

CS392: Computer Security

CS6813: Internet Security and Privacy

Midterm 1 and Beyond!

[*] Slides based upon materials maintained by Justin
Cappos at NYU

About assignment 1.1-1.3, Quiz 1

A few students lost points for not following directions
Essential to follow directions on 2.1-2.3!!!

The “Quiz” was the locks exercise we did in class. We will have more quizzes like this and they will not be announced.

If you want to raise your grade, do an extra credit

Students have had a 0 on an assignment and still get an A

If you have questions, feel free to ask me in office hours

What is to come...

First part of the class is done!

No-Tech / Low Tech

Next up is:

Access Control

Capability Systems

Information Flow Control

Fault Isolation

Performance Isolation

Still to come:

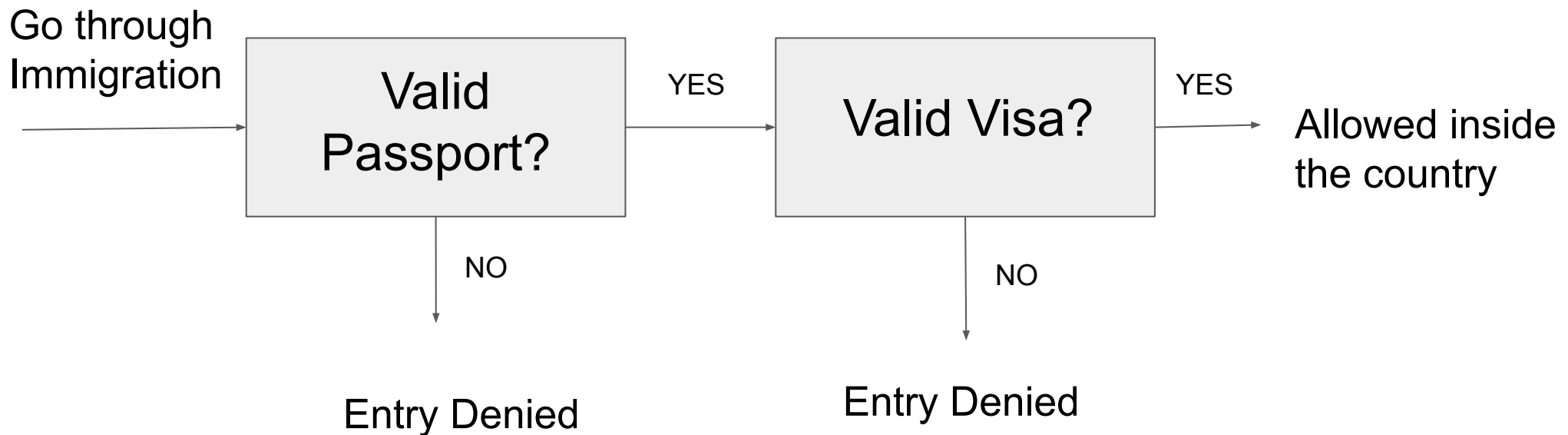
Advanced topics!

Access Control - Definition

Access control is a series of mechanisms used by management, to specify what users can do, which resources they can access, and what operations they can perform on a system. More generally, it permits managers of a system to direct or restrain the behavior, use and content of a system.

Access Control - Example

When you go through immigration at the airport at a foreign country...

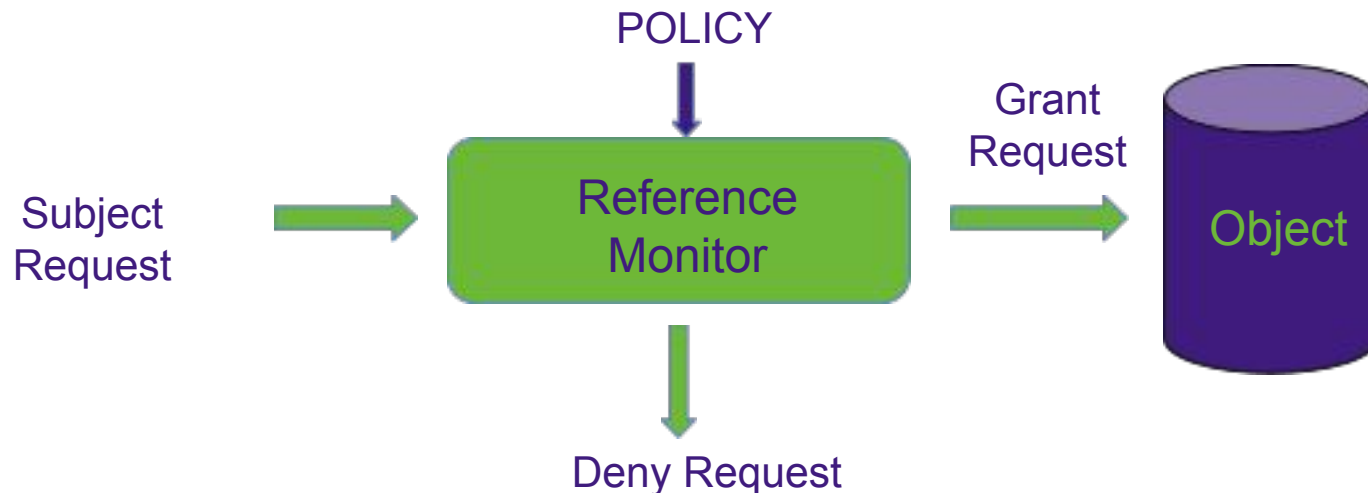


Do you notice the layers?

Custom Reference Monitors

[Anderson 72]

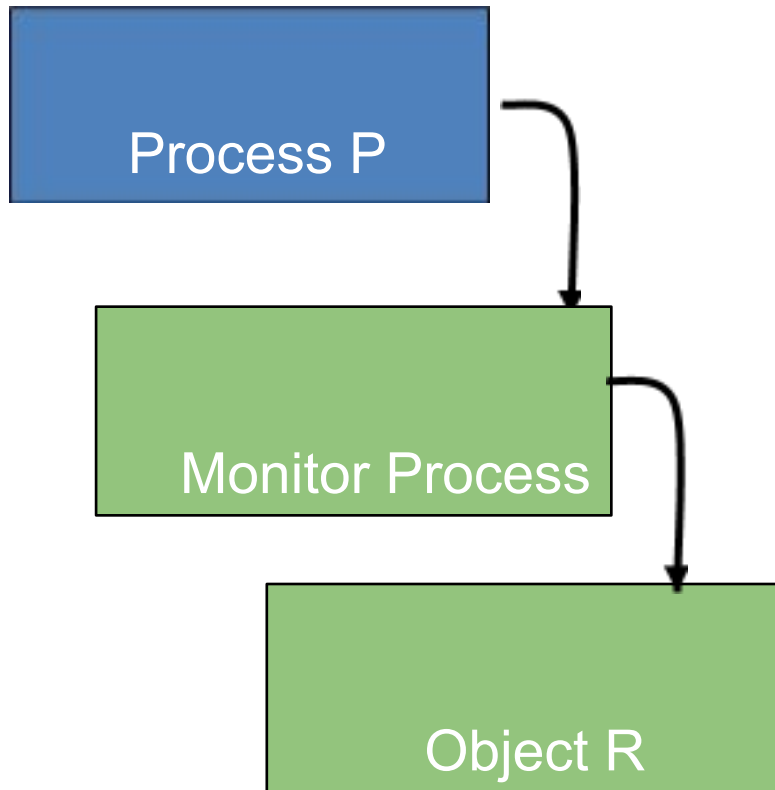
- Assumptions:
 - System knows who the user is
 - Authentication via credentials
 - Access requests pass through the gatekeeper, aka, **reference monitor**
 - System must not allow monitor to be bypassed



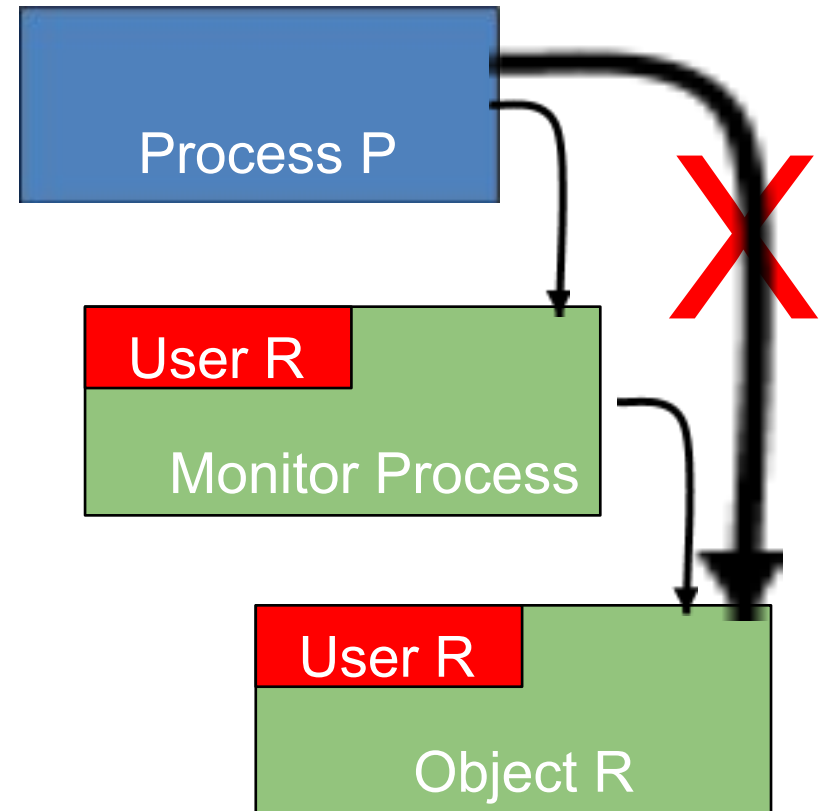
- Is it easier to build a custom reference monitor when objects (access-control list) or subjects (capability-based systems) contain permissions?

Custom Reference Monitor

Capability Model



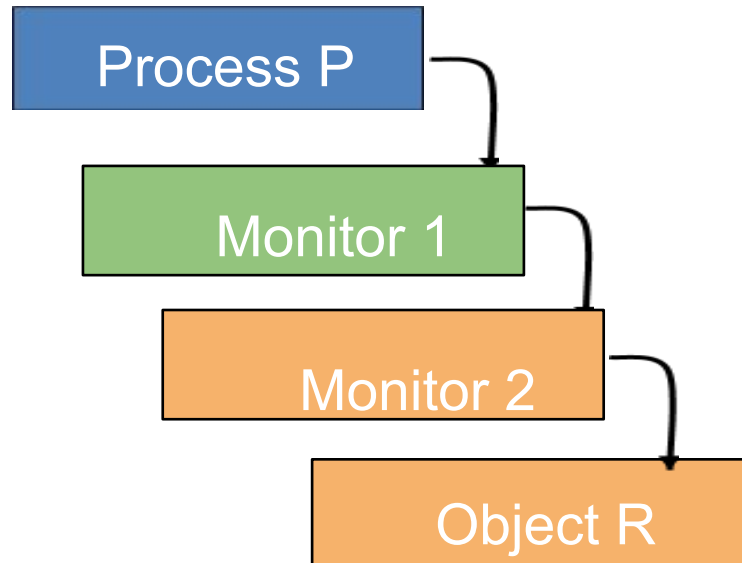
ACL Model



Custom Reference Monitor (Cap)

To implement a capability system:

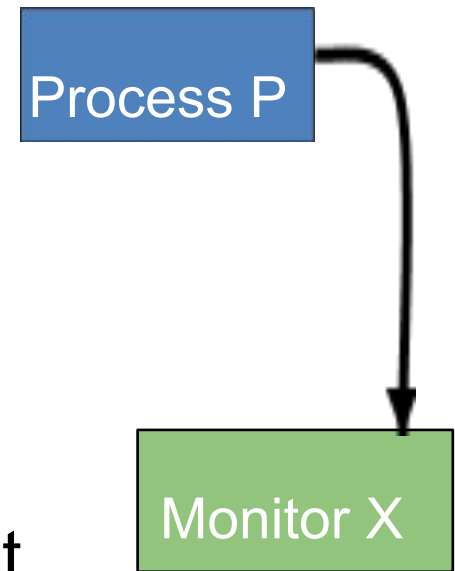
- Get the capability to what you want to mediate
 - Need to ensure that restricted processes do not have a copy of this capability
- Create a new capability for calling your reference monitor
- Pass this capability in place of the original capability
- Can be repeated for multiple reference monitors!



Custom Reference Monitor (ACL)

System call interposition:

- Process X performs a system call to trace a process P
- The kernel adds a hook at P's system call entry to trap calls
- When P makes a "sensitive" call, X is notified
 - "sensitive" is defined by the kernel
- P is suspended (either by X or the kernel)
- X checks the call arguments, etc.
 - If a permitted call, allows the kernel to issue it
 - If denied, abort the system call
 - This causes problems for applications...



Custom Reference Monitor (ACL)

[Garfinkel 2003]

System call interposition problems:

- Hooks require lots of kernel code to do things right
- Reference monitor must correctly replicate OS state / code
 - Code is tricky (dup2, rel path / links / chroot)
- Easy to overlook some path
 - Most things in Unix / Linux are files (sockets, devices)
 - We also have descriptor passing
- Race conditions (time-of-check-to-time-of-use)
 - Symbolic link race conditions
 - Relative path races
 - Argument replacement
 - Writable multi-process shared memory or threads

Custom Reference Monitor (ACL)

[Garfinkel 2003]

System call interposition problems (cont):

- Subsetting an interface is hard
 - Attempt 1: Deny the creation of symlinks to files such that we do not allow unrestricted access.
 - Should prevent the impact of a race, right?
 - What about preexisting "unsafe" symlinks?
 - Attempt 2: Deny the creation and renaming of symlinks.
 - Protects against previous attacks
 - Directories may contain symlinks
 - Directories can be renamed
 - Attempt 3: Deny access through symlinks.
 - What if the path to a file contains a symlink?
 - This is hard to check / prevent

Custom Reference Monitor (ACL)

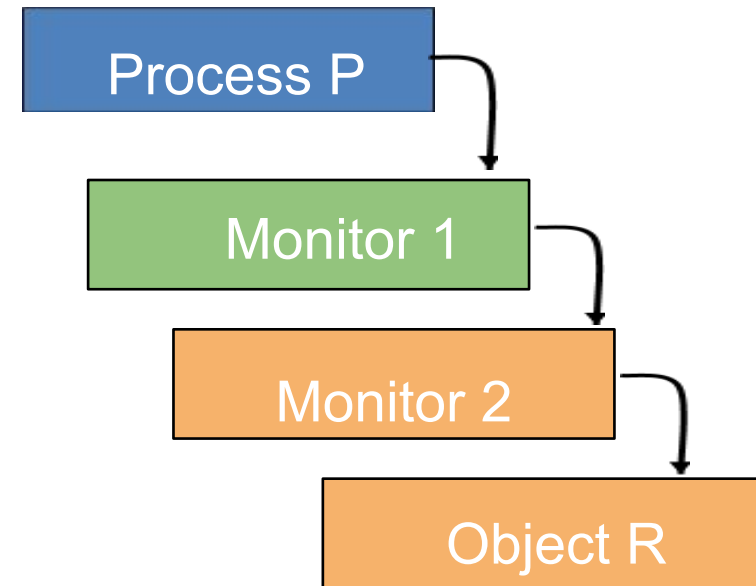
System call interposition problems (cont):

- Very tricky to implement correctly in practice
- In most cases:
 - Cannot chain reference monitors
 - Not recommended for security
 - Breaks applications unnecessarily
- Isn't widely used except for debugging
 - As a result, "custom" code ends up in kernel, etc.

Custom Reference Monitor Summary

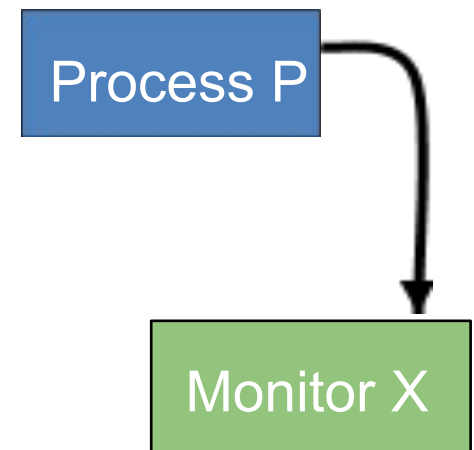
Capability systems:

- Very natural and straightforward
 - The monitor hides the current capability behind its checks and exposes a "safe" version
 - Easy to chain multiple together



Access Control List systems:

- Hard to do custom reference monitors
 - Difficult to implement correctly
 - Lots of corner cases to handle
 - Cannot chain multiple together
 - Not popular in practice



Programming Language VM

Core idea:

Run code within a special process that checks the safety / validity.

Technique used often by programming languages

- Java, Lua, Flash, JavaScript, Seattle / Remy, etc.
- Often use 'byte code' instead of source code
 - Intermediate representation.
- Any bug in the PL VM is usually fatal.

Trusted Code in Sandboxes

Computation

- Type checking
- Public, private, etc.
- Object creation, garbage collection, etc.

Standard libraries

- File I/O, networking, reflection, etc.
- Subject to **policy restrictions**
 - Configurable policies

Trusted Code in Sandboxes

Computation

- Type checking
- Public, private
- Object creation, garbage collection, etc.

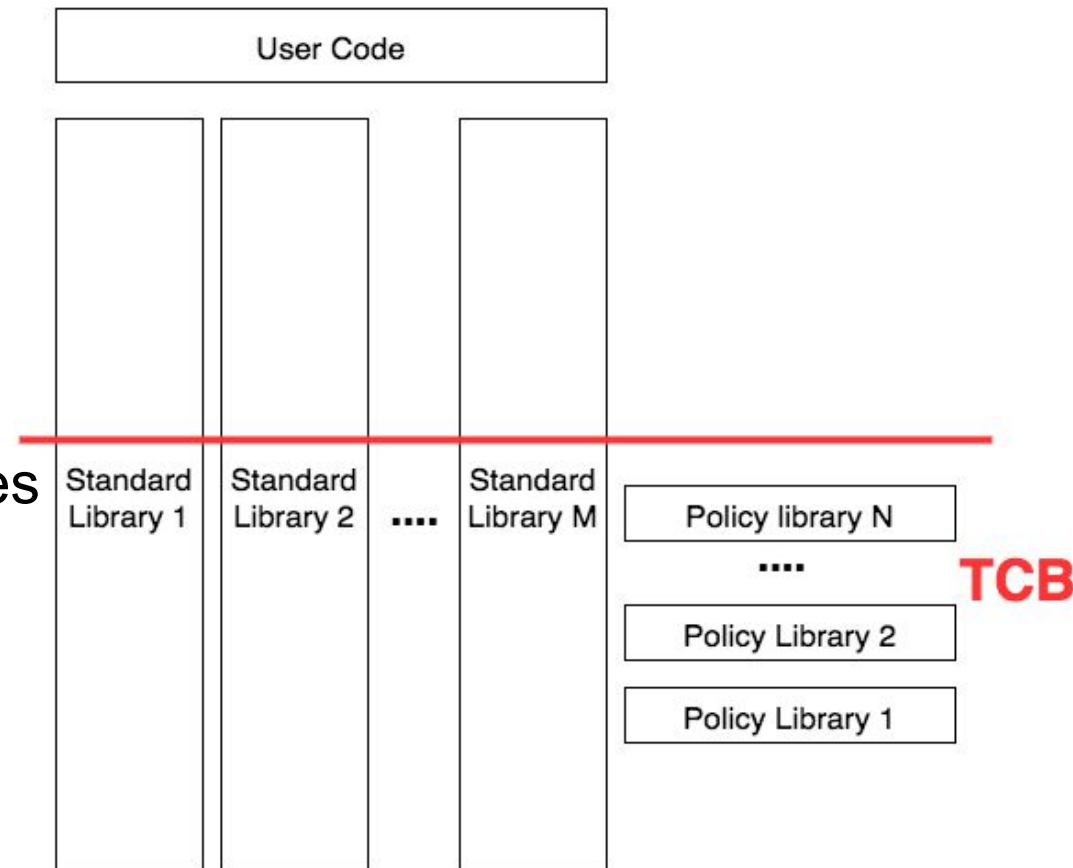
59 security critical bugs
in ~1% of code [Tan 08]

Standard libraries

- File I/O, networking, reflection, etc.
- Subject to **policy restrictions**
 - Configurable policies

Traditional Library Design

- Large, single Trusted Computing Base (TCB)
 - Any failure is fatal
 - Privilege is widely used
 - Java's Calendar library uses privileges when deserializing
- Isolation only in the TCB
 - Must contain policy



Traditional Policy Interposition

Functionality is scattered




Traditional Policy Interposition

Functionality is scattered



File Access Checker



Prevent
access
to certain
files

Traditional Policy Interposition

Functionality is scattered



The diagram illustrates the concept of traditional policy interposition. It features three main components: a yellow rectangular box labeled 'File Access Checker', a red square box labeled 'open()', and a yellow scroll-shaped box labeled 'Prevent access to certain files'. The 'File Access Checker' box is positioned above the 'open()' box. The 'Prevent access to certain files' scroll is positioned to the right of the 'File Access Checker' box. The text 'Functionality is scattered' is located above the 'File Access Checker' box.

File Access Checker

open()

Prevent
access
to certain
files

Traditional Policy Interposition

Functionality is scattered

File Access Checker

import

open()

Prevent
access
to certain
files

Traditional Policy Interposition

Functionality is scattered

File Access Checker

import

open()

DB

Prevent
access
to certain
files

Traditional Policy Interposition

Functionality is scattered

File Access Checker

Prevent
access
to certain
files

import

open()

DB

XML

CAB

crypto

Traditional Policy Interposition

Functionality is scattered

File Access Checker

Prevent
access
to certain
files

import

open()

DB

XML

CAB

crypto

Must check **all** code paths

Traditional Policy Interposition

Functionality is scattered

File Access Checker

Prevent
access
to certain
files

import

open()

DB

XML

CAB

crypto

Must check **all** code paths including future code!

Key Observations

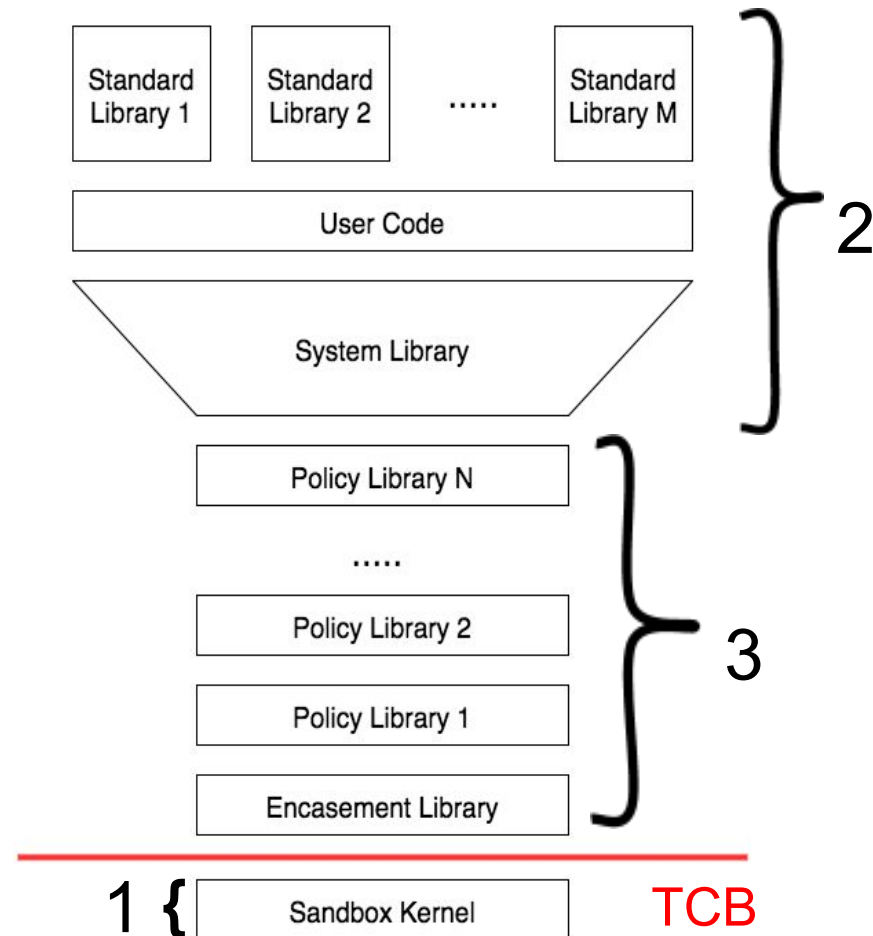
- Most code (and bugs) in libraries.
- Large TCB is the result of:
 - Legacy module reuse
 - Policy existing in the TCB (isolation)
- Policy implementation is difficult.

Repy V2

- Securing library code

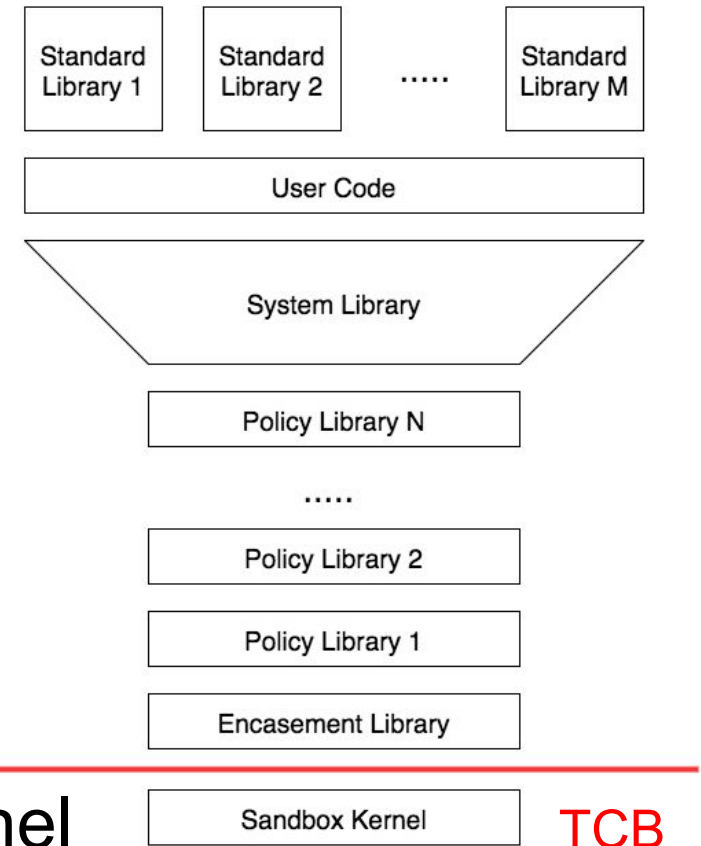
- Small kernel (TCB)¹
 - Externalize library code²
- Privileges are external³
 - Isolation above the kernel
 - Explicit capability passing
- Privileged ops through kernel

Failures outside of the kernel are non-fatal.



Repy V2

- Securing library code
 - Small kernel (TCB)
 - Separate components above
 - Privilege is external
 - Isolation above the kernel
 - Explicit capability passing
 - Privileged ops call through kernel



Failures outside of the kernel are non-fatal

Sandbox Computation

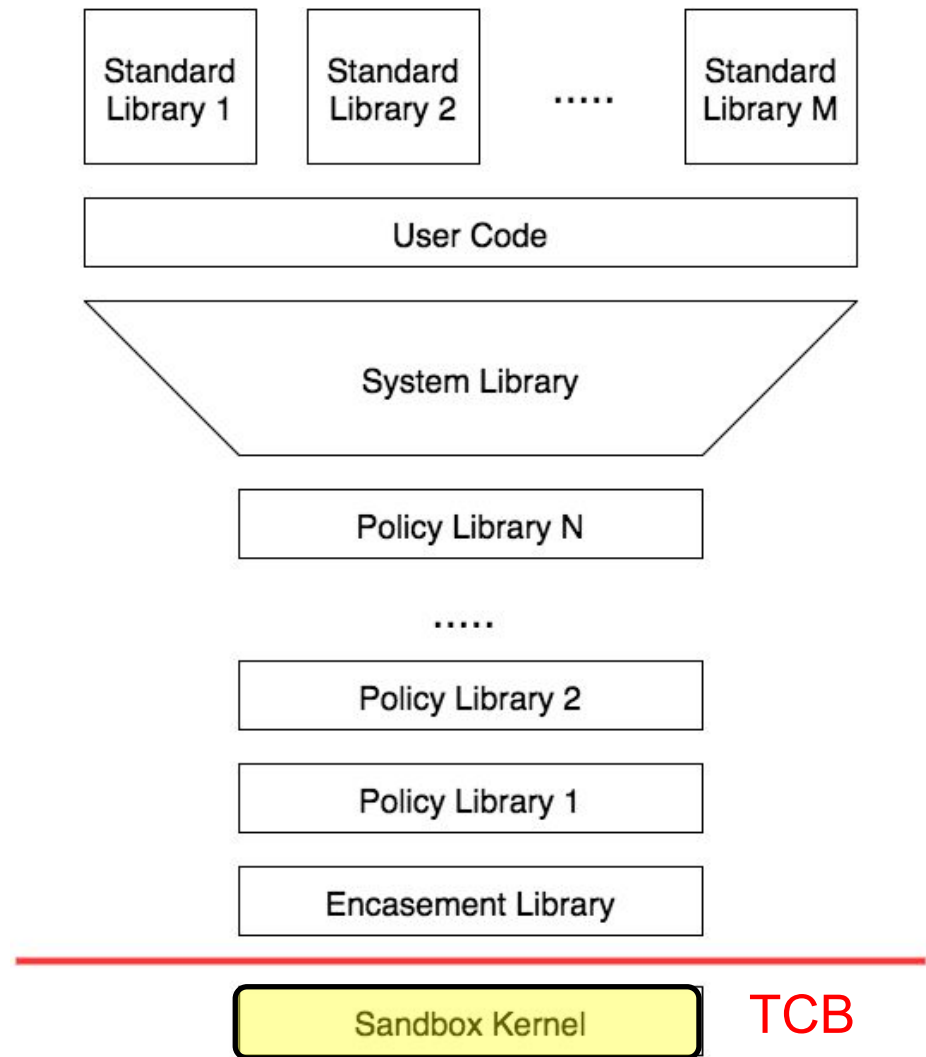
- Subset of Python (Repy)
 - Created for the testbed
 - Supports ‘object-capability’ interposition

Details in the paper

http://isis.poly.edu/~jcappos/papers/cappos_seattle_ccs_10.pdf

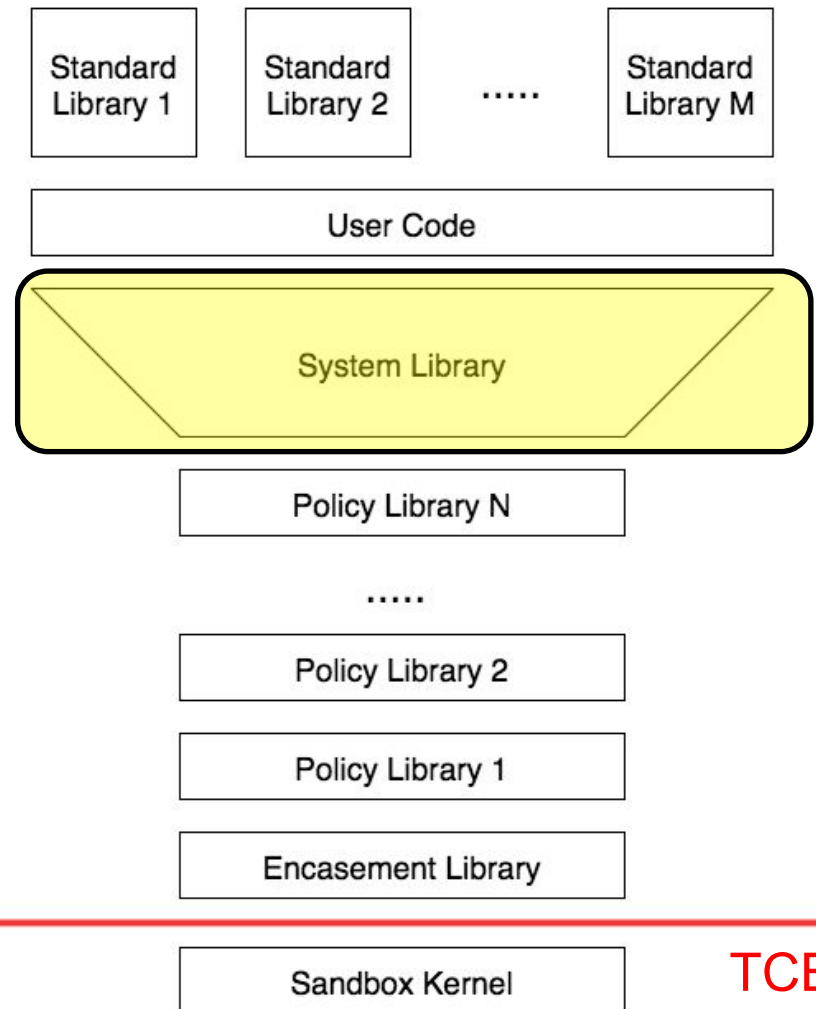
Sandbox Kernel

- 32 capabilities
 - 13 network, 6 file I/O, 3 lock, 3 thread, 2 namespace, 4 information, and 1 random
 - No import, etc.
- Small implementation
 - Above glibc
 - From Python: 1.5K native, 2K Python
 - 4.5K Seattle code (Python)



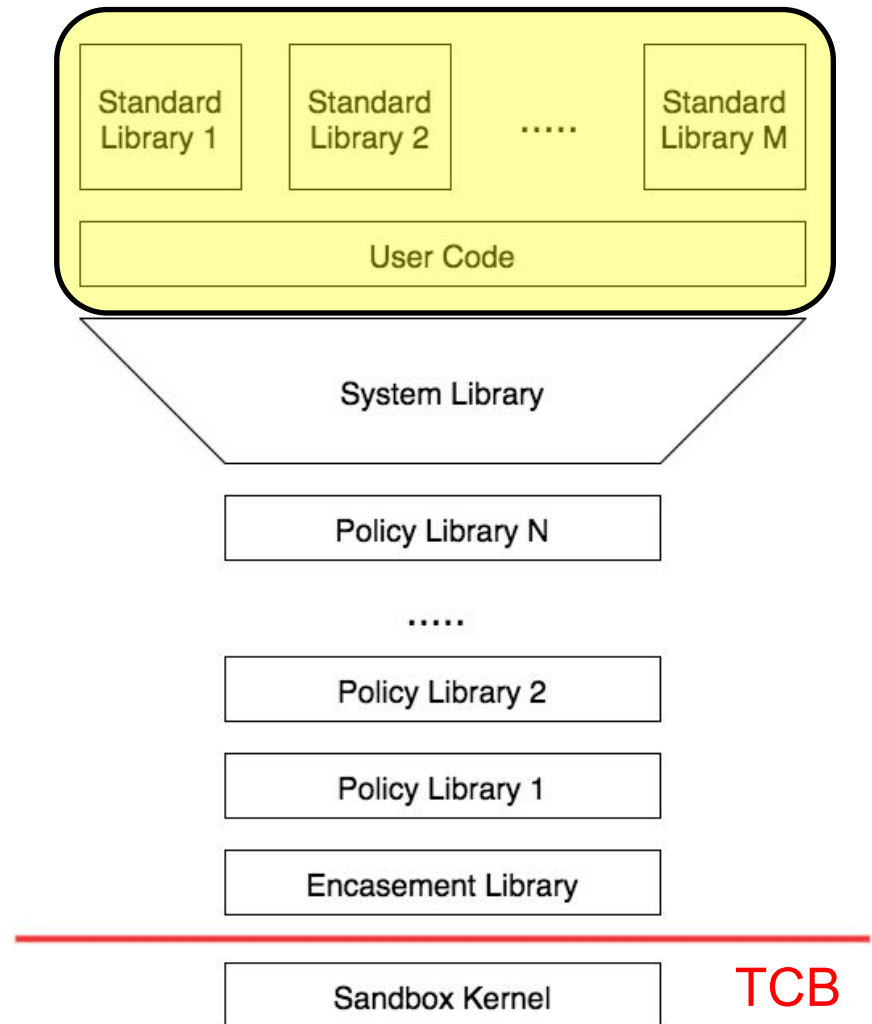
System Library

- System library functionality
 - Similar role to `stdlib.h`
 - Import
 - Uses file I/O to read in code
 - Code safety evaluate
 - Instantiate with dict
 - Remap symbols
 - Rich File I/O, Network, etc.



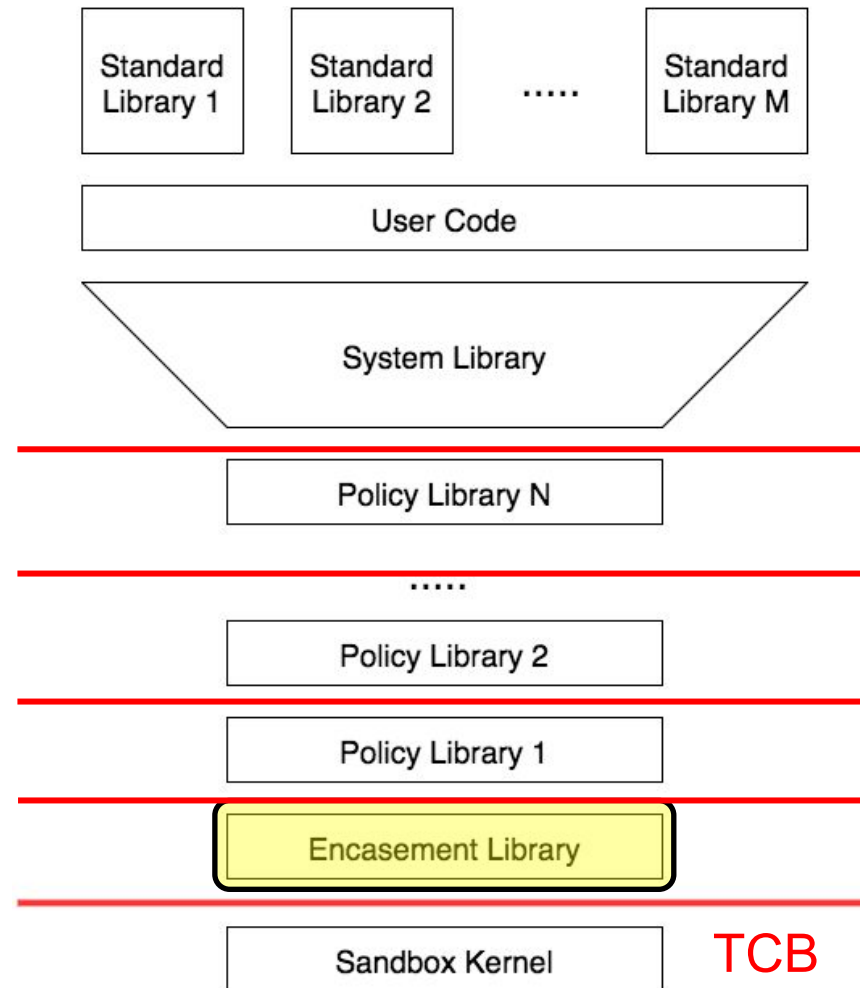
Standard Libraries

- User programs
 - Are written in “normal” Python
 - Different standard libraries
- Standard Libraries
 - crypto, XML parsing, serialization, arg parsing, encoding, NAT traversal, HTTP client / server code, RPC, synchronization primitives, compression, etc.



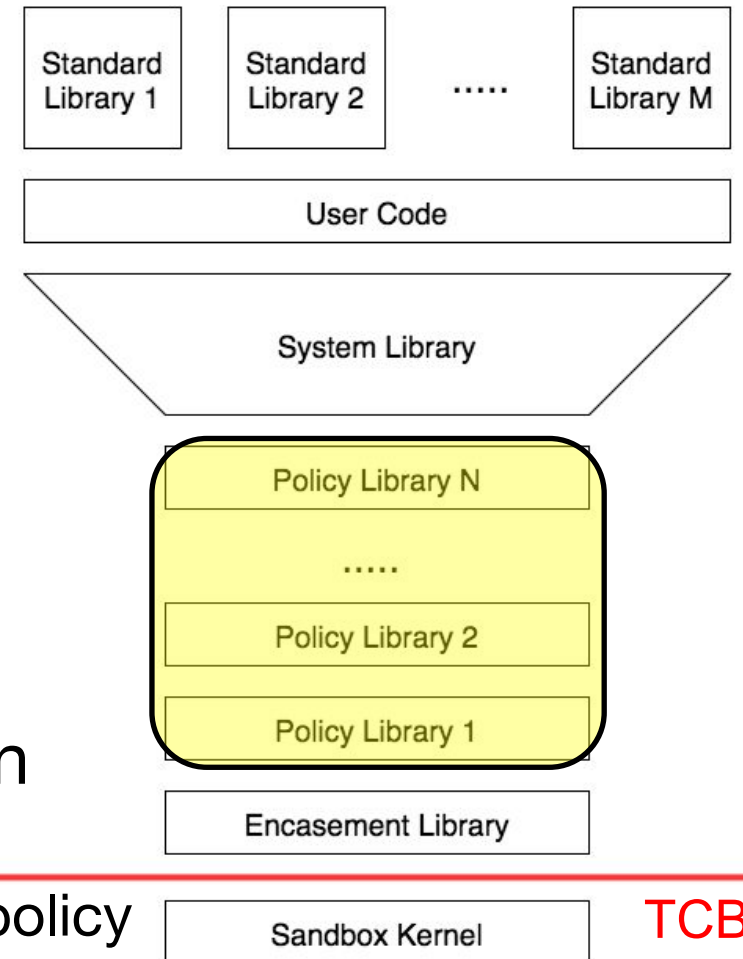
Security Layers

- Encasement library
 - Goals:
 - Prevent capability leakage
 - Check boundary conditions
 - Prevent TOCTTOU
 - Same protections as RPC
 - Contract specifies semantics
 - Wraps objects and calls
 - Completely untrusted by the sandbox kernel



Policy Libraries

- Capability interposition
 - Allows policy decisions
 - Easy to reason about
 - **Transparent!**
- Forensic Logging
 - Write traffic information to log
- Controlled Node Communication
 - Maintains Seattle membership data
 - Denies or allows traffic by owner's policy



Vulnerability Containment

Fatal Java bugs from Paul [Comp & Sec 06]:

- Unchecked load in Bytecode verifier [CVE-1999-0440]
- Execute apps with expired signatures [CVE-2001-1008]
- XML vulnerability allows code loading [CVE-2002-0865]
- DB vulnerability allows code loading [CVE-2002-0866]
- CAB loader missing security [CVE-2002-1293]
- / in classname bypasses security [CVE-2003-0896]
- Reflection vulnerability 1 [CVE-2005-3905]
- Reflection vulnerability 2 [CVE-2005-3906]

Security bugs are often minimally disclosed!

Vulnerability Containment

Fatal Java bugs from Paul [Comp & Sec 06]:

- Unchecked load in Bytecode verifier [CVE-1999-0440]
- Execute apps with expired signatures [CVE-2001-1008]
- XML vulnerability allows code loading [CVE-2002-0865]
- DB vulnerability allows code loading [CVE-2002-0866]
- CAB loader missing security [CVE-2002-1293]
- / in classname bypasses security [CVE-2003-0896]
- Reflection vulnerability 1 [CVE-2005-3905]
- Reflection vulnerability 2 [CVE-2005-3906]

Not enough info

Security bugs are often minimally disclosed!

Programming Language VM summary

Sandbox code in a language interpreter / runtime

Pros:

- Low overhead per VM
- Possible to reason about some security properties
- Architecture independent, somewhat OS independent

Cons:

- No backwards compatibility
- Some performance cost
- Very poor resource isolation (OS interface)
- Can have a bloated TCB / ill defined system interface

Assignment 2.1 - Reference Monitors

Build a Reference Monitor

- Goal: Better understand security mechanisms
- Task: Write a reference monitor for the Seattle VM (version 2.0)
 - Get a basic understanding about reference monitors
 - You will have to build reference monitor in Remy V2(Restricted Python for Seattle) using instructions.
 - Go to the following link:
<https://github.com/SeattleTestbed/docs/blob/master/EducationalAssignments/UndoPartOne.md>
 - Use instructions on the page above to download Remy V2. Go to next slide to know what your reference monitor should do.

Assignment 2.1 - Reference Monitors

Building a reference monitor

- Write reference monitor for Seattle
- Use security layers functionality
 - see the security layer example on the web page
- Rules:
 - Implement '**undo**' operation after a valid '**writeln**'.
 - Be accurate: Only modify what is required.
 - Be efficient: Use minimum number of resources
 - Be secure: Make sure that an attacker cannot circumvent this!
- DO NOT "log" anything in your code!!!

Specifications:

- Changes using the 'writeln' becomes permanent after a valid 'writeln' or after the file is closed.
- The 'undo' operation will reverse the prior valid 'writeln' operation

Assignment 2.1 - Rubrics

- +20 turning in a file in the correct format
- +20 file obeys assignment directions with respect to efficiency
- +60 determined by number of students whose attack code bypasses your security layer

Resources

Learn Python / Seattle

Python tutorial: <http://docs.python.org/tutorial/>
<https://www.w3schools.com/python/>

Seattle and Repy tutorials:

- <https://github.com/SeattleTestbed/docs/blob/master/Programming/PythonVsRepyV2.md>
- <https://github.com/SeattleTestbed/docs/blob/master/Programming/RepyV2Tutorial.md>
- <https://github.com/SeattleTestbed/docs/tree/master/UnderstandingSeattle>

Reading Next Week

Read: (Wikipedia level)

- Linux file permissions
 - http://en.wikipedia.org/wiki/Filesystem_permissions
- Windows file attributes
 - http://en.wikipedia.org/wiki/File_attribute
- Android security
 - <http://developer.android.com/guide/topics/security/security.html>

Security policies in Seattle

"Retaining sandbox containment despite bugs in privileged memory-safe code" (CCS 10)

Purpose: Understand the basics and terminology

Exam 1

- Do not sit next to someone. Spread out!
- Put all bags / coats / electronic devices up front
- Use the bathroom now if you need to
- **No questions / clarifications during the exam**
- You only need your NYU ID card and a pen or pencil

Tips:

- **Do not try to be funny!** (I grade on what you write)
- If you need a pen, raise your hand
- After you can turn in your exam you can go
- There is no rush (you have plenty of time)