The methods of sandboxing and isolation

Ján Lieskovský
Software Engineer, Red Hat
November 2015
Perspective

- Motivation
- Application level approach
- Fully-fledged sanboxes
- OS concepts
- Sandbox as development / application platform
Motivation

• Security (primary)
  – User's responsibility
  – Developer's role

• Independence (recent)
  – Gnome Sandboxed Applications
  – CS50 Sandbox
Approaches to sandboxing / isolation

- Classical
  - It's user's / administrator's responsibility to:
    - Choose sandbox solution
    - Apply resource (CPU, memory, disk size, fds) limits
    - Ensure processes can interact together only in safe way (can't access other process resources, can't kill each other etc.)
  - Developer provides the code (OS, library, end app, framework etc.)

- Recent
  - Take the responsibility for secure SW design from users / administrators to developers
  - Instead of one monolithic server handling multiple protocols within the same code base (even using shared memory) prefer code modularity based on 'least privilege approach'
  - Apply currently available isolation mechanism to improve the security of the final product
Application level approach

First Perl use case – **taint mode**:

- Based on concept of “tainted” data
- Exceptions:
  - Not checked for taintedness:
    - Arguments of `print()`, `syswrite()`
    - Symbolic methods and symbolic sub references
  - Hash keys – never tainted

- Tainted data:
  - Detection `sub is_tainted()`
  - Laundering
    - By using values as keys in a hash
    - Or by referencing subpatterns from a regular expression match
First Perl use case – **taint mode**:

- Based on concept of “tainted” data
- Exceptions:
  - Not checked for taintedness:
    - Arguments of `print()`, `syswrite()`
    - Symbolic methods and symbolic sub references
  - Hash keys – never tainted
- Tainted data:
  - Detection `sub is_tainted()`
  - Laundering
    - By using values as keys in a hash
    - Or by referencing subpatterns from a regular expression match

Already see what can go wrong here?
Application level approach

Second Perl use case – Safe module:

- Introduces the concept of “safe compartments”
- Each compartment has:
  - New namespace
    - Namespace root is changed to a new package
    - Code evaluated within compartment can't refer to variables outside
    - Code evaluated outside can share (predefined) variables
  - Operator mask
    - List of operators to be masked / prevented from execution
    - Default value is ':default' optag
- Methods for compartment objects:
  permit(), permit_only(), deny(), deny_only(), share(), share_from(), reval() ...
Application level approach

What can go wrong?

taint mode CVEs

- CVE-2011-2201
- CVE-2011-1487

Safe.pm CVEs

- CVE-2010-1168
- CVE-2010-1169
- CVE-2010-1447
Fully-fledged sanboxes

**Java sandbox Model**

- Introduced in JDK 1.0:
  - Local code executed directly in JVM
  - Remote in dedicated sandbox

- Version JDK 1.1 introduced concept of “signed applet”:
  - Local code executed directly in JVM
  - Remote code:
    - Signed?
      - Signature recognized as trusted?
        - Executed as local
      - Still executed in sandbox

- Version JDK 2
  - Fine-grained access control
  - Configurable security policy
  - Extensible access control structure
  - Security checks for all programs (both signed applets and any Java applications)
Fully-fledged sanboxes

Java v.2 sandbox model:

• The Byte Code Verifier:
  – Checks (syntactic) Java byte-code for correctness
  – Validates pointers, prevents access restriction violations, using objects with incorrect type information etc.

• The Java Applet Class Loader
  – Defines conditions under which an applet can add classes into running JVM, validates namespaces etc.

• The Java Security Manager
  – Restricts the ways in which an applet can use concrete class methods
  – Prevents execution of dangerous operation by throwing a SecurityException
Java Sandbox model

What can go wrong?

- **57 CVEs found** searching master CVE database using keys:
  
  "java+sandbox+escap"
Fully-fledged sanboxes

SELinux Sandbox

Observation:
- Many applications require access to user's home directory and /tmp to work properly

SELinux Sandbox approach:
- Application launch = new empty /home and /tmp directories each time
- Independence – application can't see / modify directories of other application running in sandbox
- Applications originally can't:
  • Access Window Manager
  • Access network
  • Access other files than stdin, stdout, or files included by sandbox launch
Fully-fledged sanboxes

SELinux Sandbox

SELinux Sandbox workflow:
- Setup file system
- Create two new directories (each of them having own SELinux context):
  - One in user's home directory
  - Second in /tmp
- Select random MCS label
- Copy provided input files into homedir and /tmp
- Create .sandboxrc in homedir with command to run
- Execute seunshare (unshare, bind-mount, setexeccon, drop capabilities, exec sandboxX.sh)
- Delete temporary homedir and /tmp
Fully-fledged sanboxes

SELinux Sandbox

SELinux Sandbox selected options:
- **M**  Create sandbox with temporary $HOME and /tmp directories
- **H / -T**  Alternate home / temporary directory
- **i / -I**  Copy selected file / files to temporary directory
- **t**  Specify sandbox type to use
  - **sandbox_t**  No X, no network access, no open
    - RW only on included files
  - **sandbox_min_t**  No network access
  - **sandbox_x_t**  Printer ports only
  - **sandbox_web_t**  Ports for web browsing accessible
  - **sandbox_net_t**  All network ports
- **X**  X based sandbox for GUI apps (temporary $HOME and /tmp, secondary Xserver, **sandbox_x_t**)
- **C / -c**  Specify capabilities / cgroups to pass to sandbox
OS concepts for isolation

Linux kernel namespaces

- 6 types currently
  - IPC      System V IPC, POSIX message queues
  - Network  Network devices, stacks, ports
  - Mount    Mount points
  - PID      Process IDs
  - User     User and Group IDs
  - UTS      Hostname and NIS domain name
OS concepts for isolation

Linux kernel namespaces

- System calls:
  - clone():
    - Creates new process like fork()
    - Allows to share selected parts of execution context
    - NS flags: CLONE_NEWIPC, CLONE_NEWNET, CLONE_NEWNS, CLONE_NEWPID, CLONE_NEWUSER, CLONE_NEWUTS
  - setns():
    - Allows calling process to join existing namespace
  - unshare():
    - Moves the calling process to a new namespace. If CLONE_* flags are provided, new namespace is created for each of those flags and the calling process is made a member of those namespaces
    - CLONE_NEWUSER doesn’t require a capability, but the rest require CAP_SYS_ADMIN capability when creating new namespace via clone() and unshare()
OS concepts for isolation

Linux kernel namespaces

Examples:

$ unshare --help

Usage:
unshare [options] <program> [<argument>...]

Run a program with some namespaces unshared from the parent.

Options:
-m, --mount      unshare mounts namespace
-u, --uts        unshare UTS namespace (hostname etc)
-i, --ipc        unshare System V IPC namespace
-n, --net        unshare network namespace
-p, --pid        unshare pid namespace
-U, --user       unshare user namespace
-f, --fork       fork before launching <program>
--mount-proc[=<dir>]  mount proc filesystem first (implies --mount)
-r, --map-root-user map current user to root (implies --user)
-s, --setgroups  allow|deny control the setgroups syscall in user namespaces

-h, --help       display this help and exit
-V, --version    output version information and exit

For more details see unshare(1).
OS concepts for isolation

Linux kernel namespaces

Examples:

$ ip netns help
Usage: ip netns list
    ip netns add NAME
    ip netns delete NAME
    ip netns identify PID
    ip netns pids NAME
    ip netns exec NAME cmd ...
    ip netns monitor
OS concepts for isolation

Linux kernel namespaces

Further information:

$ man namespaces
$ man pid_namespaces
$ man user_namespaces
$ man unshare / man 2 unshare
OS concepts for isolation

Capabilities

• Historical user roles separation:
  – Privileged
  – Unprivileged

• Kernel 2.2 introduced capabilities:
  – More distinct units for superuser account
  – Per-thread attribute
OS concepts for isolation

Capabilities

Selected prototypes:

- CAP_CHOWN: Arbitrary UID/GID changes
- CAP_DAC_OVERRIDE: Bypass RWX permission checks
- CAP_DAC_READ_SEARCH:
  - Bypass file R, dir RX permission checks
  - Invoke `open_by_handle_at()`
- CAP_NET_ADMIN: Perform network related operations
- CAP_SYS_ADMIN: System administration operations

And many more:

- CAP_SYS_BOOT, CAP_SYS_CHROOT, CAP_SYS_TIME, CAP_SYSLOG, ...
OS concepts for isolation

libseccomp

- Interface to Linux kernel's syscall filtering mechanism
- Platform independent
- C or Python bindings available
- Operation on syscall filters:
  - reset, merge, exist_arch, add_arch, remove_arch, load, get_attr, set_attr, syscall_priority, add_rule, add_rule_exactly, export_pfc, export_bpf
- Filter action values:
  - KILL, ALLOW, TRAP, ERRNO(x), TRACE(x)
OS concepts for isolation

libseccomp

• Workflow:
  - Initialize the library
  - Add / Define rules
  - Load the filter into the kernel

```python
import sys
from seccomp import *
sf = SyscallFilter (defaction = KILL)
sf.add_rule (ALLOW, "open")
sf.add_rule (ALLOW, "read", Arg (0, EQ, sys.stdin))
sf.load()
```
OS concepts for isolation

Control groups

System resources ($sudo lssubsyss$):

- **blkio**: Sets I/O limits on block devices
- **cpu**: Limits of available CPU time
- **cpuacct**: Automated reports on CPU usage
- **cpuset**: Assign individual CPUs to concrete tasks
- **devices**: Allows / denies access to devices
- **freezer**: Suspend / Resume tasks in cgroups
- **memory**: Limits on memory use
- **net_cls**: Tagging network packets
- **net_prio**: Dynamically set network traffic per network iface
- **ns**: Namespace subsystem
OS concepts for isolation

Control groups

Configuration: /etc/cgconfig.conf

```plaintext

group restrictmem {
    memory {
        memory.limit_in_bytes = 200m;
    }
}
```

- service cgconfig restart
- Lscgroup
- cgexec -g memory:restrictmem virsh ...

Sandbox as development / application platform

Gnome Sandboxed Applications

Goals:
- Enable 3rd parties to create / distribute applications working on multiple distributions
- Run the applications with the as little as possible access to the hosting system

Technologies:
- Cgroups, Linux namespaces, SELinux, kdbus, Wayland
Sandbox as development / application platform

Gnome Sandboxed Applications

Concepts:
- **Runtime**
  - Well-defined environment application can run in
  - By application launch runtime is mounted at `/usr`
- **SDK**
  - Special runtime
  - Contains parts for building and packaging the application (devel files, compilers etc.)
- **Application bundle**
  - Metadata
  - Actual application files (mounted under `/app`)
  - List of exported files (desktop file, icon etc.)
  - Building an application = install SDK, then build the application against it using `--prefix=/app`

Installation locations (both for runtime & application bundles):
- Per-user: `$HOME/.local/share/xdg-app/`
- System-wide: `/var/xdg-app`
Sandbox as development / application platform

Gnome Sandboxed Applications

Tools – Xdg-App

- Uses OSTree to install and update runtimes and applications
- Ensures isolation of application from hosting OS
- Runtime for the application is provided via /usr specification
- Contains tools for building and distribution of application bundles
Sandbox as development / application platform

Gnome Sandboxed Applications

Xdg-App Sandbox details

- All processes run as the user with no capabilities
- Filesystem namespace
  - / private tmpfs, /usr bind mount of the runtime, /app bind mount of the application
  - /proc shows only processes within sandbox
  - /sys read-only mount of host's /sys
  - /dev/[f,n]ull, /dev/[,u]random, /dev/zero, /dev/tty
  - /dev/shm is private tmpfs
  - /run/user/$uid is set up and $XDG_RUNTIME_DIR points to it
- Host fonts are bind mounted to /run/host/fonts
- All mounts are nosuid, nodev (except /dev and /dev/pts), and read-only (except whitelisted locations)
Sandbox as development / application platform

Revisit the way how to compose Linux systems

Approaches:
- “Toolbox” Upstream creates tarball, downstream builds / distributes it
- “Instant” Application is possible quickly to find and download. Vendor is responsible for keeping the application updated

Goals:
- Efficient way for sw packaging (regardless of complexity)
- Ease of installation / maintenance (regardless of the concrete distribution)
- Universal solution (applicable to whole OS, containers, end user apps, devel toolkits)
- Signed (i.e. trusted) images
Sandbox as development / application platform

Revisit the way how to compose Linux systems

Proposal:
- Use selected concepts of btrfs (e.g. sub-volumes) and Linux filesystem namespaces

- Introduce naming scheme for btrfs sub-volumes:
  - `usr:<vendorid>:<architecture>:<version>`
  - `root:<name>:<vendorid>:<architecture>`
  - `runtime:<vendorid>:<architecture>:<version>`
  - `framework:<vendorid>:<architecture>:<version>`
  - `app:<vendorid>:<runtime>:<architecture>:<version>`
  - `home:<user>:<uid>:<gid>`
Sandbox as development / application platform

Revisit the way how to compose Linux systems

Workflow:

- Booting OS:
  - Mount root directory from one of root sub-volumes
  - Mount /usr from matching usr sub-volumes
- Enumerate system users = traverse the list of home sub-volumes
- After user's login his home directory from home sub-volume is mounted
- Application launch:
  - Create new filesystem namespace
  - Mount app sub-volume to /opt/<vendorid>/
  - Mount corresponding runtime sub-volume of that application to /usr
  - Set /home/$USER
- Developing against specific runtime:
  - Obtain the framework sub-volume
  - Create new filesystem namespace
  - Mount /usr to the framework sub-volume
Sandbox as development / application platform

CS50 Sandbox

Architecture:

• CS50 Sandbox
  – Asynchronous HTTP server
  – Offers clients to (non)-interactively execute programs written in any compiled and / or interpreted language
  – Tightly controlled, resource constrained environment
  – HTTP-based API takes files, command lines, and other inputs
  – Returns stdout, stderr + exitcode

• CS50 Run
  – Web based code editor
  – Enables browser based code writing in any (compiled / interpreted) language

• CS50 Check – autograding framework to support both black-box and white-box testing

• CS50 Sandbox Readme
Sandbox as development / application platform

CS50 Sandbox

Design goals:

• Support (non)-interactive execution of a program written in any language
• Support build tools like ant and make
• Support stdin, stdout, stderr
• Limit resource consumption (CPU cycles, fds, memory, disk space)
• Restrict network access
• Prevent reading / writing of unsandboxed files
• Prevent DoSes
Thank you for your attention.

Any questions?
References

- Perl taint mode:
  - http://perldoc.perl.org/perlsec.html

- Perl Safe module:
  - http://perldoc.perl.org/Safe.html
  - Predefined opcode tags:
    - http://perldoc.perl.org/Opcode.html#Predefined-Opcode-Tags
References #2:

- **Java Sandbox model:**
  - [https://docs.oracle.com/javase/7/docs/technotes/guides/security/spec/security-spec.doc1.html](https://docs.oracle.com/javase/7/docs/technotes/guides/security/spec/security-spec.doc1.html)

- **Java v2 Sandbox model**

- **SELinux Sandbox:**
  - [https://dwalsh.fedorapeople.org/SELinux/Presentations/sandbox.pdf](https://dwalsh.fedorapeople.org/SELinux/Presentations/sandbox.pdf)
References #3:

- **Libseccomp:**
  - Repository:
    - https://github.com/seccomp/libseccomp
  - Python bindings:
References #4:

• Gnome Sandboxed Applications:
  – Main wiki page:
    • https://wiki.gnome.org/Projects/SandboxedApps
  – Xdg-App repository:
    • https://github.com/alexlarsson/xdg-app
  – Xdg-App Sandbox design overview:
    • https://wiki.gnome.org/Projects/SandboxedApps/Sandbox
References #5:

- Systemd proposal – revisit the way how to compose Linux system:
  - http://0pointer.net/blog/revisiting-how-we-put-together-linux-systems.html

- Harvard University CS50 Sandbox:
  - Design overview:
  - Installation & usage guide:
    - http://cs.harvard.edu/malan/sandbox50/readme.html