

# Secure System Development Mechanisms

Cyber Security Lab

Spring 2006

# Reading Material

- Books
  - *Secure Coding: Principles and Practices*, Graff and van Wyk  
<http://www.securecoding.org/>
  - Chapter 13 of Bishop
  - *Writing Secure Code*, Howard and LeBlanc. Written with a Microsoft bent.
- Web sites
  - DHS/SEI BuildSecurityIn <https://buildsecurityin.us-cert.gov/portal/>
  - From MSDN <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/iissdk/html/ff9e706c-edbe-4ffb-bb4b-d916bd424ffd.asp>
  - Linux checklist from LLBN [http://www.lbl.gov/ITSD/Security/systems/redhat-linux-checklist\\_v7.html](http://www.lbl.gov/ITSD/Security/systems/redhat-linux-checklist_v7.html)
  - Linux Capabilities - <http://www.linuxjournal.com/article/5737>
  - [Windows NT Security in Theory and Practice](#)
- Papers
  - “The Security Architecture of gmail”, Hafiz, Johnson, and Afandi. PLoP, 2004.  
[http://hillside.net/plop/2004/papers/mhafiz1/PLoP2004\\_mhafiz1\\_0.pdf](http://hillside.net/plop/2004/papers/mhafiz1/PLoP2004_mhafiz1_0.pdf)
  - [Setuid Demystified](#) Hao Chen, David Wagner, and Drew Dean. [11th USENIX Security Symposium](#), 2002.

# Outline

- Review of general Secure Development
  - 8 Design Principles
- Concentrate on two problems and solutions in Windows and Linux
  - Compromise of high privilege program
  - Running code as other users

# Secure Coding Issue

- Good software engineering principles
  - Common sense
  - Stuff you know you should be doing
  - An art not a science. Valuable to review and be aware of
- Look at the issues from three levels
  - Design, e.g., what are the communication channels
  - Implementation, e.g., are the inputs checked adequately
  - Operational, e.g., what programs are running on the system systems, are the patches up to date.

# Checklists and Security Principles

- Many security check lists and design principles
  - Many lists depend on your environment
    - Target application
    - Development culture
    - Development environment
    - Target audience
- Salzer and Schroeder published one widely referenced list of 8 design principles

# 1. Economy of Mechanism

- *Keep the design as simple and small as possible*
- Simpler means less can go wrong
  - And when errors occur, they are easier to understand and fix
- Interfaces and interactions

## 2. Fail-Safe Defaults

- *Base access decisions on permission rather than exclusion*
- Burden of proof is on the principal seeking permission
- If the protection system fails, then legitimate access is denied but illegitimate access is also denied

# 3. Complete Mediation

- *Every access to every object must be checked for authority*
- Usually done once, on first action
  - UNIX: access checked on open, not checked thereafter
- If permissions change after, may get unauthorized access
- Proposals to gain performance by remembering the result of an authority check should be examined skeptically

# 4. Open Design

- *The design should not be secret*
- Do not depend on secrecy of design or implementation
  - Popularly misunderstood to mean that source code should be public
  - “Security through obscurity”
  - Does not apply to information such as passwords or cryptographic keys

# 5. Separation of Privilege

- *Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.*
- Require multiple conditions to grant privilege
  - Separation of duty
  - Defense in depth

# 6. Least Privilege

- *Every program and every user of the system should operate using the least set of privileges necessary to complete the job*
- A subject should be given only those privileges necessary to complete its task
  - Function, not identity, controls
  - Rights added as needed, discarded after use
  - Minimal protection domain

# 7. Least Common Mechanism

- *Minimize the amount of mechanism common to more than one user and depended on by all users*
- Mechanisms should not be shared
  - Information can flow along shared channels
  - Covert channels
- Isolation
  - Virtual machines
  - Sandboxes

# 8. Psychological Acceptability

- *It is essential that the human interface be designed for ease of use so that users routinely and automatically accept the protection mechanisms correctly*
- Security mechanisms should not add to difficulty of accessing resource
  - Hide complexity introduced by security mechanisms
  - Ease of installation, configuration, use
  - Human factors critical here

# Addressing Security at Design

- Create a good system metaphor
- Understand high-level system flow
  - Break into simple modular systems
  - Understand the trusted computing base
  - Identify points of authentication, data input, communication. Consider unifying multiple points
- Look at the problem in a non-standard way. Or work with others who can.
  - E.g., using privileged mouse driver to co-opt system
  - Standard issue of not being good testers of our own code

# More Design Issues

- It's ok to re-use the same security architecture on multiple projects
  - Many client-server architectures can use the same security paradigm
- How will you fail-safe: Open or closed
  - E.g., for a firewall, should it allow all traffic on catastrophic failure or should it block all traffic.
  - Directs how you will pick defaults
- Re-use proven code components
  - But not just because it happens to be around only if it is proven an appropriate
- Respect chain of trust

# Design Don'ts

- Design should not have “too many moving parts”
  - Keep it simple as feasible
  - Complexity introduces unintended consequences
- Don't make the security annoying
  - E.g., too many different passwords or passwords with too many constraints
- Don't make the security clash with target culture
- Don't design a security solution that costs too much

# Implementation Issues

- Buffer overflows
  - Analysis tools
  - Language techniques, e.g., use Java or use STL in C++
- No Back doors
  - Not even for testing or support.
- Input parser errors
  - Inadequate input error checking. Expectations of input being close to correct.
- Inadequate error checking
  - Unchecked failures leave program in unexpected state (e.g. with too high privilege)

# Implementation issues

- Cleaning memory
  - Who knows what was previously on heap. Or cleanse data before returning it to the heap if it is sensitive.
- Be careful when writing to disk
  - Even temporary data needs to be protected
  - Files can be co-opted

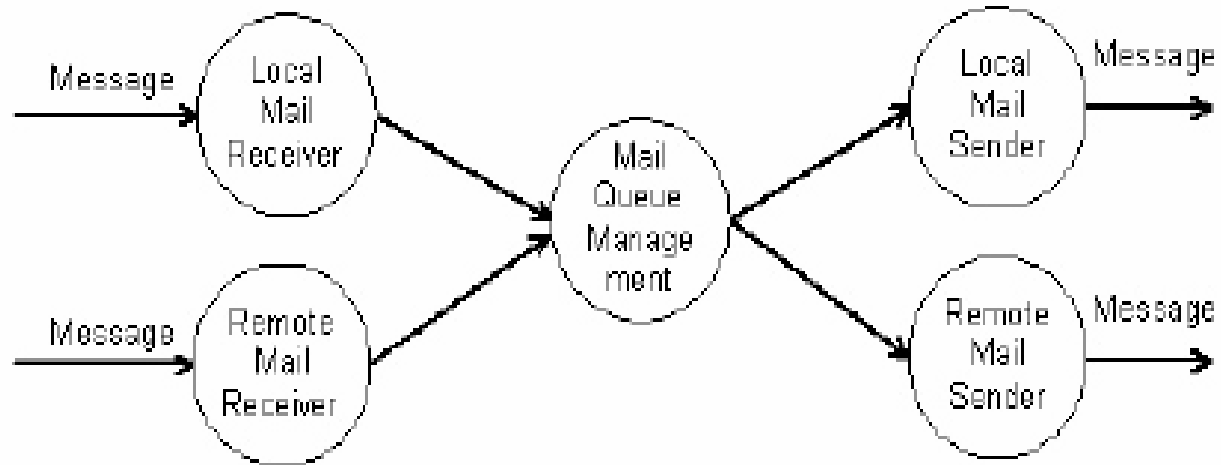
# Problem: Exploit on High Privilege Program

- 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

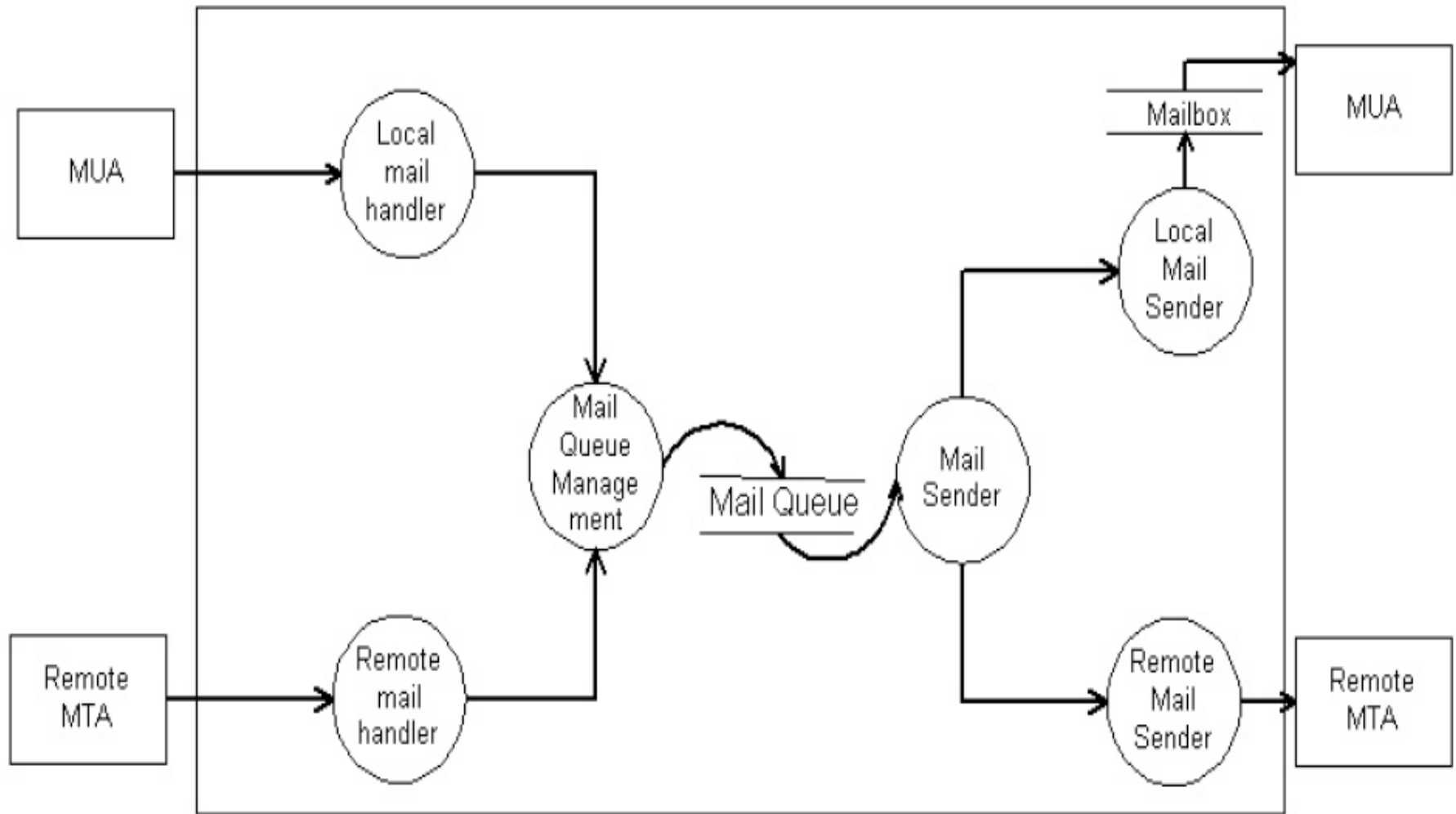
# Solution: Modularity

- 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
    - “The Security Architecture of qmail”, Hafiz, Johnson, and Afandi. PLoP, 2004.  
[http://hillside.net/plop/2004/papers/mhafiz1/PLoP2004\\_mhafiz1\\_0.pdf](http://hillside.net/plop/2004/papers/mhafiz1/PLoP2004_mhafiz1_0.pdf)
- Get simplicity as a side effect
  - Easier to test and analyze for correctness

# MTA structure



# More MTA Structure



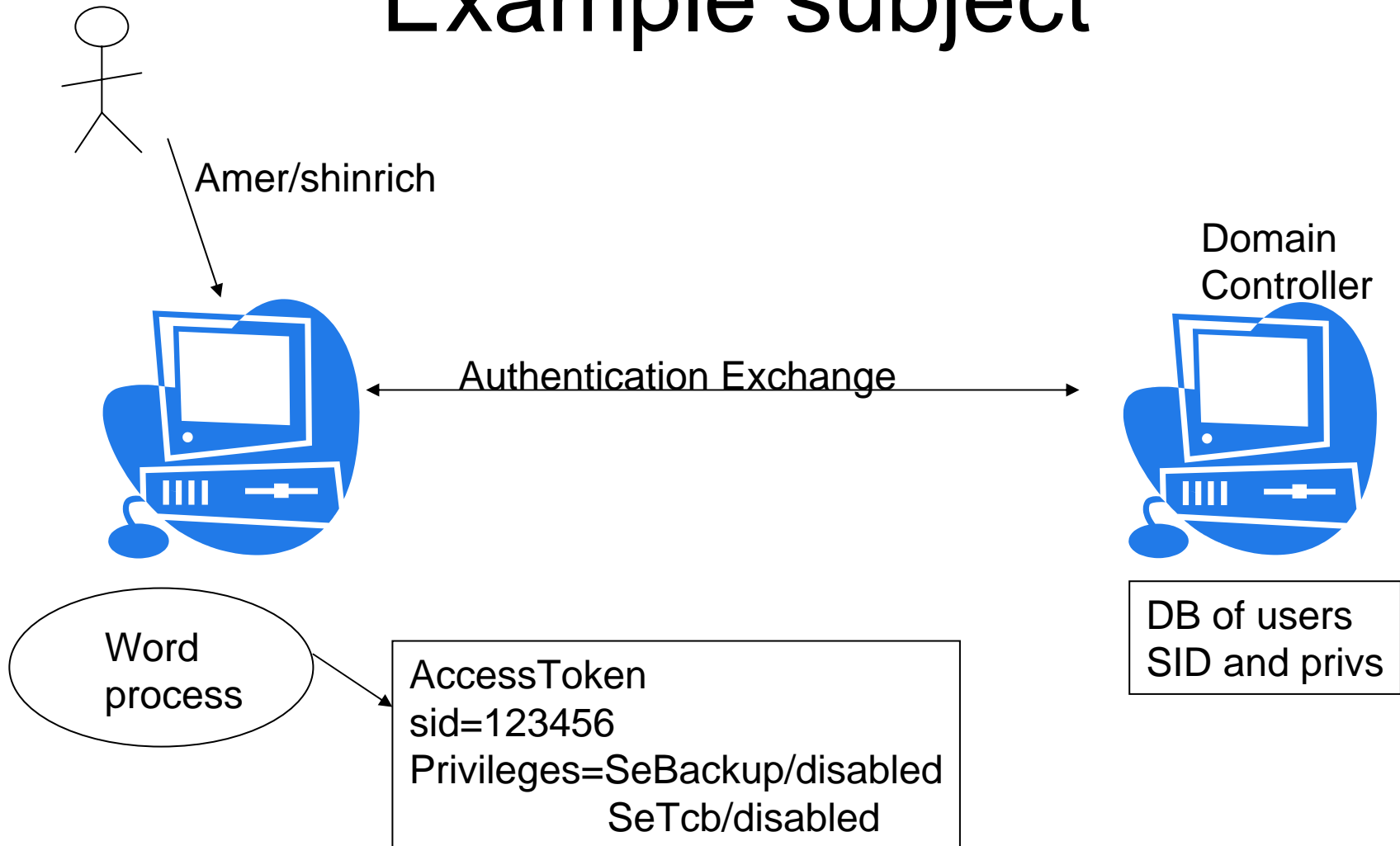
# Solution: Least Privilege

- 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-us/secauthz/security/authorization\\_constants.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secauthz/security/authorization_constants.asp)

# 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

# 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` utility

# 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

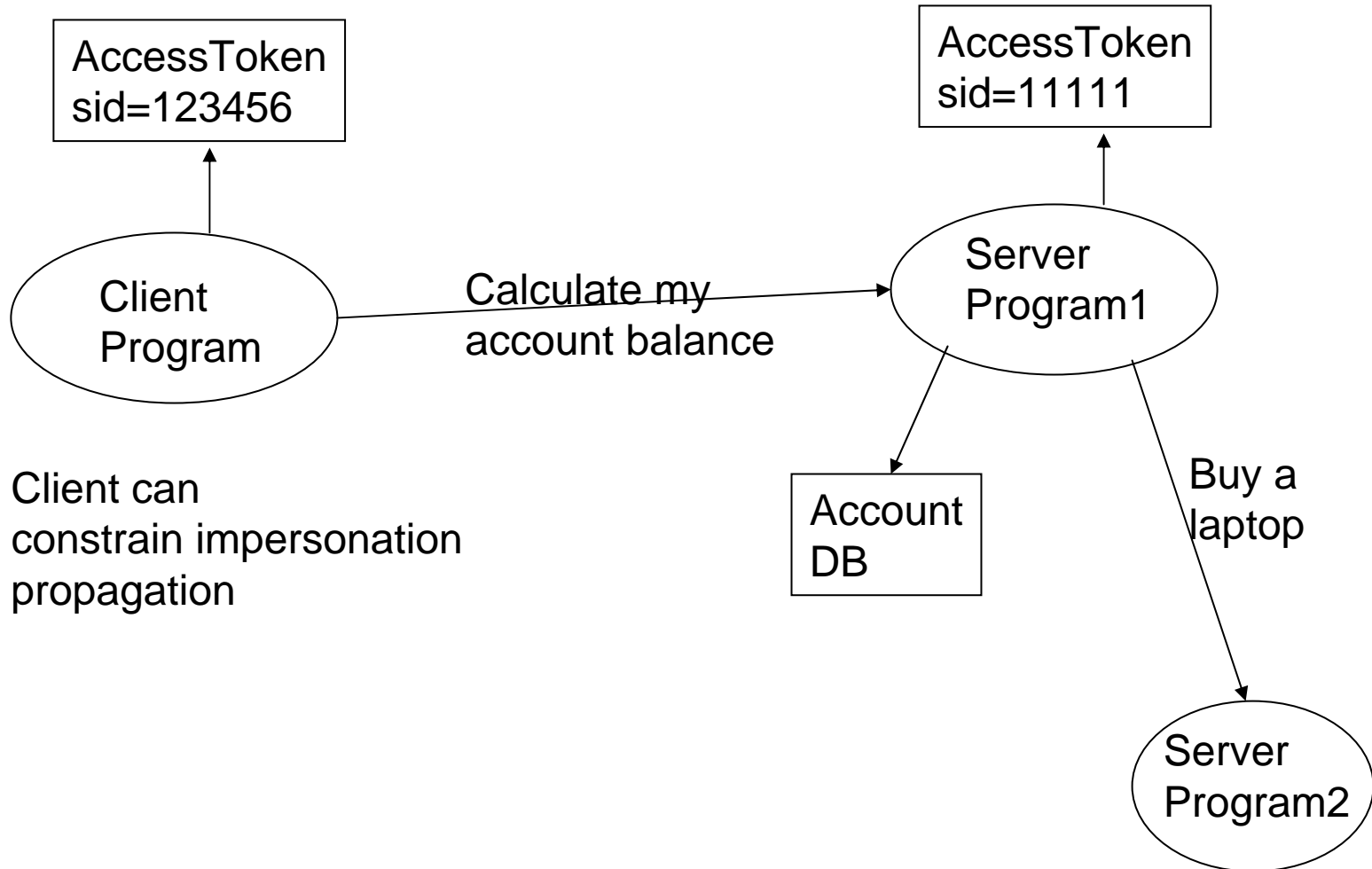
# Solution: Impersonation

- 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
  - **SelfImpersonatePrivilege** 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

# Unix Set UID Problems

- 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
  - Makes porting confusing
- Setuid Demystified paper describes some of the issues

# Implementation Bad Practices

- Do not use relative path names
- Cut and paste/parallel paths
- Don't refer to file by name twice in the same program. Use file handles to make sure all references are to the same entity
- Don't call trusted programs from untrusted programs
- Don't assume success
- Don't invoke shell

# More Bad Practices

- Be careful with pseudo-random numbers
- Don't authenticate on untrusted, spoofable data, like IP address
- Don't store sensitive data in the clear
- Don't made access decisions based on command line or environment variables

# Operational Issues

- Denial of Service attacks
- Bad Passwords
- Unexpected use of default accounts
- Turn off unused services
- Set file system access appropriately
- Monitor
- Keep systems patches up to date
- Understand your installation architecture
  - What is running on the same system?
  - Who has access?
  - Where does the traffic flow?

# Use checklists

- For all phases, checklists can help
  - Check lists are frequently used in safety-critical situations like cockpit management
  - Share awareness across the organization
  - Remind you even if you really know the issue
  - Look at checklist web sites
    - Consider the source though and critically analyze the checklist items
    - E.g., some folks say never embed comments in code because they might get out of date
    - A lot of items are influenced by personal or cultural opinions
  - Should be continually thinking about augmenting and editing the checklist