Advanced Unix System Administration

Lecture 15
March 21, 2007

Steven Luo
<sluo+decal@OCF.Berkeley.EDU>

Types of Attacks

- Other memory bugs
  - Use after free(): if the attacker can control what's in that memory afterwards, could lead to nasty security problems
  - Double free()
  - Format string vulnerability (C, C++ programs)
    - User control of the format string allows nasty memory-based attacks
  - With sufficient effort, many (most?) memory use bugs can be exploitable

Types of Attacks

- Temporary file vulnerabilities
  - On most systems, anyone can write to /tmp
  - Imagine the following sequence:
    - Attacker creates symlink /tmp/foo -> file
    - Program does open(“/tmp/foo”, O_WRONLY|O_CREAT|O_TRUNC)
  - Any time temp file names are predictable, there's a problem
    - Even when the name does change, there is a race condition

Types of Attacks

- Some remarks on PHP webapps
  - register_globals is dangerous!
    - Convenient, but discourages input validation
    - With register_globals on, an attacker can set arbitrary variables in your environment!
    - Lots of little ways to exploit this, even if you're careful
    - Sadly, many (most?) PHP apps depend on this, or (even worse) on register_globals emulation
  - PHP allows remote file includes
    - Combined with the above, makes some very dangerous exploits very easy

Types of Attacks

- Attacks that aren't so technically clever
  - Brute force
    - Particularly relevant for authentication systems
    - You can mitigate the problem sometimes, but can't make it go away
    - Always design the system with such attacks in mind!
  - Social engineering
    - Humans can be easier to exploit than computers
    - User education is only part of the solution – limit what your users can do

The Unix Permissions Model

- Users and groups
  - Users and groups have numeric IDs associated with them
  - Groups can contain multiple users
- Process credentials
  - Each process has a set of credentials associated with it
    - Real user ID: set to the UID executing the process at the beginning of the execution
    - Real group ID

The Unix Permissions Model

- Process credentials con't
  - Effective user ID: the UID used for most permissions checks
  - Effective group ID
  - Saved set-user-ID: used for flexibility in setuid applications
  - Saved set-group-ID
  - Note that access control is always by user/group ID!
  - Behavior can be very system-dependent – see the documentation, or try examples

The Unix Permissions Model

- File permissions
  - Files have a user/group ID associated with them
  - File permission bits: binary mask usually written as 4-digit octal
    - High digit: 1 = sticky, 2 = setgid, 4 = setuid
    - 2nd digit: 1 = user execute, 2 = user write, 4 = user read
    - 3rd digit: 1 = group execute, 2 = group write, 4 = group read
    - 4th digit: 1 = other execute, 2 = other write, 4 = other read

The Unix Permissions Model

- File permissions con't
  - Directory permissions:
    - High bit: 1 = deletion restricted, 2 = files created will have group set to directory's group
    - Execute bits mean permission to cd in
  - Access control is by the process's effective IDs
    - On Linux, there is a set of filesystem IDs, almost always equal to the effective UID

The Unix Permissions Model

- POSIX draft ACLs
  - Allow the addition of extra user and group permissions entries
  - A “mask” is set on each file and is ANDed with each ACL entry to determine effective permissions

Impersonating Others

- SUID/SGID execution
  - The changing ID dance
    - The real user/group IDs are inherited from the parent process
    - The effective user and/or group IDs are set to the owner/group of the binary, if the corresponding bit is set
    - The saved set-user/group-IDs are set to the effective user and group IDs
  - The “nosuid” or “nosetuid” attribute on the filesystem prevents changing IDs based on the suid/gid bits

Impersonating Others

- Changing IDs while running
  - Unprivileged programs may change their effective IDs to their real IDs or their saved set-IDs
    - SUSv3 does not specify whether real IDs may be changed
  - Privileged programs may change any of their IDs to anything
    - How to change a particular ID can be quite system-dependent!
    - Keeping track of which IDs are set to what is important for security

Impersonating Others

- Changing IDs while running con't
  - Becoming someone else temporarily
    - Change your effective ID to what you need (if unprivileged, can only be real ID or saved set-ID), using seteuid()/setegid()
    - When done, can change ID back to saved set-ID
  - Dropping privileges
    - Must change real, effective, AND saved set-IDs to new values, so that process cannot regain privileges!
    - setuid()/setgid() do this for privileged processes ONLY; unspecified whether setreuid()/setregid() do