Secure System Development Mechanisms

CS460 Cyber Security Lab

Spring 2010

Reading Material

- Web sites
  - Microsoft links from last lecture
  - Linux Capabilities - “man 7 capabilities” or http://www.linuxjournal.com/article/5737
- Papers
  - “The Security Architecture of qmail”, Hafiz, Johnson, and Afandi. PLoP, 2004. http://hillside.net/plop/2004/papers/mhafiz1/PLoP2004.
  - Setuid Demystified Hao Chen, David Wagner, and Drew Dean. 11th USENIX Security Symposium, 2002.

Outline

- Two security problems and solutions in Windows and Linux
  - Compromise of high privilege program
  - Running code as other users

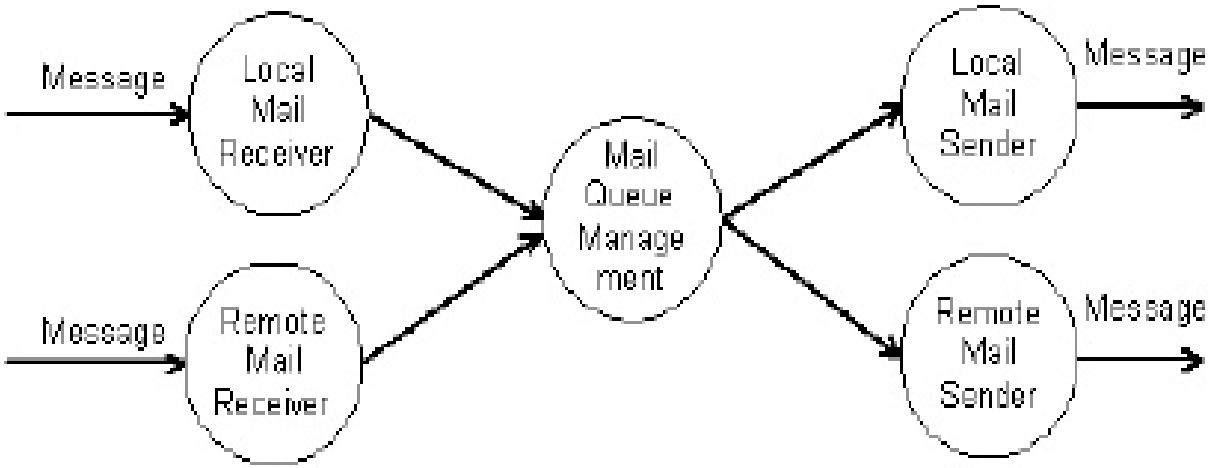
Problem: Exploit on High Privilege Program

- Attacker exploits bug in program or tricks user into running something unexpected
  - Exploits poor input processing on program
  - Surreptitiously causes exploit to be run when viewing mail
- Program is being run as high privilege user (e.g., root in Unix or Administrator in Windows)
  - Exploit is now also running at high privilege and can do most anything to the system

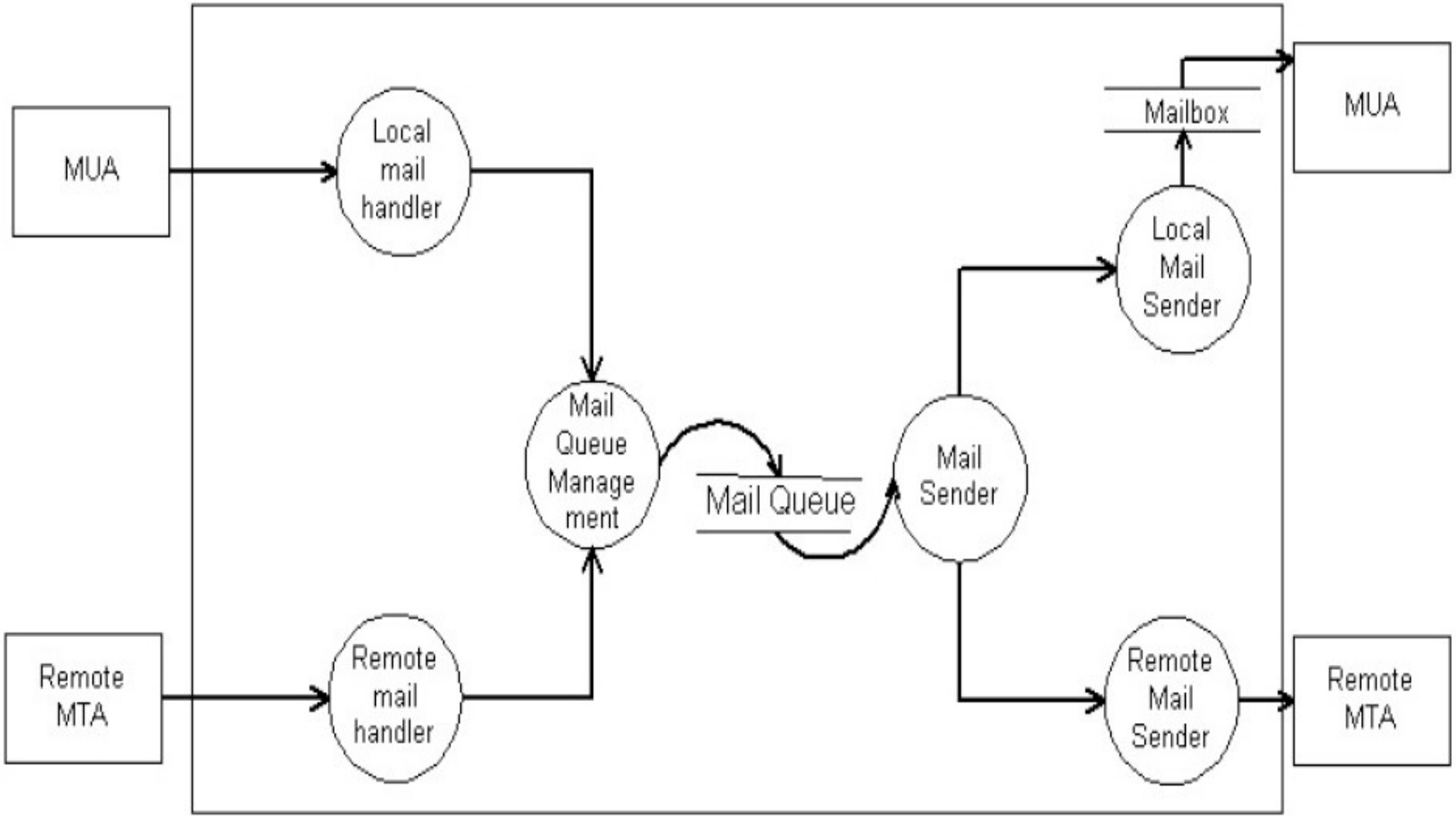
Solution: Modularity

- Divide program into smaller, communicating programs
  - Only subset of the processes need to run at high privilege
  - E.g., qmail as a redesigned MTA replacement for sendmail
- Get simplicity as a side effect
  - Easier to test and analyze for correctness

MTA structure

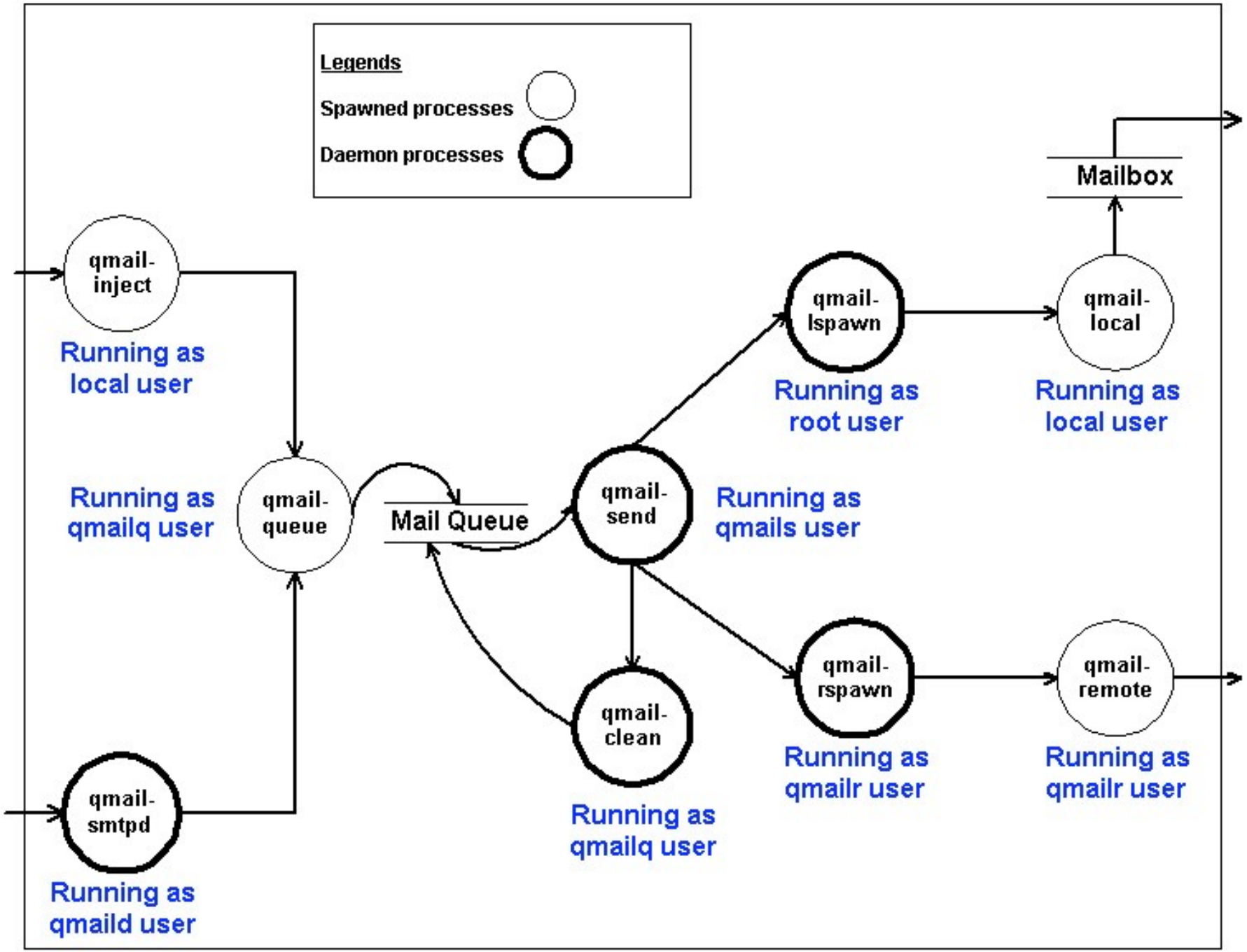


More MTA Structure



Security Patterns

- Compartmentalization
  - Failure in one part of system allows another part to be exploited
  - Put each part in separate security domain. If one part is compromised, the other parts remain secure
- Distributed Responsibility
  - A failure in a component can change any data in that component.
  - Partition data across components.


Legends
Spawned processes
Daemon processes
qmail-inject
Running as local user
qmail-smtpd
Running as qmaild user
qmail-queue
Running as qmailq user
Mail Queue
qmail-send
Running as qmails user
qmail-clean
Running as qmailq user
qmail-lspawn
Running as root user
qmail-rspawn
Running as qmailr user
qmail-local
Running as local user
qmail-remote
Running as qmailr user
Mailbox

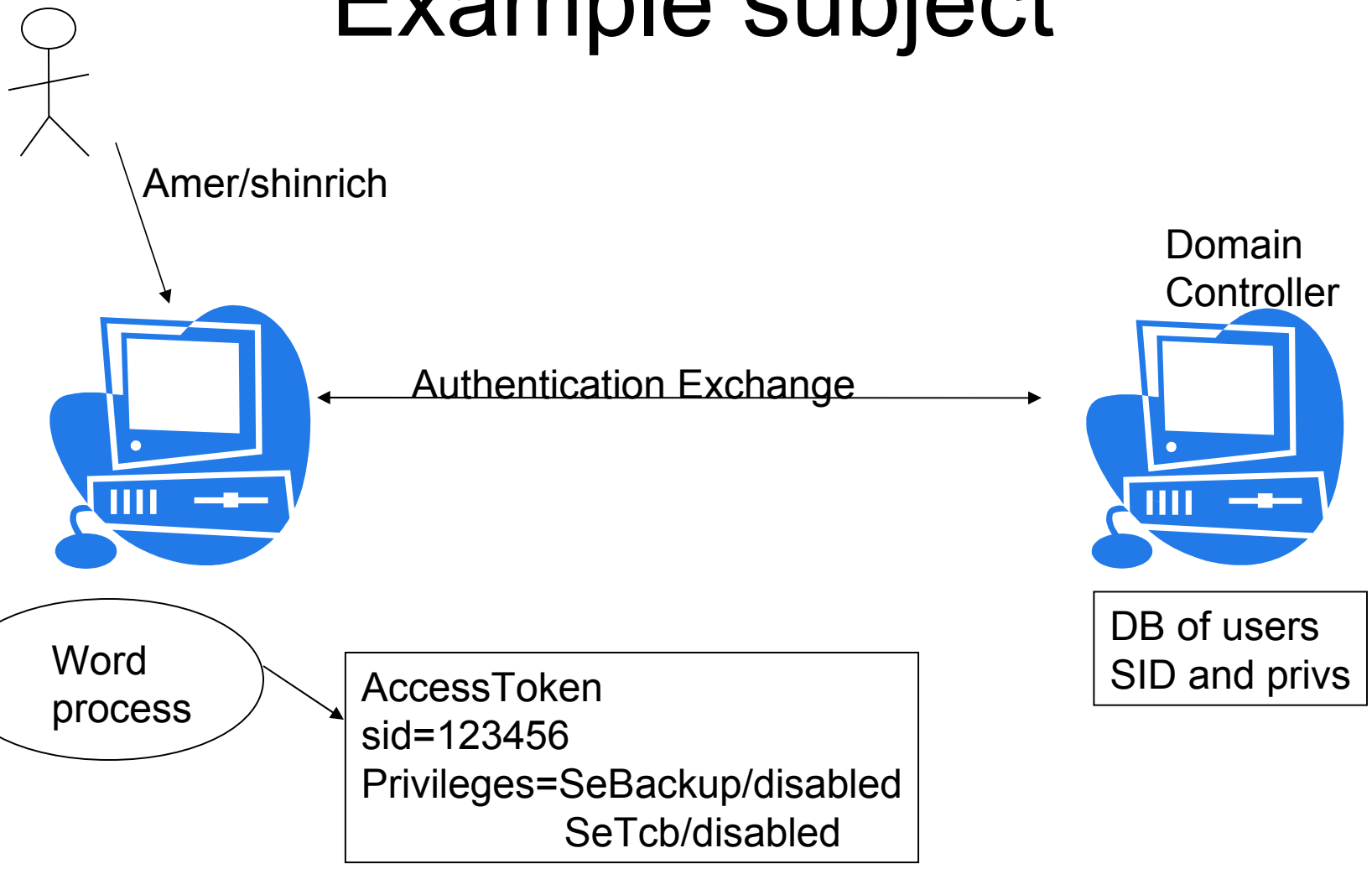
Solution: Least Privilege

- Even high privilege programs only need the extra powers for small parts of its execution
  - Turn off privilege when not needed
  - Permanently drop privileges that are never needed

Windows Security Elements

- Subject – Process or thread running on behalf of the system or an authenticated user
- Security ID (SID) – A globally unique ID that refers to the subject (user or group)
- Access token – the runtime credentials of the subject
- Privilege – ability held by the subject to perform “system” operations. Usually breaks the standard security model
  - Associated with the access token
  - Generally disabled by default.
  - Can be enabled and disabled to run at least privilege
  - Example powerful privileges
    - **SeAssignPrimaryTokenPrivilege** – Replace process token
    - **SeBackupPrivilege** – Ignore file system restrictions to backup and restore
    - **SeIncreaseQuotaPrivilege** - Add to the memory quota for a process
    - **SeTcbPrivilege** – Run as part of the OS
    - Other privileges http://msdn.microsoft.com/library/default.asp?url=/library/en-u

Example subject



Running at reduced privilege

- Two system calls disable or remove privileges from the current access token
  - **AdjustTokenPrivileges** – enables/disables privileges
  - **CreateRestrictedToken** – permanently restrict or remove privileges

Example to Find Token Info

- ```
  // find the buffer size
  DWORD dwSize = 0;
  PTOKEN_PRIVILEGES pPrivileges = NULL;
  GetTokenInformation(hToken,
  TokenPrivileges, NULL, dwSize, &dwSize);

  // allocate the buffer
  pPrivileges = (PTOKEN_PRIVILEGES)
  GlobalAlloc(GPTR, dwSize);

  // now that we have a buffer, try again
  GetTokenInformation(hToken,
  TokenPrivileges, pPrivileges, dwSize,
  &dwSize);
  ```
- MSDN pointer
  http://msdn.microsoft.com/en-us/library/aa446671(VS.85).aspx
  ```

Linux/POSIX Privilege Model

- Privileges called capabilities
  - http://www.linuxjournal.com/article/5737
  - Each process has three capability sets
    - Effective – Set of currently activated privileges
    - Permitted – Set of privileges that process can use
    - Inheritable – Passed onto child processes created by exec
- Can remove capabilities globally
  - Global 32 bit mask that bounds capabilities that can be enabled on the system
  - /proc/sys/kernel/cap-bound can be accessed by lcap utilitiy
  - /usr/include/sys/capability.h

Example

- `lcap CAP_SYS_CHOWN`
  - Once done, it becomes impossible to change a file's owner:

- **`chown nobody test.txt`**
- *`chown: changing ownership of `test.txt':`*
- *`Operation not permitted`*

Set of capabilities

- lcap
- Current capabilities: 0xFFFDFCFF
- 0) *CAP_CHOWN 1) *CAP_DAC_OVERRIDE
- 2) *CAP_DAC_READ_SEARCH 3) *CAP_FOWNER
- 4) *CAP_FSETID 5) *CAP_KILL
- 6) *CAP_SETGID 7) *CAP_SETUID
- 8) *CAP_SETPCAP 9) *CAP_LINUX_IMMUTABLE
- 10) *CAP_NET_BIND_SERVICE 11) *CAP_NET_BROADCAST
- 12) *CAP_NET_ADMIN 13) *CAP_NET_RAW
- 14) *CAP_IPC_LOCK 15) *CAP_IPC_OWNER
- 16) *CAP_SYS_MODULE 17) CAP_SYS_RAWIO
- 18) *CAP_SYS_CHROOT 19) *CAP_SYS_PTRACE
- 20) *CAP_SYS_PACCT 21) *CAP_SYS_ADMIN
- 22) *CAP_SYS_BOOT 23) *CAP_SYS_NICE
- 24) *CAP_SYS_RESOURCE 25) *CAP_SYS_TIME
- 26) *CAP_SYS_TTY_CONFIG 27) *CAP_MKNOD
- 28) *CAP_LEASE 29) *CAP_AUDIT_WRITE
- 30) *CAP_AUDIT_CONTROL
- * = Capabilities currently allowed

Linux Privileges/Capabilities

- Can disable or remove capabilities per process
  - Libcap or setcap/getcap system calls
  - Can specify the affected process, the process group, or all processes
  - Can specify the capability mask for all three sets of capabilities
- Limited by lack of file system support

Problem: Run privileged program portions as regular user

- File server program must have portions run at high privilege, but ultimately only returns information that the invoking user has access to
- More frequently allow low privilege user to run high privilege program

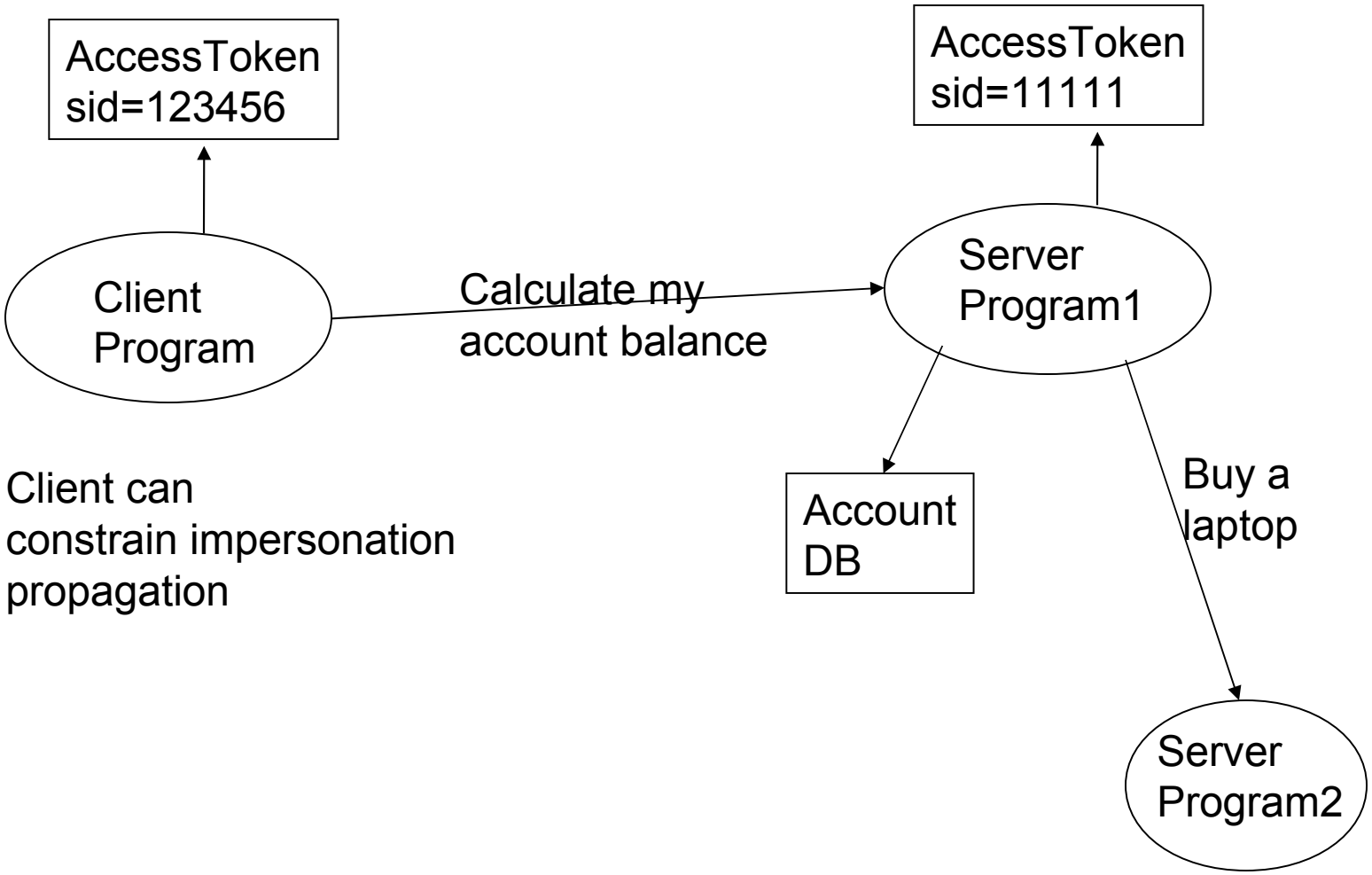
Solution: Impersonation

- Client program runs as end user
- Client program communicates with privileged daemon or service
- Privileged service picks up client's identity
- "Impersonates" client while acting on behalf of the client

Windows Impersonation

- Each process has three access tokens associated
  - Real access token
  - Effective access token
  - Saved access token
- Server program can run with client access token
  - **ImpersonateLoggedOnUser** - runs under the access token of the logged on user
    - Several variations of this system call which pull the impersonation token from various sources
  - **RevertToSelf** to return to the original user
  - **SeImpersonatePrivilege** has been introduced
- Presumably client has lower privilege than server
- Multiple impersonation levels to restrict token propagation

Example impersonation



Impersonation problems

- Knowledgeable exploit can use RevertToSelf
- Base user is most likely a privileged user

Solution: Set User ID

- Mark executable so it runs as a different user than the invoking user
  - Mark file system program to run as privileged user
- Rely on system calls to reset user ID to less privileged user

Unix Set UID

- Each Unix process has three user ID's associated
  - Effective – Used in access checks
  - Real
  - Saved
- setresuid system call enables application to set all three
  - Assuming caller meets requirements, e.g., regular user cannot set UID to 0

SetUID File Bit

- Normally, new process will run under UID of invoking process
- If SetUID bit is set
  - New process will run under executable File's UID for effective UID
  - Real UID will still be that of invoking user.
  - Setting SetUID bit is restricted for normal user

Setting SetUID bit

- Consider executable Foo
  - Owned by Bob
    - What does this mean when run by Fred?
  - Owned by root
    - What does this mean when run by Fred?

SetUID system calls

- This concept has been in Unix since the beginning
- The concept has evolved over time
  - Slightly different calls and semantics in different flavors of Unix
- In general for all flavors
  - Effective user ID of 0, can set effective UID to any value
  - Otherwise can only set effective UID to real or saved UID

Unix Set UID

- Example
  - setuid(getuid())
  - Run as non-root user to permanently clear the root privilege
  - Simple API hides details and may reveal exploitable vulnerability