Vulnerability Assessment and Secure Coding Practices for Middleware

Part 1

James A. Kupsch
Computer Sciences Department
University of Wisconsin

Tutorial Objectives

- Show how to perform the basics of a vulnerability assessment
- Create more people doing vulnerability assessments
- Show how different types of vulnerabilities arise in a system
- Teach coding techniques that can prevent certain types of vulnerabilities
- Make your software more secure

Roadmap

- Part 1: Vulnerability Assessment Process
  - Introduction
  - Evaluation process
  - Architectural analysis
  - Computer process
  - Communication channels
  - Resource analysis
  - Privilege analysis
  - Data Flow Diagrams
  - Component analysis
  - Vulnerability Discovery Activities
- Part 2: Secure Coding Practices

Security Problems Are Real

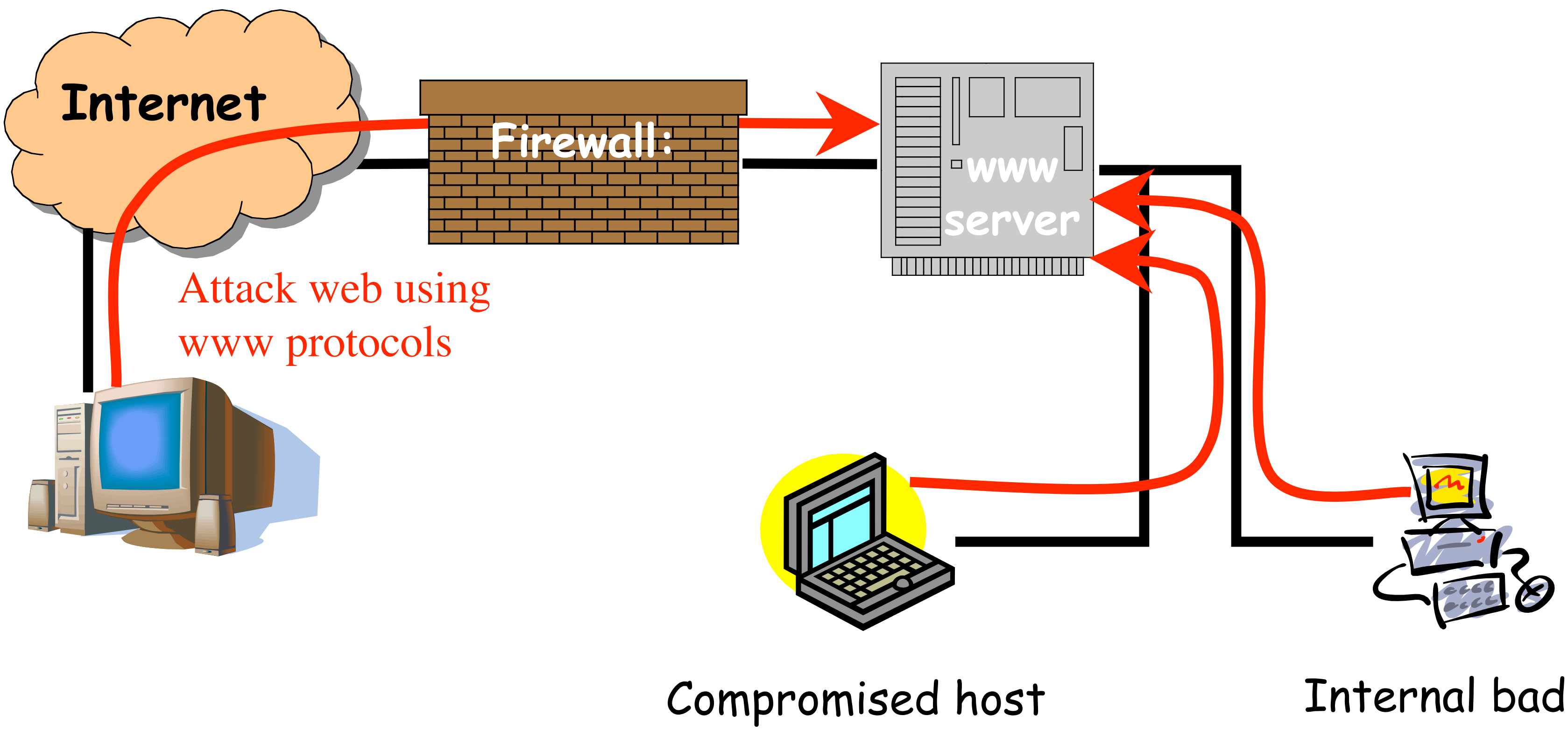
Everyone with a computer knows this.

If you're not seeing vulnerability reports and fixes for a piece of software, it doesn't mean that it is secure. It probably means the opposite; they aren't looking or aren't telling.

The grid community has been largely lucky (security through obscurity).

Many Avenues of Attack

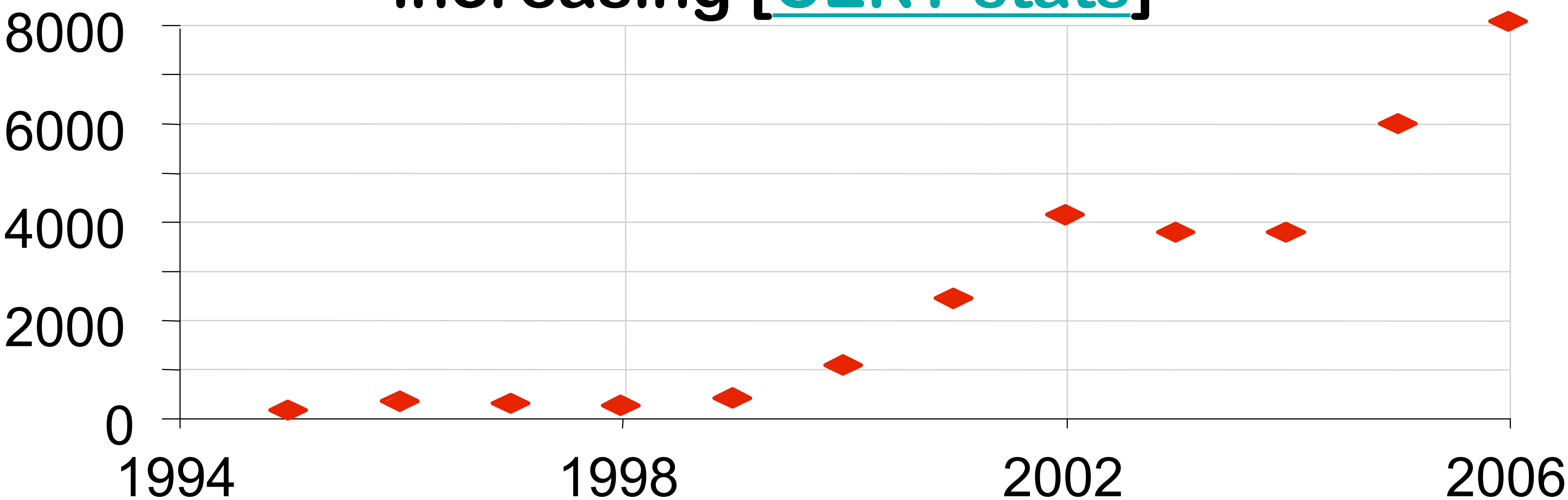
We're looking for attacks that exploit inherent weakness in your system.



Impact of Vulnerabilities

FBI estimates computer security incidents cost U.S. businesses $67 billion in 2005 [CNETnews.com]

Number of reported vulnerabilities each year is increasing [CERT stats]



Security Requires Independent Assessment

Fact #1:

Software engineers have long known that testing groups must be independent of development groups

Fact #2:

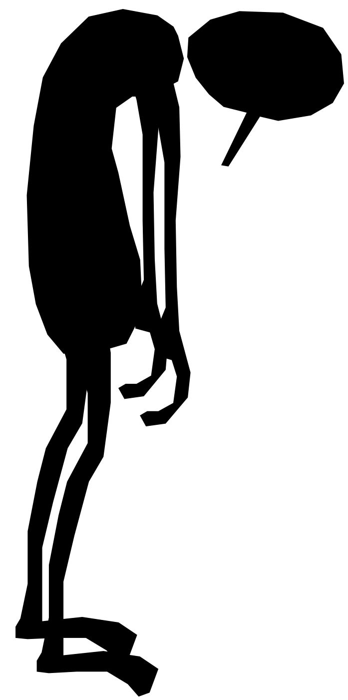
Designing for security and the use of secure practices and standards does not guarantee security

Independent vulnerability assessment is crucial…

…but it's usually not done ☹

Security Requires Independent Assessment (cont.)

- You can have the best design in the world, but can be foiled by …
  - Coding errors
  - Interaction effects
  - Social engineering
  - Operational errors
  - Configuration errors
  - …



THE UNIVERSITY of WISCONSIN MADISON

SDSC

Project Goals

- **Develop** techniques, tools and procedures for vulnerability assessment focusing on Grid software
- **Apply** to production software
- **Improve** the security of this software
- **Educate developers** about best practices in coding and design for security
- **Increase awareness** in the grid and distributed systems community about the need for vulnerability assessment
- **Train** and build a community of security specialists

Systems Investigated

- Univ. of Wisconsin's **Condor Project**
  - Batch queuing workload management system
  - 600K lines of code, began 15 years ago
  - http://www.cs.wisc.edu/condor
- SDSC's **Storage Resource Broker (SRB)**
  - Distributed data store, with metadata and federation capability
  - 275K lines of code, began 9 years ago
  - http://www.sdsc.edu/srb
- NCSA's **Myproxy** (just starting)

Security Evaluation Process

- Architectural analysis
- Resource and privilege analysis
- Component analysis
- Codification of techniques and dissemination
- Overview
  - Insider - full access to source, documents, developers
  - Independent - no agenda, no blinders
  - First principles - let the process guide what to examine

Goal of Vulnerability Analysis

- **Audit a software system looking for security problems**
- **Look for vulnerabilities**
- **Make the software more secure**

"A vulnerability is a defect or weakness in system security procedures, design, implementation, or internal controls that can be exercised and result in a security breach or violation of security policy."

- Gary McGraw, Software Security

i.e., A bad thing

THE UNIVERSITY of WISCONSIN MADISON

SDSC

Attacker Supplied Data

- All attacks ultimately arise from attacker (user) communicated data
- If not, your system is malware
  - The mere installation causes a security violation
- It is important to know where the system can potentially get user supplied data

Get Application Overview

- Goal of architectural, resource and privilege analysis is to learn about the application
- Meet with the developers to get an overview
  - What does application do
  - How does it work
  - What documentation exists
    - End-user
    - Internal design documents
  - What external libraries or environment is needed

Building and Running

- How to obtain source code
- How to build
- How to install and configure
  - What is a typical installation and configuration
- How to control
  - Start
  - Stop
  - Reconfigure
  - Get status

Testing and Debugging

- How to test / What tests exist
- How to debug
  - Any special build options
  - How to control logging
    - What gets logged
    - Where it gets logged
  - Any debugging techniques used in development
- Get access to bug database, find out if there are recurring bugs
- Find out about prior security problems

General Analysis Techniques

- Applies to architectural, resource and privilege analyses
- Find needed information
  - Use existing documentation
    - Often incomplete, out of date, or just wrong
  - Talk to developers
  - Experiment with the system
  - Look at the code - most precise, but most time consuming (later slides will have hints on what to look for)

Analysis By Observing Running Process

- Useful as a starting point
- Will only reveal information about exercised paths in the process
- System monitoring tools
  - `ps` - information about the process
  - `lsof netstat` - information about files/network
  - `ptrace strace dtrace truss` - trace of system calls
  - `ltrace` - trace of library calls
  - `diff tripwire` - can show what objects in the file system were modified

Architectural Analysis

- Create a detailed big picture view of the system
- Document and diagram
  - What executables exist and their function
  - How users interact with them
  - How executables interact with each other
  - What privileges they have
  - What resources they control and access
  - Trust relationships

Hosts in the System

- Each host that software was installed on or external software was configured should be accounted for here
- Types of different hosts in the system
  - Client hosts
  - Server hosts for system executables
  - Hosts running servers used by the system
  - Single host may be in multiple categories
- Classes of hosts: same software running on multiple hosts with only minor configuration differences

Executables in the System

- Find all the executables in the system
  - If install directories are known

    `find` ***installDirs*** `-type f -perm +0111`
  - Look at startup scripts or instructions
- Note high level functionality of each
- If any are not documented ask developers about their function
- Note what executables run on what hosts or classes of hosts

Process Configuration

- How is an executable configured
  - Configuration file
    - Format
    - Other instructions in file such as process another configuration file
    - Search path to find
    - Processing language
  - Hard coded
  - Other
- What can be configured
  - How does it affect the application
  - Often reveals functional and architectural information

Process Attributes

- What user/group is the process started as
- Is the process setuid/setgid
  - `find` *`installDirs`* `-type d -perm +06000`
  - Use `ps` on running process looking for different effective and real ids
- Any unusual process attributes
  - `chroot`
  - Limits set
  - Uses capabilities

Process uid/gid Use

- uid/gid switching
  - For what purpose
  - Must be setuid/getgid or started as root
  - Signs in the code: `setuid setgid seteuid setegid setreuid setregid setresuid setresgid setfsuid setfsgid`
- Is uid/gid sensitive processing done
  - For what purpose
  - Signs in the code: `getlogin cuserid getuid getgid geteuid setegid` / environment variables `LOGNAME USER USERNAME`

External Programs Used

- How are external programs used
- External servers
  - Database
  - Web server
  - Application server
  - Other
- External executables launched
  - Signs in the code: `popen system exec*`
  - What executables

User Interaction with System

- How do users interact with the system
  - Client executables
  - API
- What type of interaction can they have
- What data do the inject into the system

Process Communication Channels

- What exists between…
  - Servers
  - Client and server
  - Server and external programs
    - DBMS
    - Network services
      - DNS
      - LDAP
      - Kerberos
      - File services: NFS AFS ftp http …
- Shows interaction between components

Communication Methods

- OS provides a large variety of communication methods
  - Command line
  - Files
  - Creating processes
  - IPC
    - Pipes
    - FIFO's or named pipes
    - System V IPC
    - Memory mapped files
  - Environment
  - Sockets
  - Signals
  - Directories
  - Symbolic links

Command Line

- Null-terminated array of strings passed to a starting process from its parent
- Convention is that `argv[0]` is the path to executable file
- Signs in code
  - C/C++: `argc argv`
  - Perl: `@ARGV $0`
  - Sh: `$0 $1 $2… $# $@ $*`
  - Csh: `$0 argv`

Environment

- Null-terminate array of string passed to a process from its parent
- Convention is that each string is of the form `key=value`, and `key` can not contain an equal sign
- Program can change environment
- Contents can affect program behavior
- Inherited by children
- Signs in code:
  - C/C++: `environ getenv setenv putenv`
  - Perl: `@ENV`
  - bash/csh: not easy to tell uses

Files

- Represented by a path to a file in the file system
- Can be created or opened, or inherited from parent process
- Contents can be data, configuration, executable code, library code, scripts
- Signs in code:
  - C/C++: `open creat fopen`

Standard File Descriptors

- Convention is creating process opens file descriptors 0, 1 and 2 for use by the created process to be used as standard in, out, and err
- Functions and libraries often implicitly use these and expect them to be opened
- Signs in code
  - C/C++: `stdin stdout stderr STDIN_FILENO STDOUT_FILENO STDERR_FILENO getchar gets scanf printf vprintf vscanf cin cout cerr`

Sockets

- Allows creating a communication path
  - local to the system
  - between hosts using protocols such as TCP/IP
- Can be stream or message based
- Signs in code
  - C/C++: `socket bind connect listen accept socketpair send sendto sendmsg recv recvfrom recvmsg getpeername getsockname setsockopt getsockopt shutdown`
  - Bash: `/dev/tcp/`*`host`*`/`*`port`*
    `/dev/udp/`*`host`*`/`*`port`*

Creating a Process

- When a process is created man properties of the original are inherited such as
  - User and group ids
  - File descriptors without close-on-exec
  - Current and root directories
  - Process limits
  - Memory contents
- Exit status communicated back to parent
- Signs in code
  - C/C++: `fork popen system exec* exit _exit wait waitpid wait3 wait4`
  - Perl: `` open system qx ` exit _exit wait waitpid wait3 wait4 ``

Signals

- Asynchronous notification to a process generated from the operating system, run-time events or sent from related processes
- Essentially a 1 bit message
- Signs in code
  - C/C++: `kill raise signal sigvec sigaction sigsuspend abort`

IPC

- Intra-host communication methods
- Some can pass file descriptors between processes
- Signs in code:
  - Pipes: `pipe`
  - SysV Message Q: `msgget msgctl msgsnd msgrcv`
  - SysV Semaphore: `semget shmctl semop`
  - SysV Shared Mem: `shmget shmctl shmat shmdt`
  - Memory mapped files: `mmap`

Directories

- Directories contain a list of file system objects such as files and directories
- Directory can be read to get list of names or updated by creating, renaming or deleting existing entries
- Signs in code:
  - C/C++: `opendir` `readdir` `closedir` `creat` `open`(with `O_CREATE`) `fdopen` `mkdir` `mkfifo` `mknod` `symlink` `link` `unlink` `remove` `rename` `rmdir`

Symbolic Links

- Symbolic links are an entry in a directory that contain a path (referent)
- When evaluating a path the operating system follows the referent in the link
- Referent can be read and used by a program
- Signs in code:
  - C/C++: any function taking a path, `symlink` `readlink`

Messaging & File Formats

- Document messaging protocols
  - This is really an API between executables
  - What is the format and purpose of each message
  - How are message boundaries and individual pieces of data determined
- Document file formats
  - Same thing as for messages
  - You can think of files as persistant asynchronous messages

Libraries Used

- What libraries does the executable use
  - Run `ldd` on executable
  - Look at link command line
  - Look for uses of `dlopen` in the code
- Need to check it for vulnerabilities and for safe use
  - Audit the library
  - Rely on reports from others

Resource Analysis

- A resource is an object that is useful to a user of the system and is controlled by the system
  - Physical things
    - Disk space
    - CPU cycles
    - Network bandwidth
    - Attached devices
  - Data

Documenting Resources

- What resources exist in the system
- What executables/hosts control the resource
- What operations are allowed
- What privileges are required
- What does an attacker gaining access to the resource imply

Privilege Analysis

- Privilege is the authorization for a user to perform an operation on a resource
- Role is a set of privileges assigned to multiple users to create types of user such as admin
- How is authentication performed, if an attacker can authenticate as another user they gain their privileges

Privileges in the System

- What privileges exist in the system
- Do they map appropriately to operations on resources
- Are they fine grained enough
- How are they enforced

Interactions with OS privileges

- What OS user/group account are used and what is their purpose
- Does the system use the operating system to enforce its privilege model
- File system privileges can be used to enforce files being read or written by attackers
- If process is run as root it can change privilege to an OS user to restrict privileges to that user

External Server Privileges

- DMBS
  - How is authentication performed
    - How is password stored
  - DBMS accounts used
    - privilege granted to each
  - Are the privileges granted the minimum necessary
- Other external servers
  - Privilege model
  - Interaction with internal

Trust

- An executable trusts another when
  - It relies on a behavior in the other
  - Doesn't or can't verify the behavior
- Implicit trust
  - The operating system
  - Process with root privilege on the same host
    - they can do anything
  - Processes with same uid on the same host
    - they can do anything to each other
  - All the code in your executable including libraries

Bad trust

- Not validating data from another trust domain for proper form (form, length, range)
- Bad assumptions
  - User entered data in proper form
  - Data passed to client is returned unchanged
    - Need a cryptographic signature
    - Happens with hidden input field and cookies in HTML

More Bad Trust

- Bad assumptions
  - Client validated data
    - Client can be rewritten or replaced
    - Good to validate on the client, but server validation is required
- Best to validate data even from trusted executables as it provides security in depth
  - One server could be used as a conduit for an attack

Use/Abuse Cases

- Use cases
  - Document typical use scenarios for the software
  - Often times created by testing team
- Abuse cases
  - Anti-use case, what an attack might do to break the system
- Both will reveal the architecture and potential security problems

Data Flow Diagrams

- Takes information from previous analyses
- Turns a use/abuse case into a diagram showing
  - Hosts
  - Components such as processes
  - Privileges
  - Message flows
  - Steps in the case

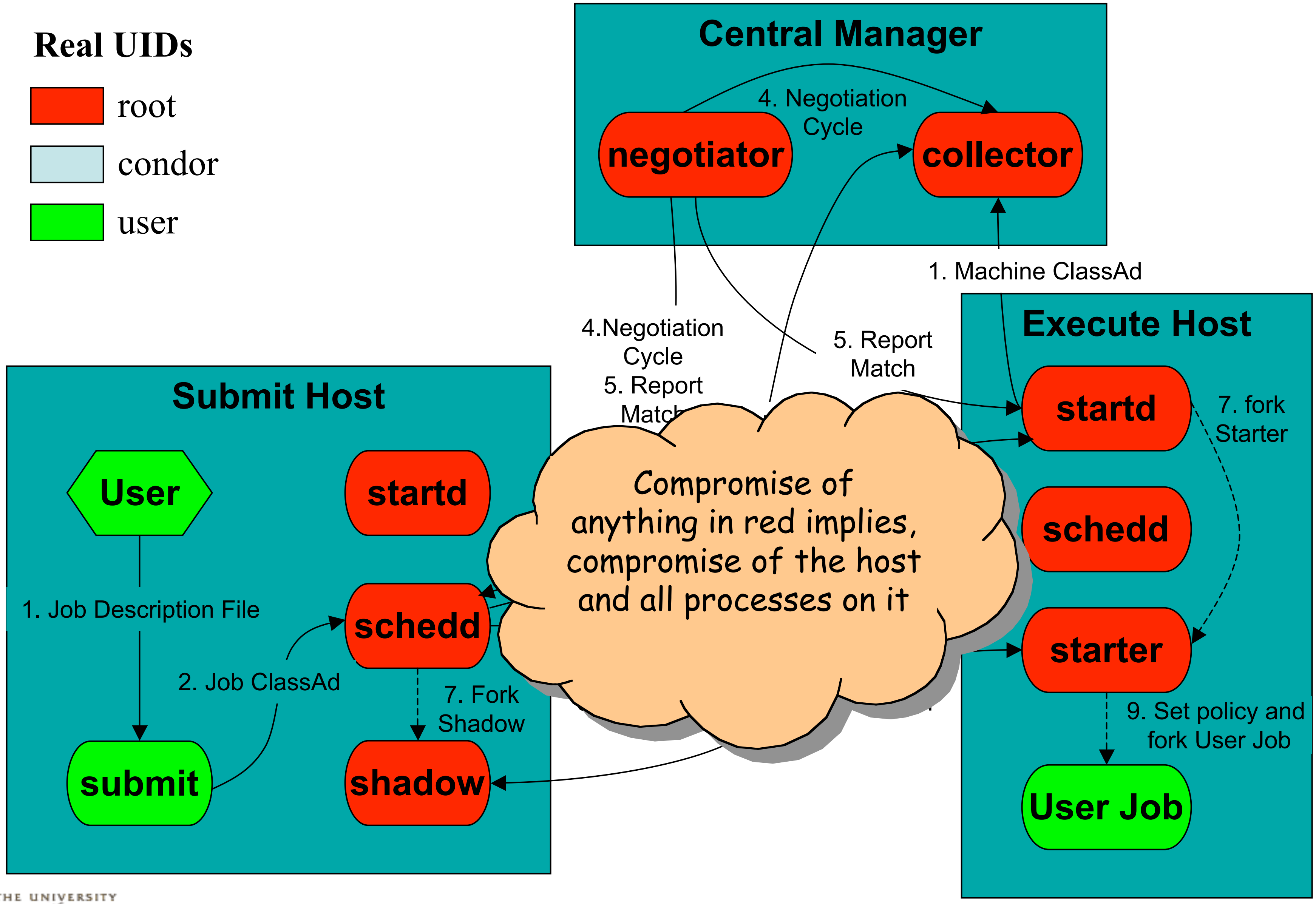
Data Flow Diagrams

- Colors represent privilege
- Hosts are represented by rectangles
- Processes by circles
- Communication flow by lines with arrows indicating direction of flow
  - Labels indicate contents of message or operation
- Other symbols can be used for other important objects in the case such as files
- We've noted that developers often learn things when presented with just these diagrams

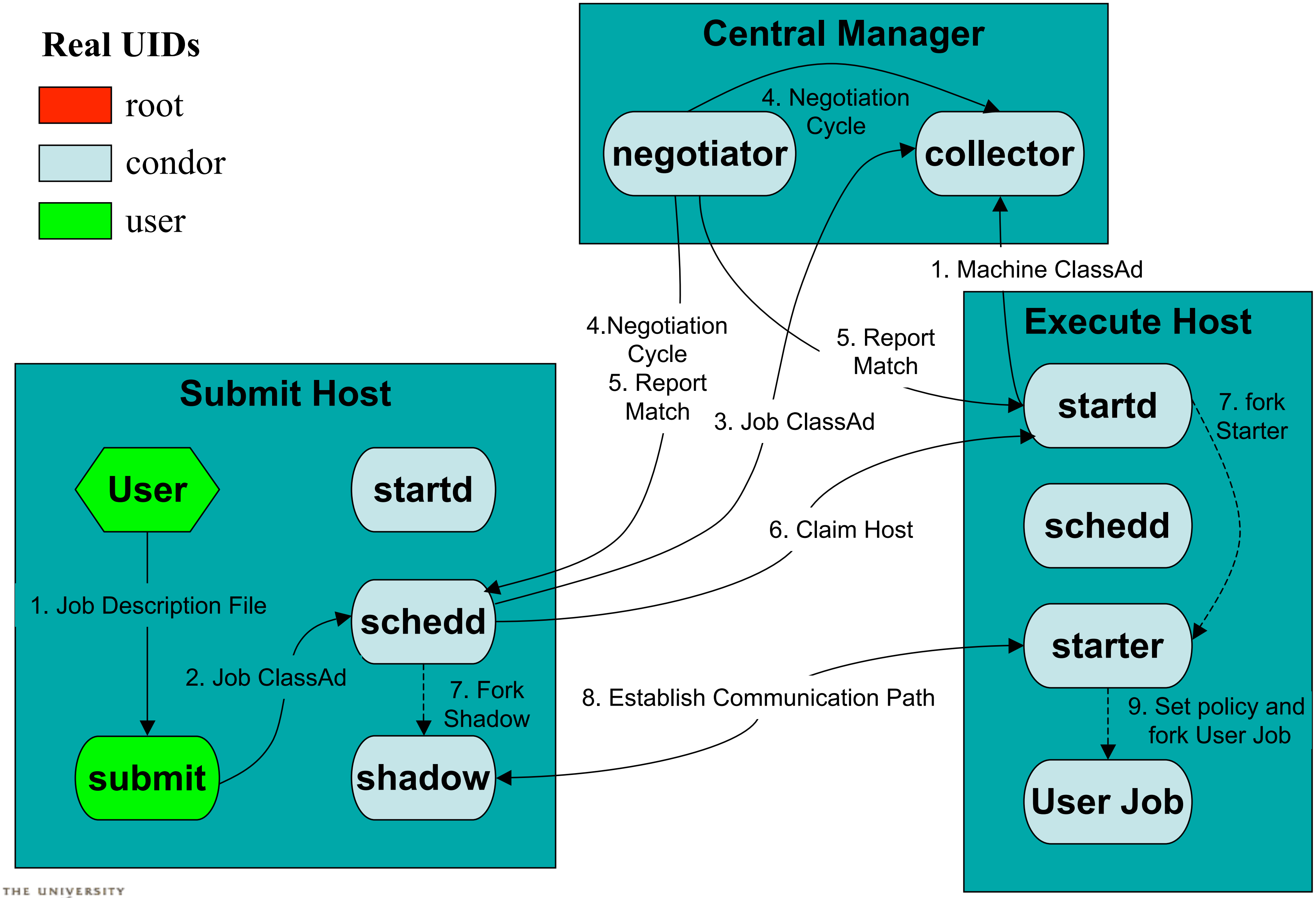
Privileges - Root Install

Real UIDs

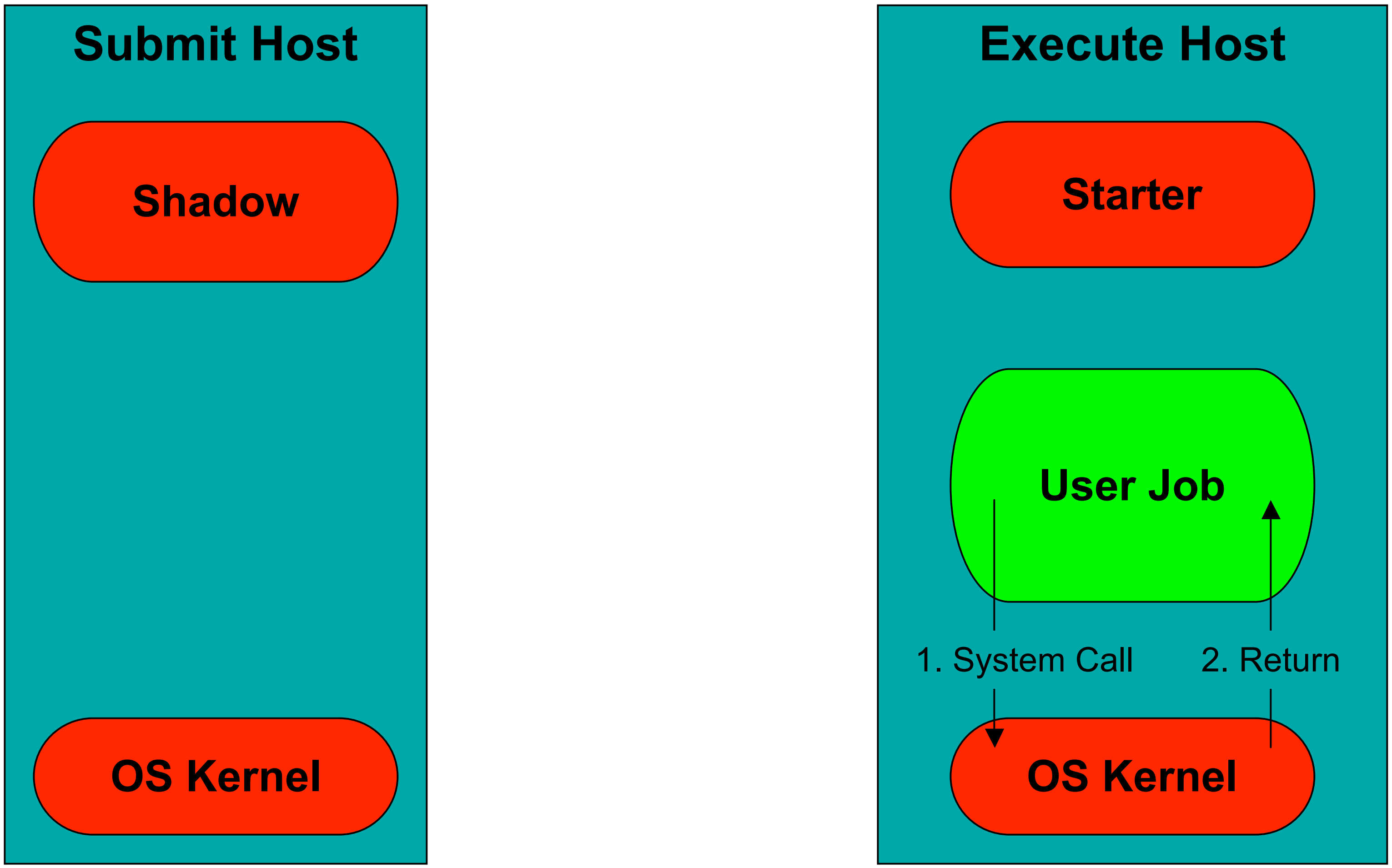
- root
- condor
- user



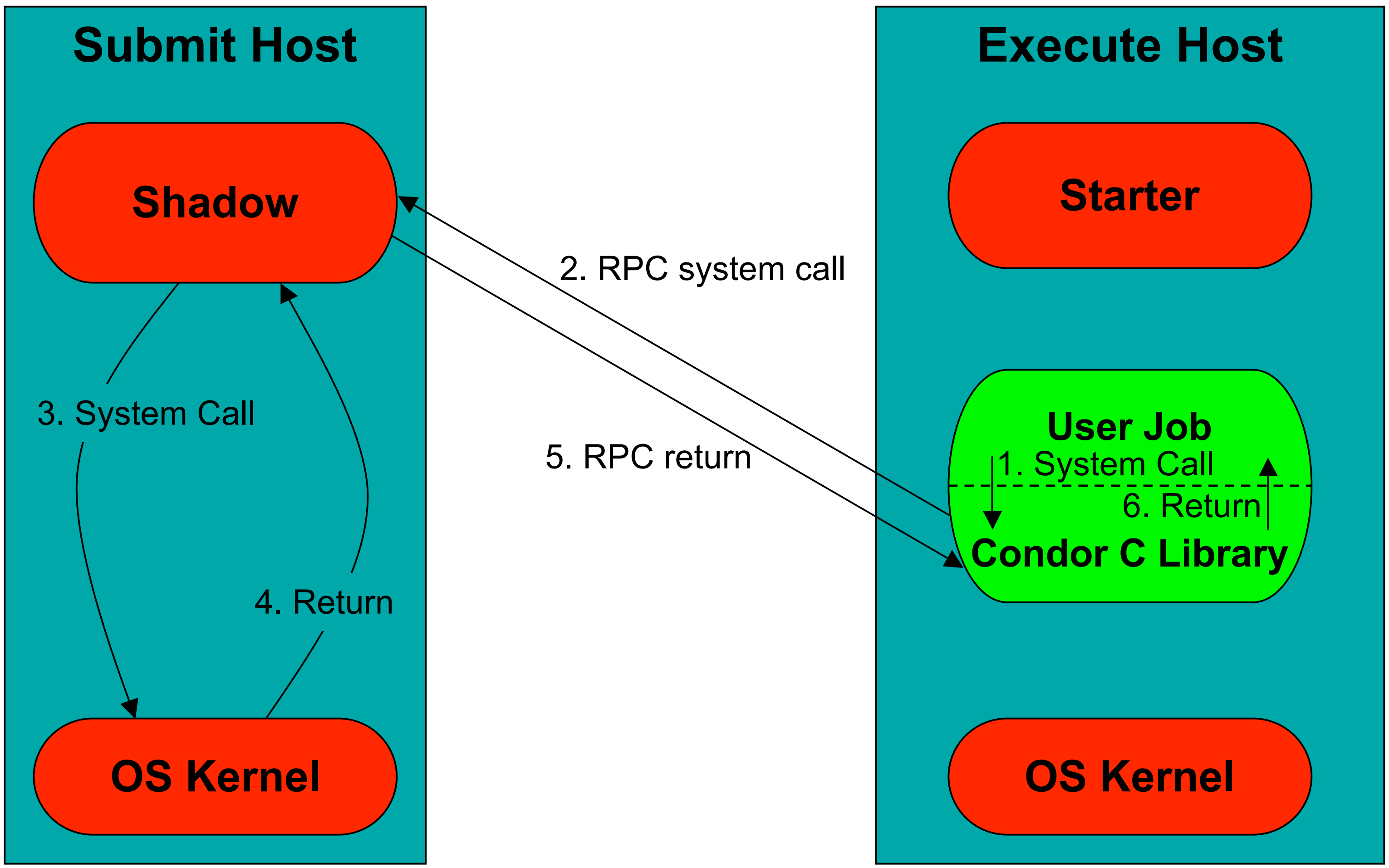
Privileges - Non-Root Install



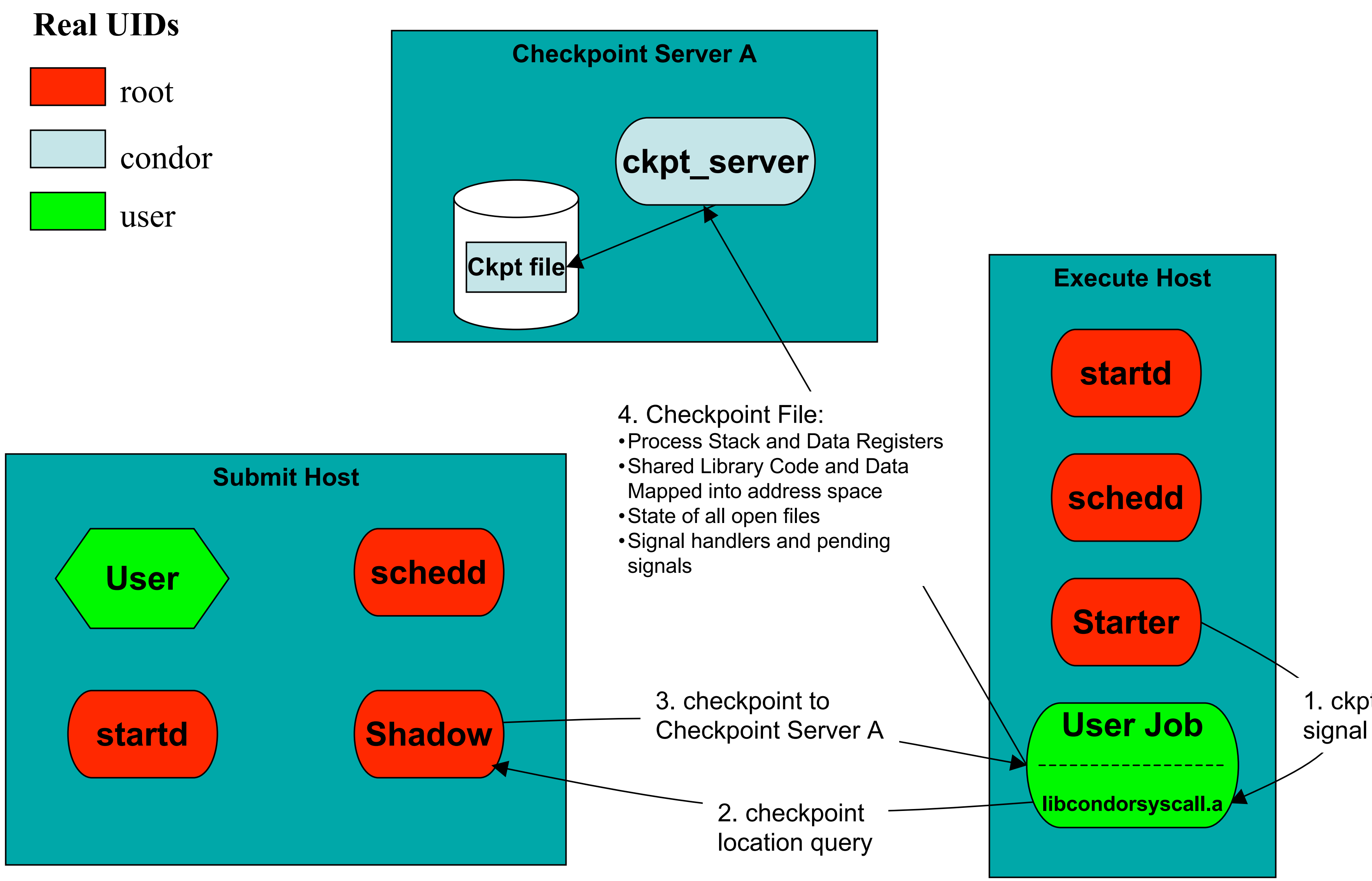
Vanilla Universe Execution



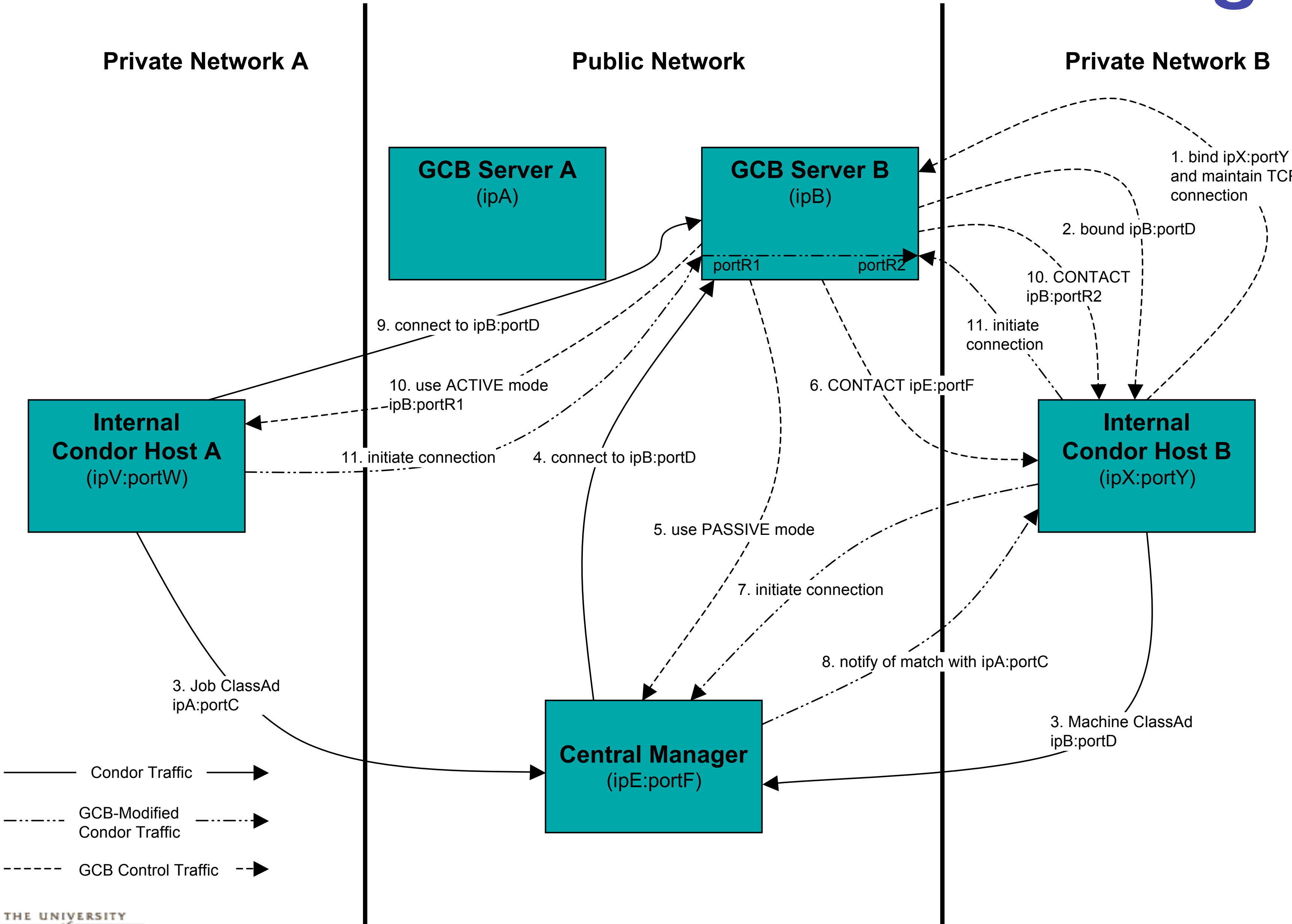
Standard Universe Execution



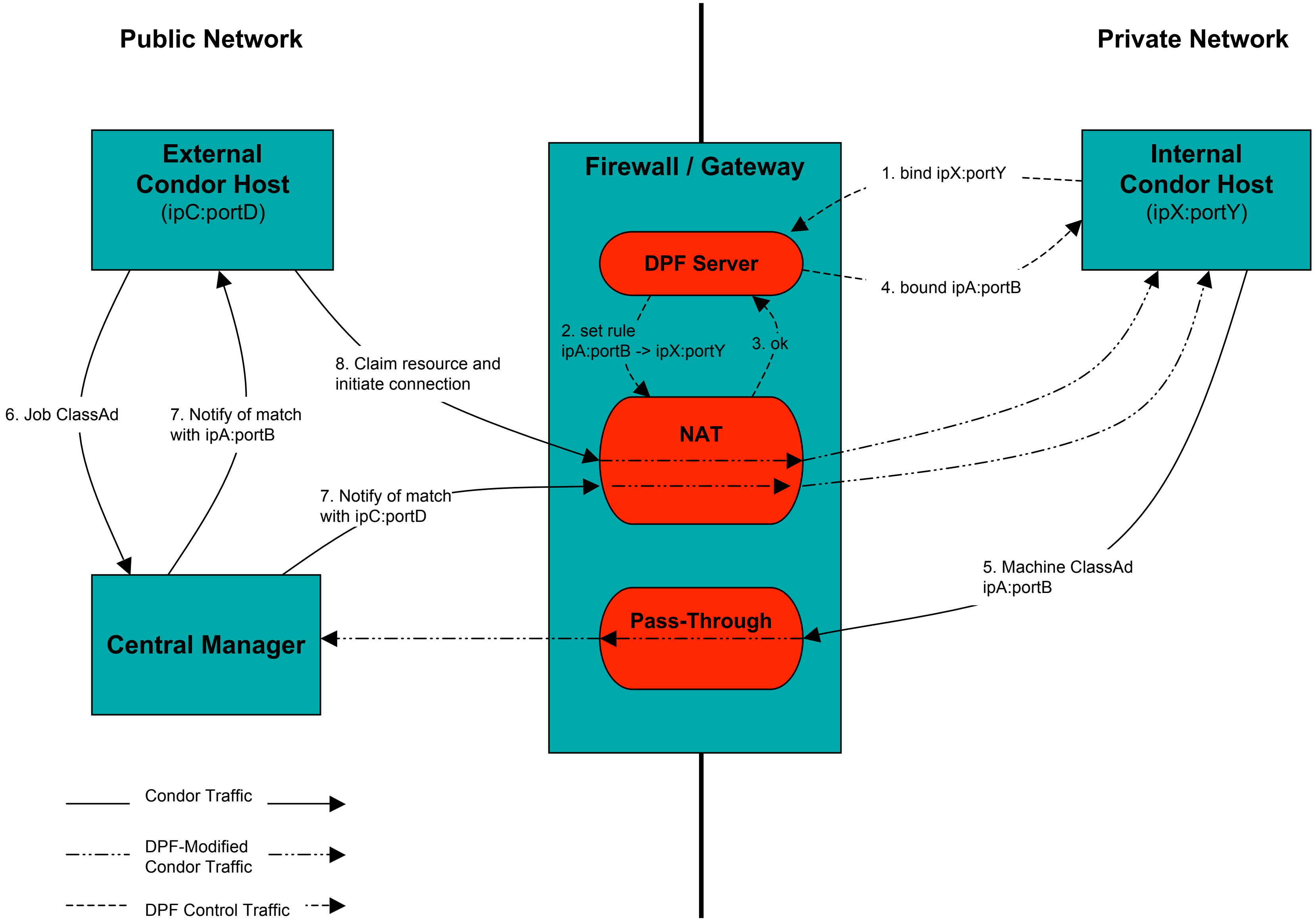
Checkpointing a Job



Generic Connection Brokering



Dynamic Port Forwarding



Drilling In / Drilling Out

- Drill in to focus on sub systems that are more likely to be vulnerable and lead to large security failures …
  - Deal with security
  - Control resources
  - Validate input
- Drill out to analyze how this system interact with others
  - Systems can be secure, but insecure when combined

Component Analysis

- **Audit the source code** of a component…
  … the audit is directed by earlier analyses
- **Look for vulnerabilities** in a component
- Need to connect user input to a place in the code where a vulnerability can be triggered by it
- Finds deeper problems than black box testing
  - Penetration testing
  - Fuzzing

Categories of Vulnerabilities

- **Design Flaws**
  - Problems inherent in the design
  - Hard to automate discovery
- **Implementation Bugs**
  - Improper use of the programming language, or of a library API
  - Localized in the code
- **Operational vulnerabilities**
  - Configuration or environment
- **Social Engineering**
  - Valid users tricked into attacking

Occur about equally

Many Types of Vulnerabilities

- Buffer overflows
- Injection attacks
  - Command injection (in a shell)
  - Format string attacks (in printf/scanf)
  - SQL injection
  - Cross-site scripting or XSS (in HTML)
- Directory traversal
- Integer vulnerabilities
- Race conditions
- Not properly dropping privilege
- Insecure permissions
- Denial of service
- Information leaks
- Lack of integrity checks
- Lack of authentication
- Lack of authorization

Focusing the Search

- It's impossible to completely analyze a system for vulnerabilities
- From places where vulnerabilities can occur in the code
- From the point of view of an attacker's goal and try to think of ways the threat can be realized
- If there were prior security problems look for similar problems
- Focus on subsystem that are one of
  - Important
  - Security related
  - Poorly written
  - Poorly tested (little used)
  - Developer/Testing functionality

Difficulties

- Need to trace function call graphs to trace data flows to determine potential values
- It is difficult in C++ to determine function call graphs using a textual analysis, due to the ambiguity of identifiers; the name alone is insufficient to determine the actual function
- The use of function pointers also complicate this analysis

Code Browsing Tools

- cscope
  - Doesn't understand C++
- ctags
  - Useful for finding definitions of global variables and functions, but not uses
- eclipse
  - Doesn't handle size of code and style well
- Hand written perl scripts to search code
  - Useful, but really need to parse C/C++

Static Code Analysis Tools

- Require a human to analyze the results for false positives
- Won't find complex problems
- They aid the assessor, but they're not one-click security
- Commercial analyzers
  - Coverity http://www.coverity.com
  - Fortify http://www.fortifysoftware.com
  - Secure Software http://www.securesoftware.com
  - Grammatech http://www.grammatech.com
- Freely available analyzers
  - Flawfinder http://www.dwheeler.com/flawfinder
  - RATS (Rough Auditing Tool for Security) http://www.securesoftware.com/rats
  - ITS4 http://www.citigal.com/its4
- Compiler warnings

Vulnerability Report

- One report per vulnerability
- Provide enough information for developers to reproduce and suggest mitigations
- Written so that a few sections can be removed and the abstracted report is still useful to users without revealing too much information to easily create an attack.

Condor Vulnerability Report

CONDOR-2005-0003

Summary:

Arbitrary commands can be executed with the permissions of the condor_shadow or condor_gridmanager's effective uid (normally the "condor" user). This can result in a compromise of the condor configuration files, log files, and other files owned by the "condor" user. This may also aid in attacks on other accounts.

Component	Vulnerable Versions	Platform	Availability	Fix Available
condor_shadow condor_gridmanager	6.6 - 6.6.10 6.7 - 6.7.17	all	not known to be publicly available	6.6.11 - 6.7.18 -
Status	**Access Required**	**Host Type Required**	**Effort Required**	**Impact/Consequences**
Verified	local ordinary user with a Condor authorization	submission host	low	high
Fixed Date	**Credit**			
2006-Mar-27	Jim Kupsch			

Access Required: local ordinary user with a Condor authorization

This vulnerability requires local access on a machine that is running a condor_schedd, to which the user can use condor_submit to submit a job.

Effort Required: low

To exploit this vulnerability requires only the submission of a Condor job with an invalid entry.

Impact/Consequences: high

Usually the condor_shadow and condor_gridmanager are configured to run as the "condor" user, and this vulnerability allows an attacker to execute arbitrary code as the "condor" user.

Depending on the configuration, additional more serious attacks may be possible. If the configuration files for the condor_master are writable by condor and the condor_master is run with root privileges, then root access can be gained. If the condor binaries are owned by the "condor" user, these executables could be replaced and when restarted, arbitrary code could be executed as the "condor" user. This would also allow root access as most condor daemons are started with an effective uid of root.

Vulnerability Report Items

- Summary
- Affected version(s) and platform
- Fixed version(s)
- Availability - is it known or being exploited
- Access required - what type of access does an attacker require: local/remote host? Authenticated? Special privileges?
- Effort required (low/med/high) - what type of skill and what is the probability of success

Vulnerability Report Items

- Impact/Consequences (low/med/high) - how does it affect the system: minor information leak is low, gaining root access on the host is high
- Full details - full description of vulnerability and how to exploit it
- Cause - root problem that allows it
- Proposed fix - proposal to eliminate problem
- Actual fix - how it was fixed

Vulnerability Disclosure Process

- Disclose vulnerability reports to developers
- Allow developers to mitigate problems in a release

Now here's the really hard part:

- Publish abstract disclosures in cooperation with developers. When?
- Publish full disclosures in cooperation with developers. When?

When a Vulnerability Is Found

- **Don't Panic!!!** Have a plan.
- Plan **what, how, when and to whom to announce**
- Plan **how to fix**, and **what versions**
- Separate security release or combine with other changes?
- When to release full details
  - are details known or being exploited externally
  - open/closed source projects
  - allow time for users to upgrade