Secure and Simple Sandboxing in SELinux

James Morris
jmorris@namei.org

FOSS.my 2009
Kuala Lumpur, Malaysia
Overview

● Sandboxing

● SELinux

● Sandbox design and implementation

● Use examples

● Status and future directions
Sandboxing

- Many types of sandbox

- Basic concept is to isolate code
  - Process arbitrary input
  - Run third party code
  - Contain vulnerabilities

- For this talk: process-level sandbox
Existing Sandboxes

- Chroot, seccomp, ptrace etc., all problematic
- New design: setuid sandbox (Evans/Tiennes)
- Don't utilize MAC facilities (SELinux, Smack)
- Typically based around restricting ambient privilege
Sandboxing with MAC

- Utilize MAC (mandatory access control) to enhance sandboxing

- Layered approach:
  - Process-level isolation (MMU)
  - DAC separation (e.g. privsep, UID allocator)
  - Namespaces / chroot etc.
  - MAC isolation policy
Reduce Ambient Authority

• Security can be simplified by reducing ambient authority.

• Consider:
  
  • `wc file.txt`
    
    – `wc` needs general read permission for the system & uses this 'ambient' authority to open file.
  
  • `cat file.txt | wc`
    
    – `cat` opens the file and passes open fd to `wc`, bundling the object and authority together. Specific authority is delegated and `wc` now needs no permissions to access filesystem!
Usability

• Combining MAC policy with fd passing is conceptually simple for users: the latter follows standard Unix use conventions.
• Does not require policy administration
  • Simple supplied policy which strongly isolates sandboxed apps
  • Zero config
• High level abstraction:
  • Simply run apps via a sandbox launcher
  • Kiosk mode, sVirt etc. are similar approaches
SELinux Implementation

- New sandbox label added to policy
  - **Has no general permissions**, only those absolutely required to execute (e.g. load shared libraries, which can be further locked down if desired)
  - *sandbox* launcher causes app to be executed with this label; I/O happens via fd
  - Unique MCS label assigned to each instance for MAC isolation (cf. UID allocation – both could be used)
  - Sets up home & tmp dirs; copies in specified files; cleans up at exit
/usr/bin/sandbox

- Creates temporary sandbox directory
- Copies in specified files
- Sets up security labeling
- Executes specified application in sandbox
- Cleanup at exit
Basic Use

$ /usr/bin/id -Z
unconfined_t:c0.c1023

$ sandbox /usr/bin/id -Z
sandbox_t:c533,c903

- sandbox_t -- broad MAC policy for all sandboxes, isolate them from wider system
- c533,c903 -- unique MCS label to separate sandboxes from each other (actual value does not matter, just needs to be unique)
Demonstration

$ touch /tmp/foo1

$ sandbox touch /tmp/foo2
/bin/touch: cannot touch `/tmp/foo2': Permission denied
Demonstration

$ sandbox cat /proc/$$/maps
/bin/cat: /proc/3034/maps: Permission denied
Advanced Uses

• Processing pipelines:
  • Scanning mail for viruses, spam etc.; run each stage in a sandbox
  • Packet dissectors, etc.

• Web application framework
  • e.g. XSLT rendering, CGI handling

• Any case where a separate process can be launched and use fd for I/O
Desktop Security

- Difficult to sandbox desktop apps because of environment (X, GNOME, DBus etc.)
  - complicated

- Sandbox X:
  - Launch sandboxed applications in nested X server: *simple* and effective!
  - Extends basic sandbox utility:
/usr/sbin/seunshare

- setuid program:
  - unshare(2) – dissociates mount namespace
  - bind mounts new $HOME and /tmp dirs
  - calls setexeccon to set security label
  - drops all capabilities
  - calls sandboxX.sh
/usr/share/sandbox/sandboxX.sh

- Configures X environment
- Launches Xephyr nested X server
  - runs matchbox window manager
  - runs specified application
  - everything runs with sandbox security label
  - cleans up at exit

- Some limitations (currently):
  - Cannot resize window
  - No copy/paste
Current status

- SELinux Sandbox will be in Fedora 12
  - Currently available in rawhide
Demo
Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

KEN THOMPSON

INTRODUCTION
I thank the ACM for this award. I can’t help but feel that I am receiving this honor for timing and serendipity as much as technical merit. UNIX swept into popularity with an industry-wide change from central mainframes to autonomous minis. I suspect that Daniel Bob-
Future Directions

- Continued high-level integration, e.g. make it easy to run sandboxed web browsers
  - Interaction issues to resolve, e.g. ask user to save changed data when exiting sandbox?

- Integration with XACE window labeling, hardware security etc.

- Use sandboxing to restrict administrative privilege
What we really need most...

- A standardized high-level API
- Developers / ISVs currently roll their own security or just give up
- Difficult, but can be done
Resources

- Dan Walsh's blog
  - danwalsh.livejournal.com

- Dan Walsh's LPC talk
  - http://video.linuxfoundation.org/video/1565

- Dan Walsh's email address & cell phone
  - dwalsh@redhat.com
  - +1 212-555-4240