X.
  – After access is authorized and before the file open, user may change the file /tmp/X to a symbolic link to a target file /etc/shadow.
Secure Architecture Principles

Browser Isolation and Least Privilege
The web browser enforces its own internal policy. If the browser implementation is corrupted, this mechanism becomes unreliable.
Components of security policy

• Frame-Frame relationships
  – canScript(A,B)
    • Can Frame A execute a script that manipulates arbitrary/nontrivial DOM elements of Frame B?
  – canNavigate(A,B)
    • Can Frame A change the origin of content for Frame B?

• Frame-principal relationships
  – readCookie(A,S), writeCookie(A,S)
    • Can Frame A read/write cookies from site S?
Chromium Security Architecture

• Browser ("kernel")
  – Full privileges (file system, networking)
• Rendering engine
  – Up to 20 processes
  – Sandbox
• One process per plugin
  – Full privileges of browser
Chromium

Communicating sandbox components

See: http://dev.chromium.org/developers/design-documents/sandbox/
Design Decisions

• Compatibility
  – Sites rely on the existing browser security policy
  – Browser is only as useful as the sites it can render
  – Rules out more “clean slate” approaches

• Black Box
  – Only renderer may parse HTML, JavaScript, etc.
  – Kernel enforces coarse-grained security policy
  – Renderer to enforces finer-grained policy decisions

• Minimize User Decisions
## Task Allocation

<table>
<thead>
<tr>
<th>Rendering Engine</th>
<th>Browser Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML parsing</td>
<td>Cookie database</td>
</tr>
<tr>
<td>CSS parsing</td>
<td>History database</td>
</tr>
<tr>
<td>Image decoding</td>
<td>Password database</td>
</tr>
<tr>
<td>JavaScript interpreter</td>
<td>Window management</td>
</tr>
<tr>
<td>Regular expressions</td>
<td>Location bar</td>
</tr>
<tr>
<td>Layout</td>
<td>Safe Browsing blacklist</td>
</tr>
<tr>
<td>Document Object Model</td>
<td>Network stack</td>
</tr>
<tr>
<td>Rendering</td>
<td>SSL/TLS</td>
</tr>
<tr>
<td>SVG</td>
<td>Disk cache</td>
</tr>
<tr>
<td>XML parsing</td>
<td>Download manager</td>
</tr>
<tr>
<td>XSLT</td>
<td>Clipboard</td>
</tr>
</tbody>
</table>

**Both**

- URL parsing
- Unicode parsing
Leverage OS Isolation

• Sandbox based on four OS mechanisms
  – A restricted token
  – The Windows job object
  – The Windows desktop object
  – Windows Vista only: integrity levels

• Specifically, the rendering engine
  – adjusts security token by converting SIDS to DENY_ONLY, adding restricted SID, and calling AdjustTokenPrivileges
  – runs in a Windows Job Object, restricting ability to create new processes, read or write clipboard, ..
  – runs on a separate desktop, mitigating lax security checking of some Windows APIs

See: http://dev.chromium.org/developers/design-documents/sandbox/
Evaluation: CVE count

• Total CVEs:

<table>
<thead>
<tr>
<th></th>
<th>Browser</th>
<th>Renderer</th>
<th>Unclassified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Explorer</td>
<td>4</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Firefox</td>
<td>17</td>
<td>40</td>
<td>3</td>
</tr>
<tr>
<td>Safari</td>
<td>12</td>
<td>37</td>
<td>1</td>
</tr>
</tbody>
</table>

• Arbitrary code execution vulnerabilities:

<table>
<thead>
<tr>
<th></th>
<th>Browser</th>
<th>Renderer</th>
<th>Unclassified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Explorer</td>
<td>1</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>Firefox</td>
<td>5</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>Safari</td>
<td>5</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>
Summary

• Security principles
  – Isolation
  – Principle of Least Privilege
  – Qmail example
• Access Control Concepts
  – Matrix, ACL, Capabilities
• OS Mechanisms
  – Unix
    • File system, Setuid
  – Windows
    • File system, Tokens, EFS
• Browser security architecture
  – Isolation and least privilege example