

Isolated Program Execution: An Application Transparent Approach for Executing Untrusted Programs

Authors: Z Liang, V Venkatakrisnan, R Sekar

Computer Security Applications Conference, 2003

Presenters:

Renee Warriner, Bernice Chan, Fareha Shafique

ECE1776 October 10, 2006

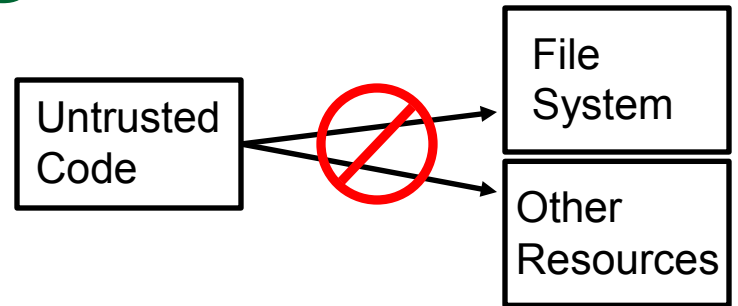
Problem

- Security of computer systems dependent on user behaviour
 - Downloading and executing malicious code
 - Firewalls cannot protect against direct program execution
 - Antivirus programs cannot detect all malicious code
-

Containment Strategies

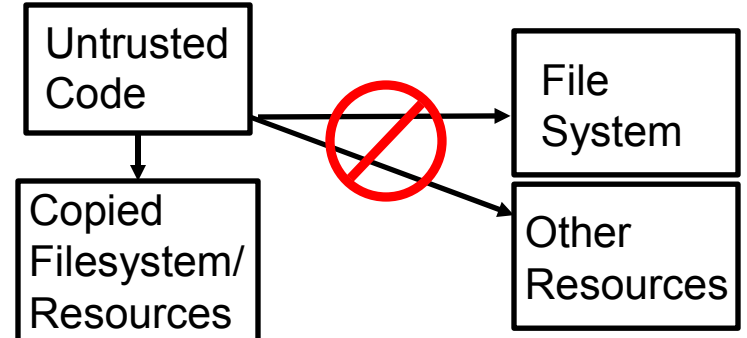
■ Sandboxing

- Untrusted code restricted from accessing resources



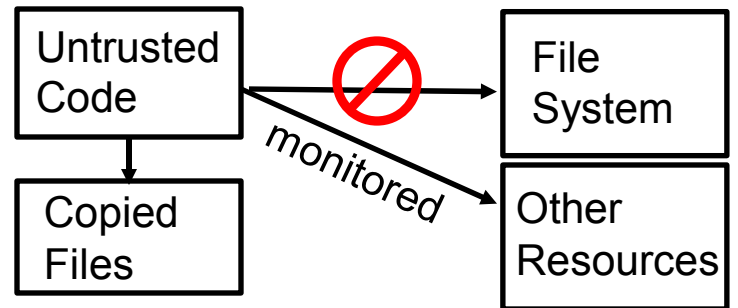
■ Isolated Execution

- Actions of untrusted code isolated from other applications by using copy of filesystem contents



■ Logical Isolation

- Effects of untrusted program logically isolated from other processes



Proposed Solution

- Copy-on-write for any file-system changes
 - Modified files are copied to cache
 - Cache is hidden from other applications
 - Files only visible when committed by user
 - User can review and commit/abort file changes
 - System Call Interposition
 - Access to non-file resources restricted
 - System state changes prevented
 - System state queries allowed
-

Program Structure

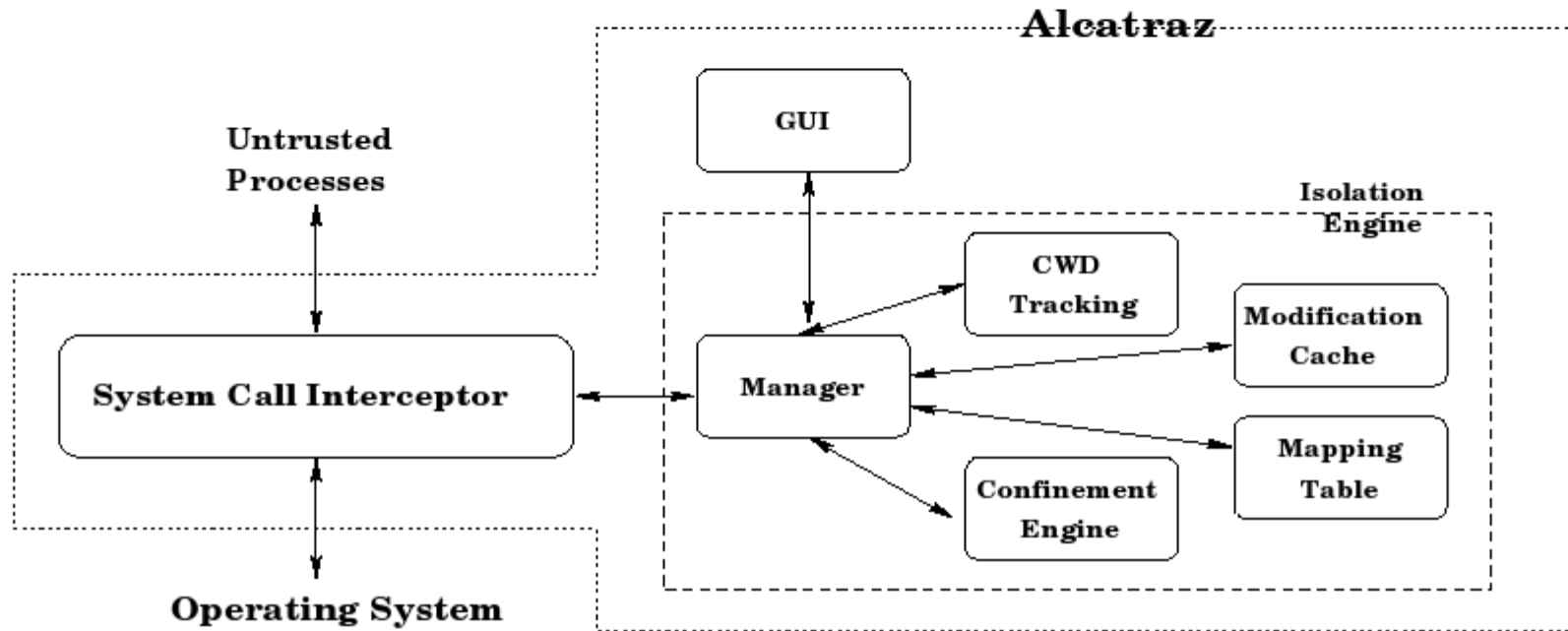


Figure 1. System Architecture

Program Structure

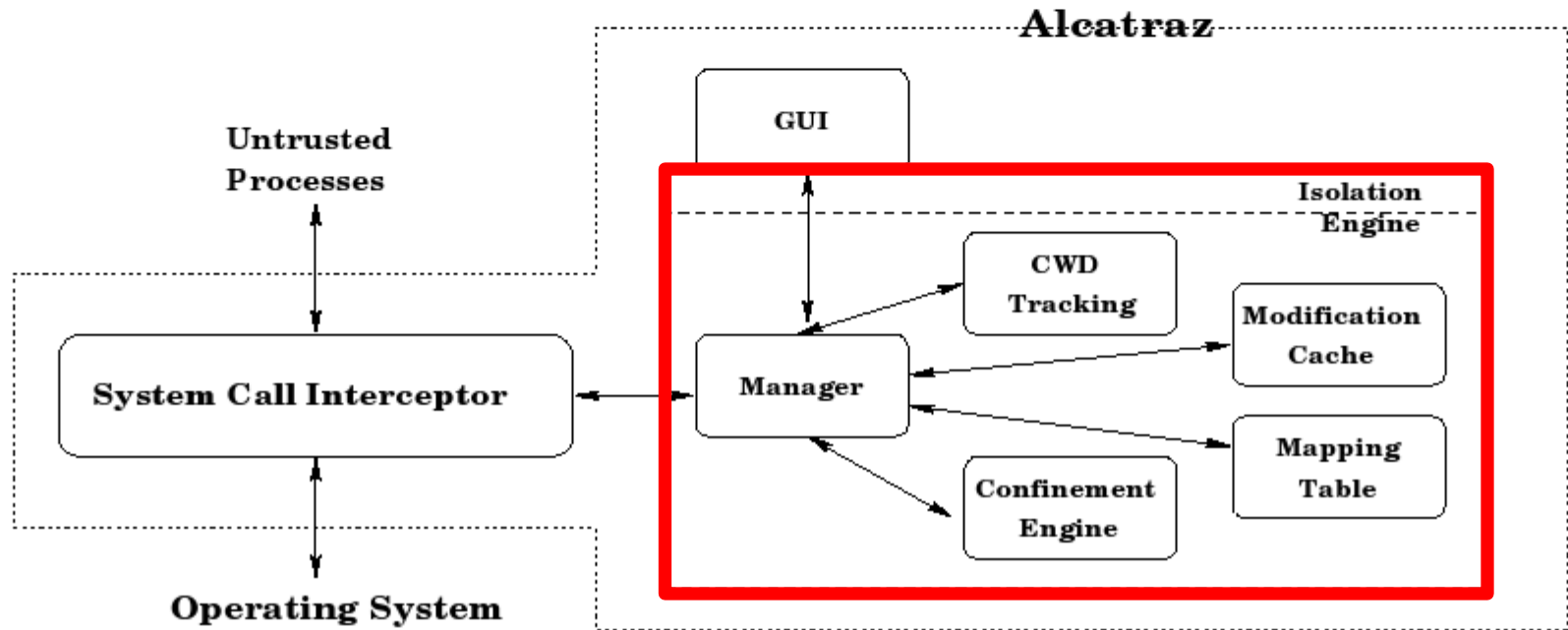


Figure 1. System Architecture

Isolation Engine: Handles file and nonfile system calls

Program Structure

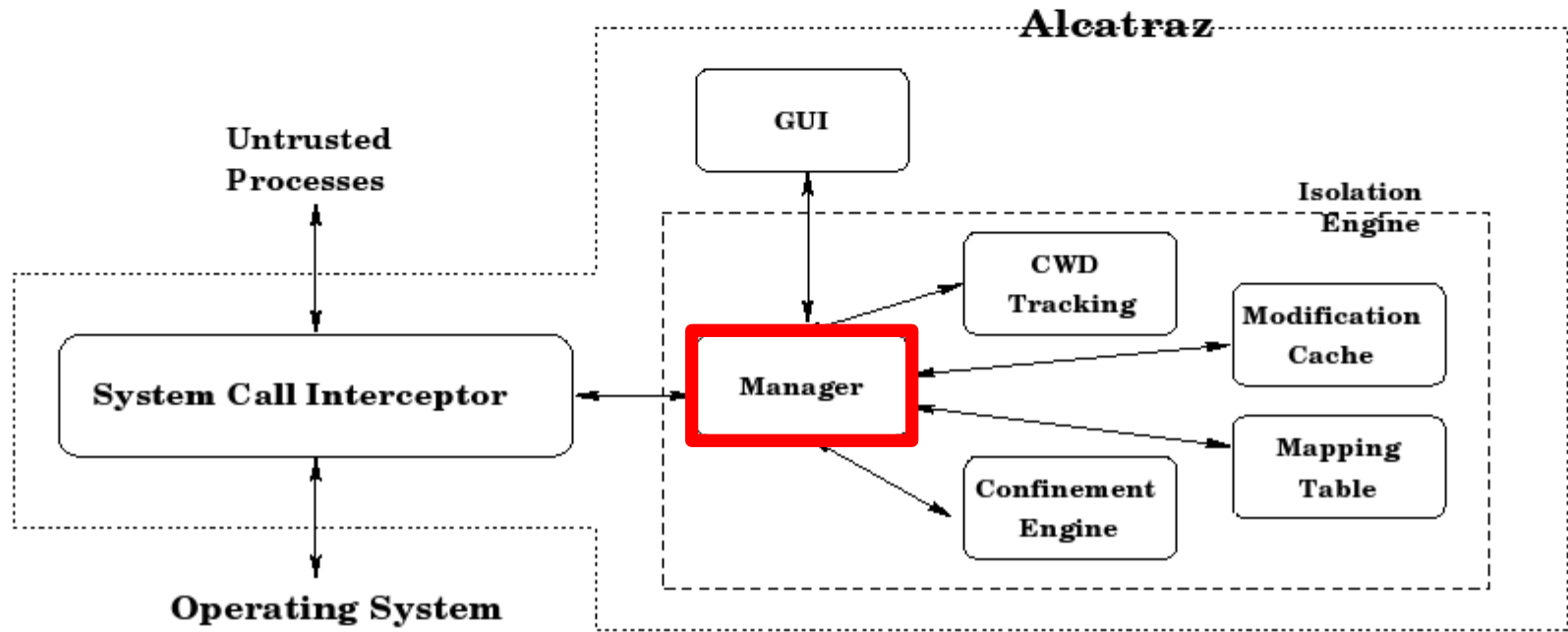


Figure 1. System Architecture

Manager: Coordinates activities of isolation engine

Program Structure

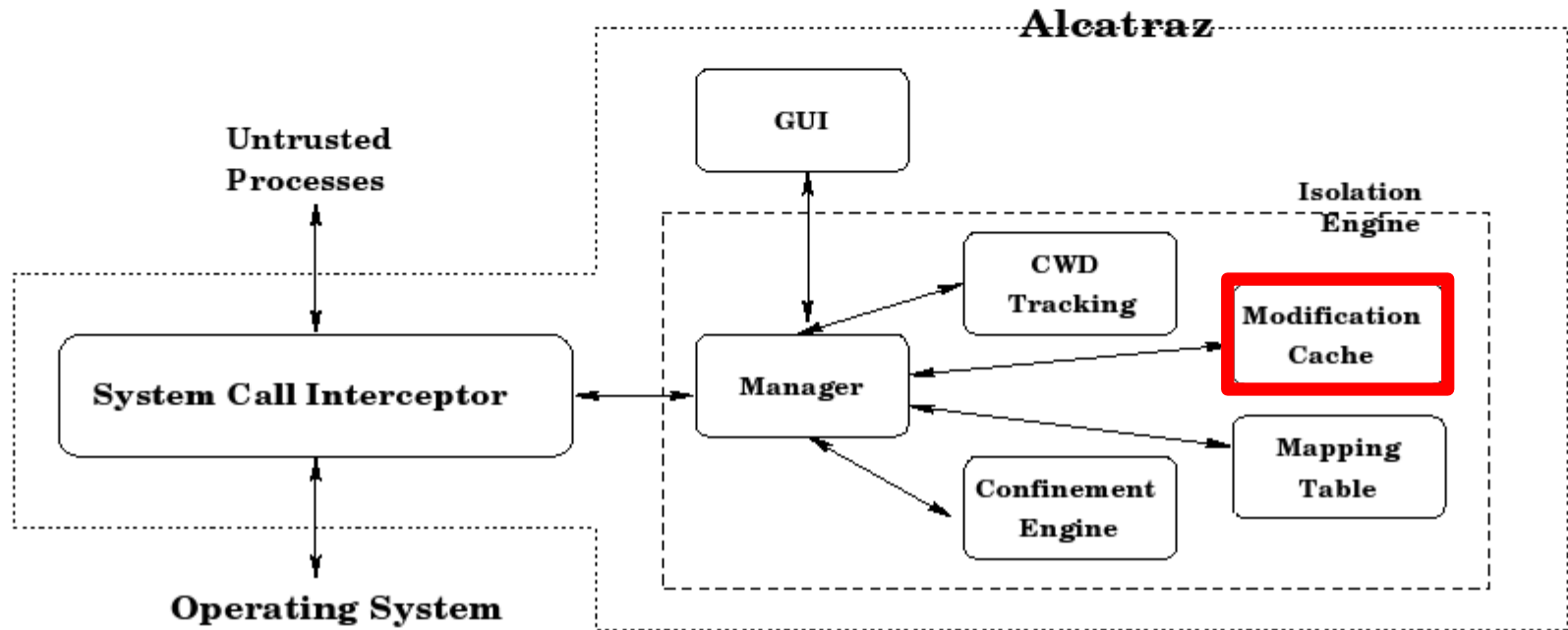


Figure 1. System Architecture

Modification Cache: Stores modified files

Program Structure

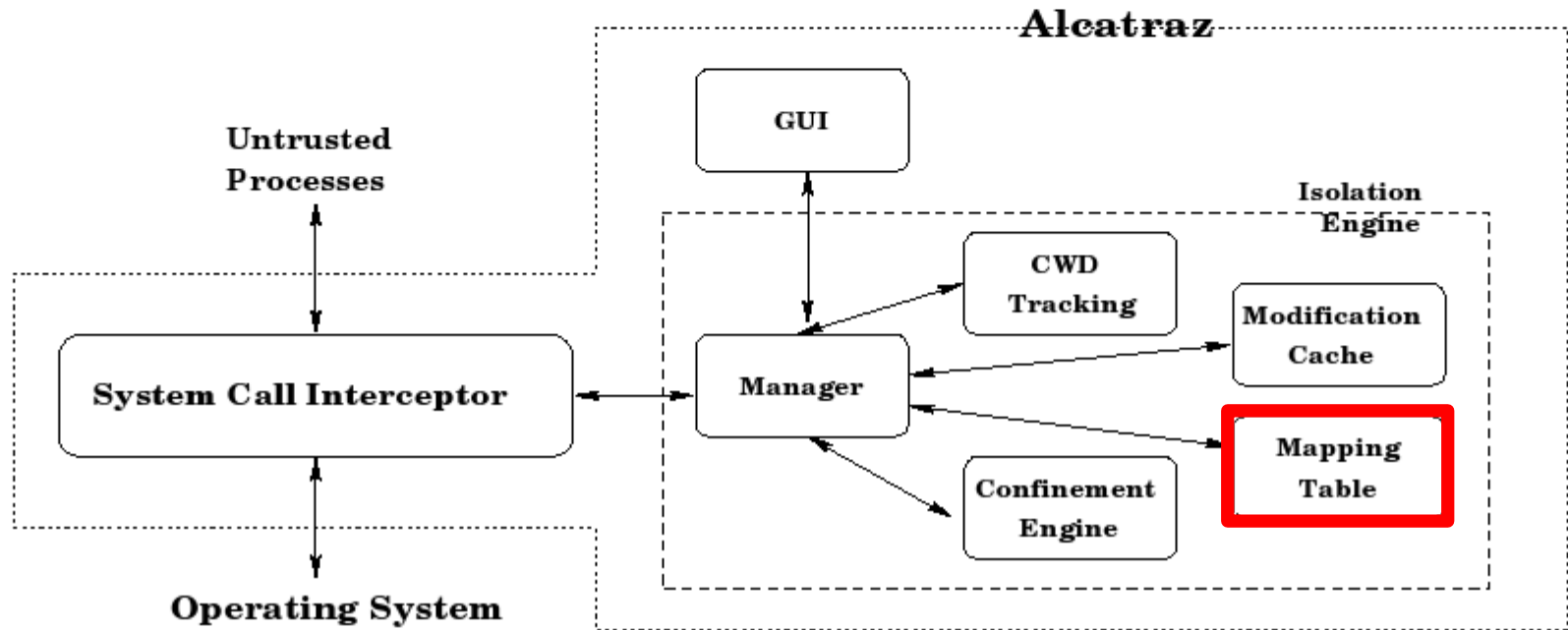


Figure 1. System Architecture

Mapping Table: Translates original and copied filenames

Program Structure

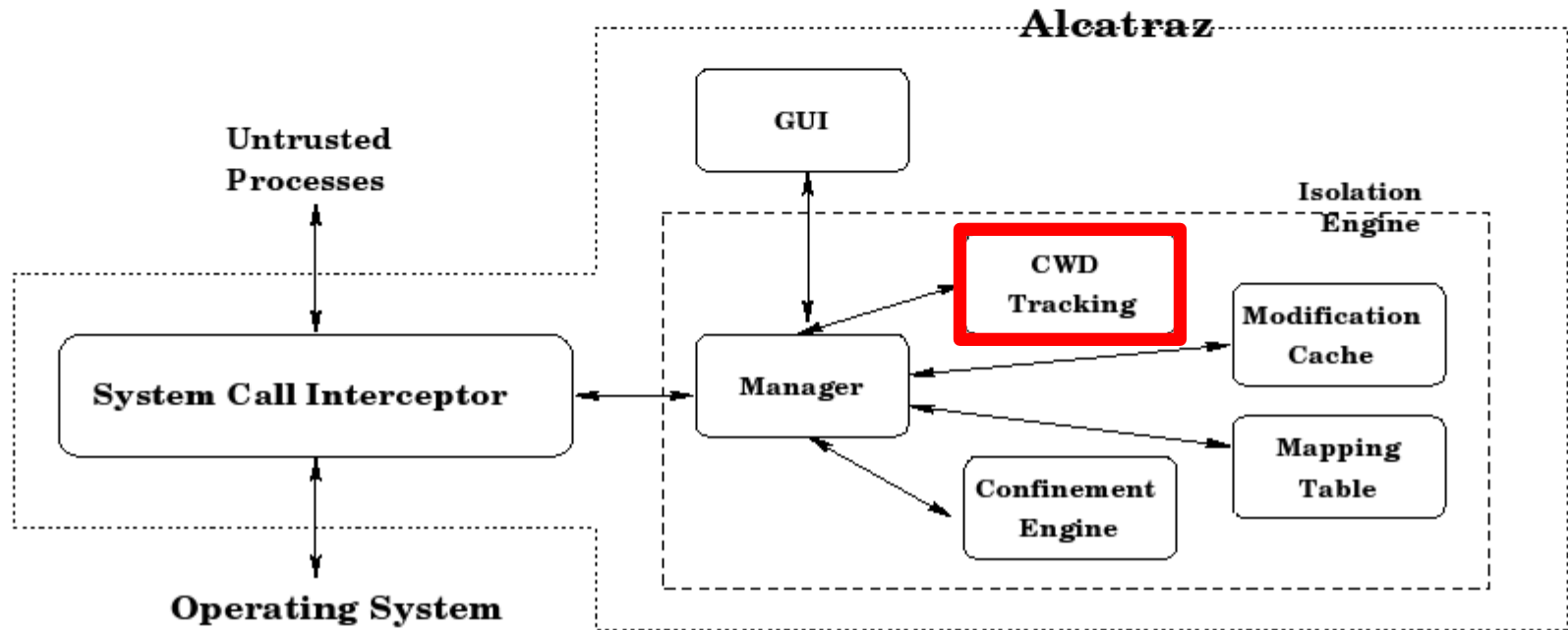


Figure 1. System Architecture

CWD Tracking: Tracks current working directory

Program Structure

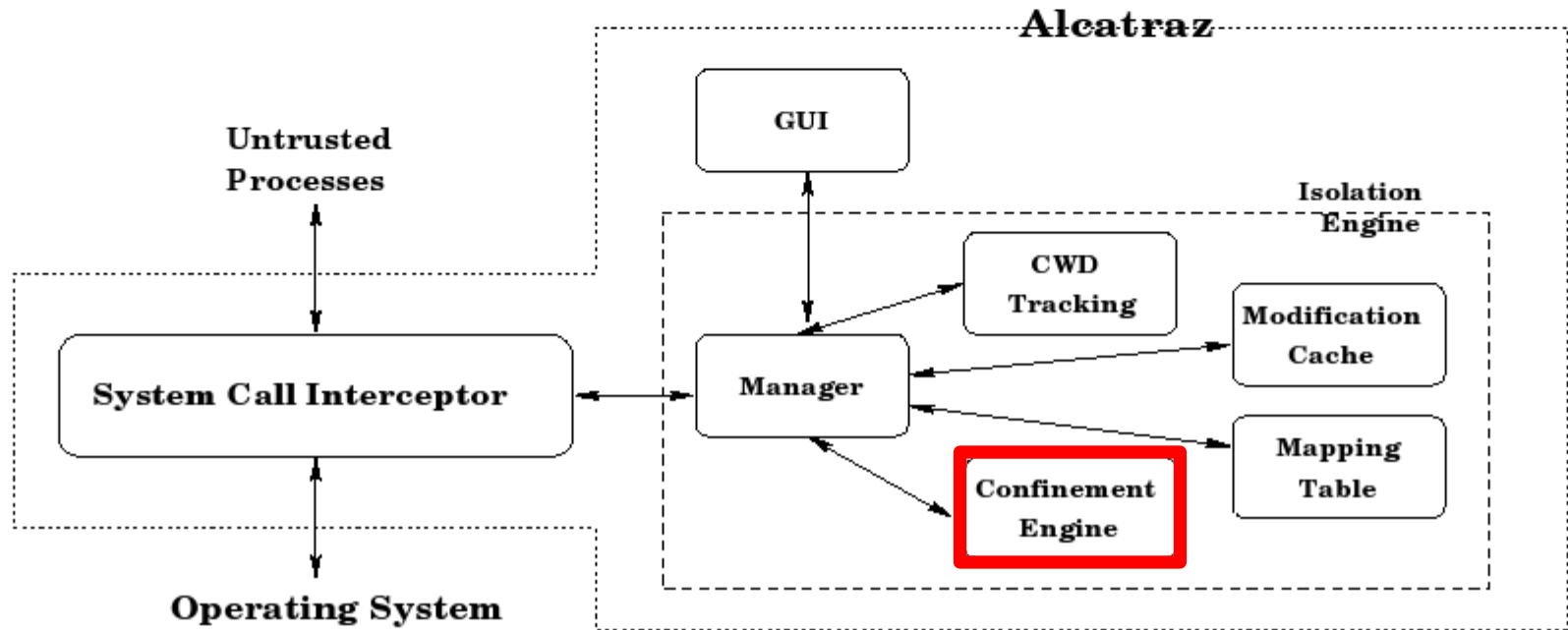


Figure 1. System Architecture

Confinement Engine: Handles nonfile system calls

Program Structure

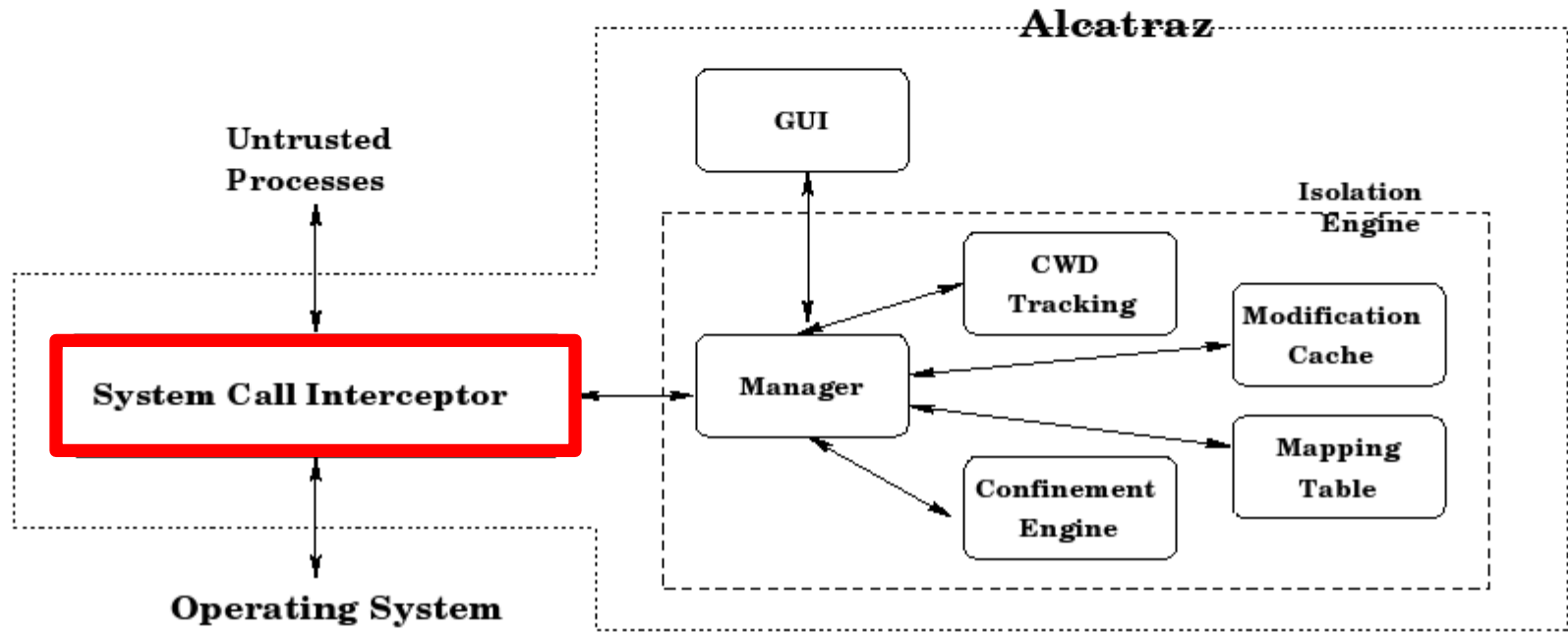


Figure 1. System Architecture

Interceptor: Catches all system calls by untrusted process

Program Structure

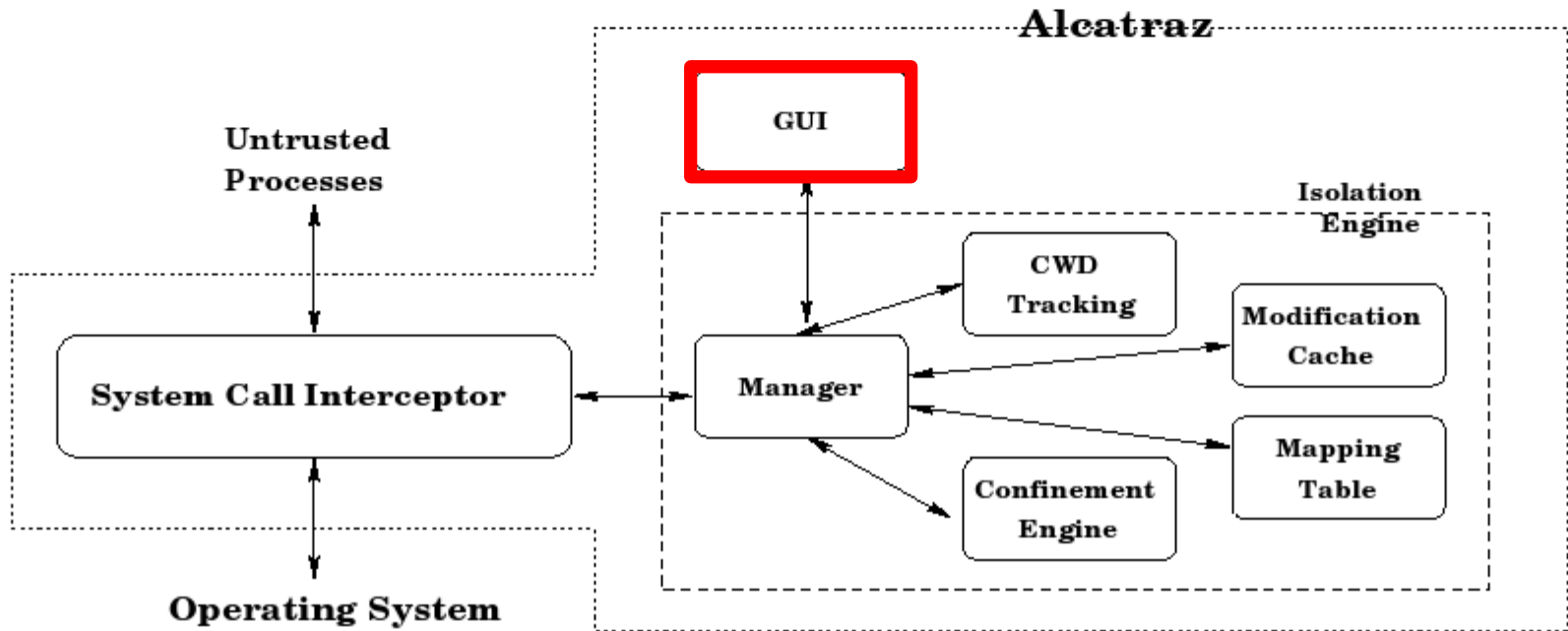


Figure 1. System Architecture

GUI: Presents file changes to user for commit/abort

File Modifications

- Regular File
 - Copy-on-write to modification cache
 - File original and copy mapped in mapping table
 - Directory
 - Inefficient to copy all directory contents
 - Directory changes logged in modification cache
 - Symbolic Link
 - Treated as directory modifications
 - Inode
 - File metadata changes using copy-on-write of file
 - Also changes file owner
-

Performance Results

- Successfully detected file modifications on three downloaded programs
 - photopages, mpls, Mozilla installation
 - committed and aborted changes
 - Overhead determined for three programs and common Unix utilities
 - Isolation only < 20% overhead
 - Interposition 10-100% overhead
 - Overhead \propto System call frequency
 - CPU intensive programs < I/O intensive programs
-

Pros

- Creates practical implementation
- User Benefits
 - Common behaviour recognized
 - User friendly
 - Do not need administrator privileges
 - Administrator can setup service-specific proxies that forward service requests
- Software Benefits
 - Application and operating system transparent
 - Works with existing software
 - Monitors file and nonfile operations
 - Allows for testing of newly installed programs
- Hardware Benefits
 - No supplemental hardware required for data storage

Cons / Unanswered Questions

- System protection relies on “copy-on-write” to modification cache
 - Described as “dedicated area within file system”
 - No details on how their system “Alcatraz” is able to access this area, but is protected against malicious processes
 - Claims to be application & OS independent
 - Uses Linux ptrace to intercept system calls
 - Is there a Windows equivalent?
-

Cons / Unanswered Questions (2)

- User must make all commit/discard decisions:
 - User can peruse changes within the Alcatraz GUI or an isolated
 - Look at files created, overwritten, modified, etc.
 - What if many files have changed... how is a user expected to look through all of the changes?
 - Doesn't solve "click-fatigue" problem of sandboxing policies
-

Cons / Unanswered Questions (3)

- Is the Alcatraz GUI secure?
 - How does the system ensure that the summary report is not tampered with?
 - What if the summary report looks “clean” and the user chooses to commit everything?
-

Cons of Paper Content & Format

- Only performance results for non-malicious applications are provided
 - Downloaded examples (picturepages, mpls, mozilla installation) and common UNIX utilities (make gcc, ghostscript, tar, gzip)
 - How would the system handle a malicious application?
 - It would be interesting to see performance comparisons vs. sandboxing techniques
-