THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

Combining WrapFS and eBPF to provide a lightweight Filesystem Sandboxing framework

Ashish Bijlani, PhD Student, Georgia Tech

@ashishbijlani

THE LINUX FOUNDATION

Goal

- Run untrusted third-party code from the internet in a safe manner.

- Examples:

  - Third-party web browser plugins,
  - Evaluate a Machine Learning model, etc.

Code vs Security Techniques

Known Good Code

Unknown/Untrusted Code

Known Bad Code

Whitelisting

Sandboxing

Blacklisting, Signatures

A safe, isolated, and controlled execution environment.

File System Sandboxing

- Restrict access to sensitive data when executing untrusted binaries.
  - Enforce security policies
    - e.g., do not allow access to ~/.ssh/id_rsa*
  - Follow the principle of least privilege
    - e.g., only allow access to *.pdf to a PDF reader

FS Sandboxing: Existing Techniques

File System Sandboxing Techniques	Dynamic Policies	Unprivileged Users	Fine-grained Control	Security Needs	Performance Overhead
UNIX DAC	X	✓	X	Inadequate	–

Discretionary Access Control (DAC)

	File-A	File-B	Untrusted App
Alice	*rwx*	*r-x*	*File-A: rw*
Bob	*r—*	*rw-*	*File-B: rw*

FS Sandboxing: Existing Techniques

File System Sandboxing Techniques	Dynamic Policies	Unprivileged Users	Fine-grained Control	Security Needs	Performance Overhead
UNIX DAC	X	✓	X	Inadequate	–
SELinux (MAC)	✓	**X**	✓	✓	–

Assign Mandatory Access Control (MAC) labels

```
$ ls -dZ - /etc/
drwxr-xr-x. root root system_u:object_r:etc_t:s0       /etc
```

FS Sandboxing: Existing Techniques

File System Sandboxing Techniques	Dynamic Policies	Unprivileged Users	Fine-grained Control	Security Needs	Performance Overhead
UNIX DAC	X	✓	X	Inadequate	–
SELinux (MAC)	✓	X	✓	✓	–
Chroot/ Namespaces	**X**	**X**	**X**	**Isolation**	**–**

Isolated file system mount point

$ unshare -m /bin/bash

FS Sandboxing: Existing Techniques

File System Sandboxing Techniques	Dynamic Policies	Unprivileged Users	Fine-grained Control	Security Needs	Performance Overhead
UNIX DAC	X	✓	X	Inadequate	–
SELinux (MAC)	✓	X	✓	✓	–
Chroot/Namespaces	X	X	X	Isolation	–
LD_PRELOAD	✓	✓	**X**	**Bypass**	**Low**

File system call wrappers in C library

$ LD_PRELOAD=./wrapper.so /bin/bash

e.g., ssize_t write_wrapper(int fd, …) { return -EACCES; }

Bypass: directly invoke system calls, mmap() I/O

FS Sandboxing: Existing Techniques

File System Sandboxing Techniques	Dynamic Policies	Unprivileged Users	Fine-grained Control	Security Needs	Performance Overhead
UNIX DAC	X	✓	X	Inadequate	–
SELinux (MAC)	✓	X	✓	✓	–
Chroot/Namespaces	X	X	X	Isolation	–
LD_PRELOAD	✓	✓	X	Bypass	Low
PTRACE	✓	✓	**X**	**TOCTTOU**	**< 50%**

Trace system calls and check arguments

ptrace(PTRACE_TRACEME,…); ptrace(PTRACE_PEEKUSER,…);
ptrace(GET/SETREGS)

TOCTTOU: arguments could be changed on-the-fly

FS Sandboxing: Existing Techniques

File System Sandboxing Techniques	Dynamic Policies	Unprivileged Users	Fine-grained Control	Security Needs	Performance Overhead
UNIX DAC	X	✓	X	Inadequate	–
SELinux (MAC)	✓	X	✓	✓	–
Chroot/Namespaces	X	X	X	Isolation	–
LD_PRELOAD	✓	✓	X	Bypass	Low
PTRACE	✓	✓	X	TOCTTOU	< 50%
FUSE	✓	✓	✓	✓	**< 80%**

All FS operations in user space

e.g., ssize_t write_wrapper(int fd, …) { return -EACCES; }

FS Sandboxing: motivation

File System Sandboxing Techniques	Dynamic Policies	Unprivileged Users	Fine-grained Control	Security Needs	Performance Overhead
UNIX DAC	X	✓	X	Inadequate	–
SELinux (MAC)	✓	X	✓	✓	–
Chroot/Namespaces	X	X	X	Isolation	–
LD_PRELOAD	✓	✓	X	Bypass	Low
PTRACE	✓	✓	X	TOCTTOU	< 50%
FUSE	✓	✓	✓	✓	< 80%
/* TODO */	✓	✓	✓	✓	**5-10% .**

Outline

- Motivation
- **Introduction**
- Key enabling technology
- Architecture
- Implementation
- Workflow
- Evaluation
- Use Cases

SandFS

- File system sandboxing framework
  - Unprivileged users and applications
  - Fine-grained access control
  - Dynamic (programmatic) custom security checks
  - Stackable (layered) protection
  - Low performance overhead

SandFS: FS sandboxing framework

```
$ sandfs -s sandfs.o -d /home/user /bin/bash
```

Non-root | Security Checks | Sandboxed Directory | Untrusted Application

eBPF code

Outline

- Motivation
- Introduction
- **Key enabling technology: eBPF**
- Architecture
- Implementation
- Workflow
- Evaluation
- Use Cases

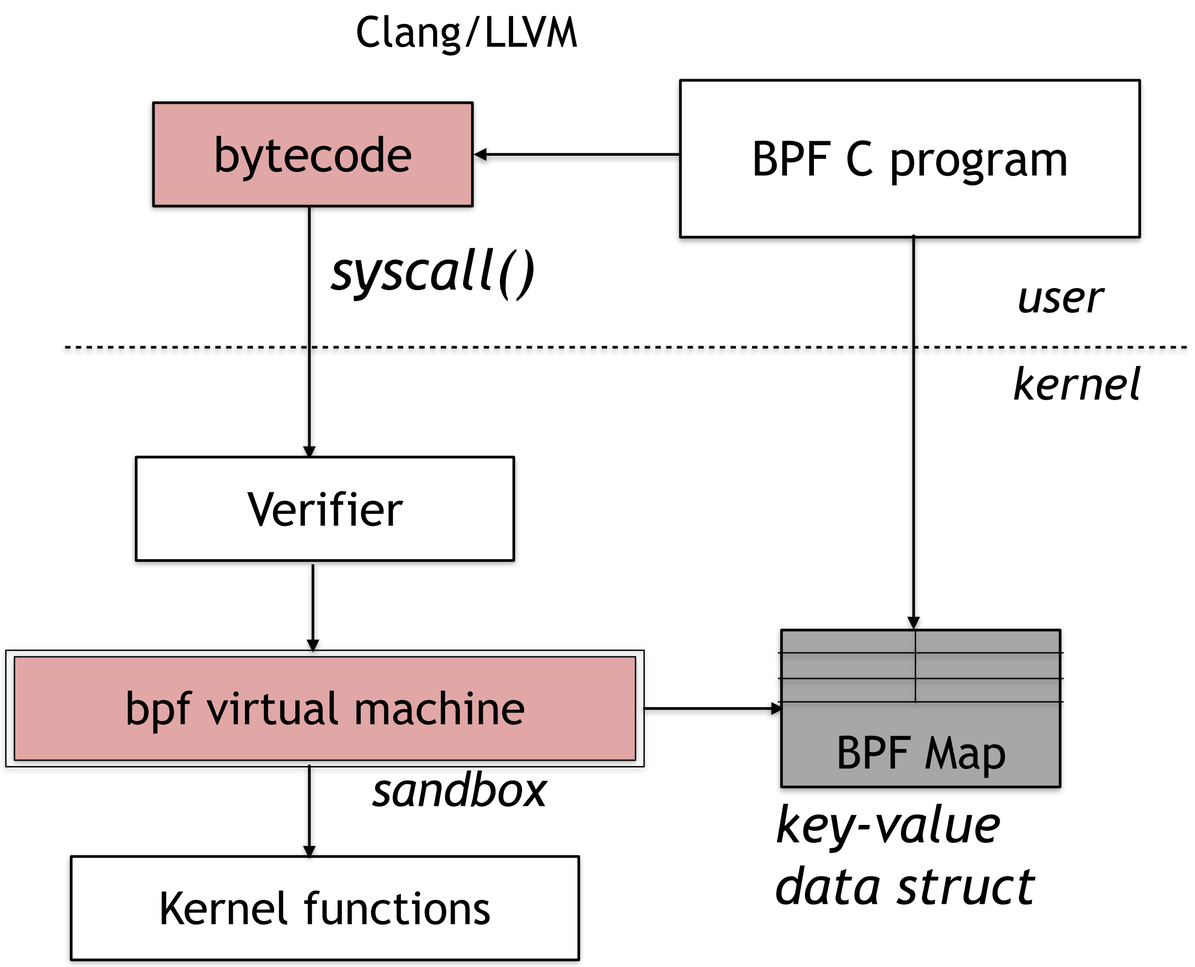
eBPF

- Berkeley Packet Filter (BPF)
  - Pseudo machine architecture for packet filtering

- eBPF extends BPF
  - Evolved as a generic kernel extension framework
  - Used by tracing, perf, and network subsystems

eBPF Overview

- Extensions written in C
- Compiled into BPF code, verified and loaded into kernel
- Execution under virtual machine runtime
- Shared BPF maps with user space



eBPF Example

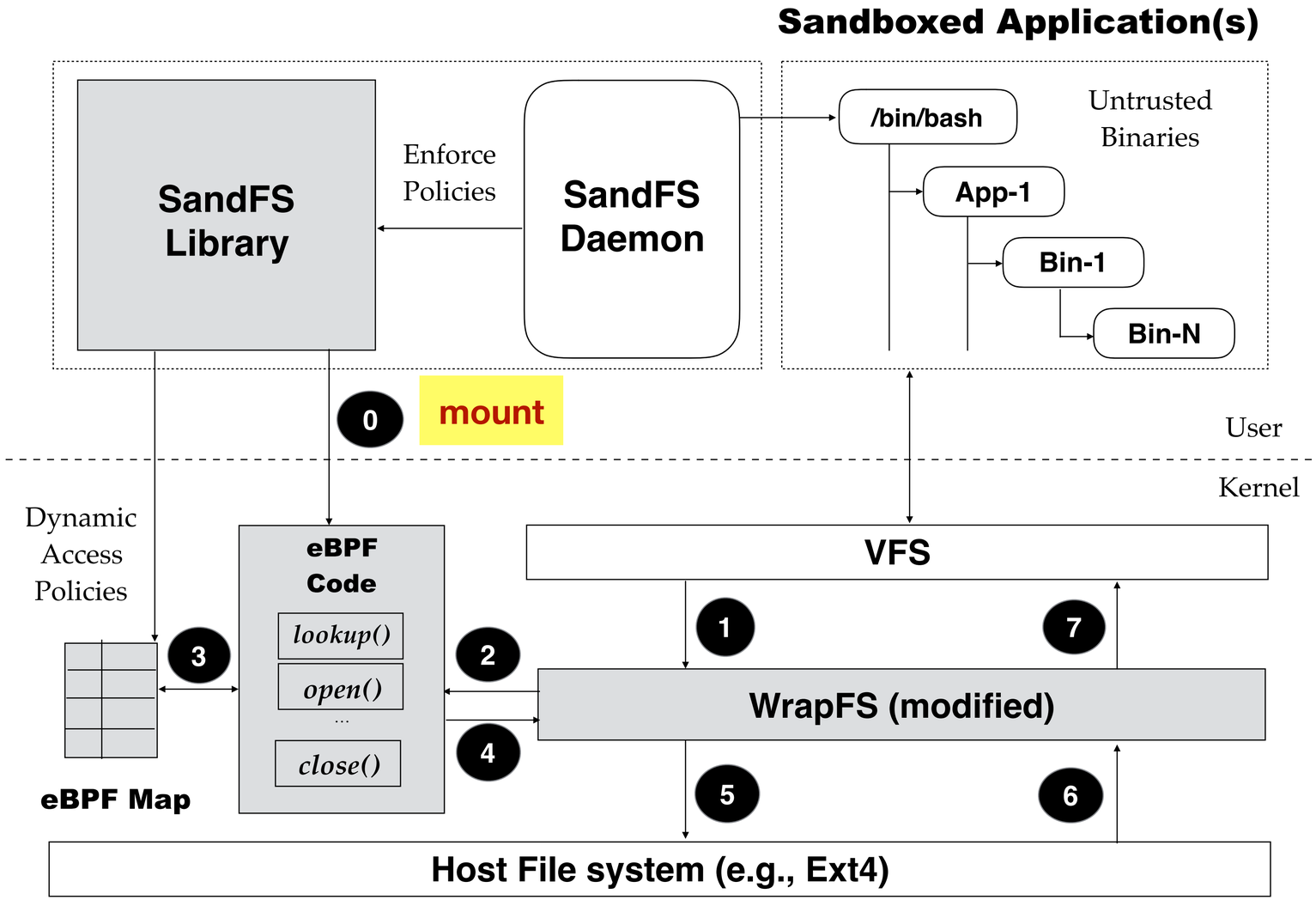
```
struct bpf_map_def map = {
  .type = BPF_MAP_TYPE_ARRAY,
  .key_size = sizeof(u32),
  .value_size = sizeof(u64),
  .max_entries = 1, // single element
};

// tracepoint/syscalls/sys_enter_open
int count_open(struct syscall *args) {
  u32 key = 0;
  u64 *val = bpf_map_lookup_elem(map, &key);
  if (val) __sync_fetch_and_add(val, 1);
}
```

Outline

- Motivation
- Introduction
- Key enabling technology: eBPF
- **Architecture**
- Implementation
- Workflow
- Evaluation
- Use Cases

SandFS: Architecture



SandFS: Example

```
1  int sandfs_lookup(void *args) {
2
3      /* get path */
4      char path[PATH_MAX];
5      ret = sandfs_bpf_read(args, PARAM0, path, PATH_MAX);
6      if (ret) return ret;
7
8      /* lookup in map if the path is marked as private */
9      u32 *val = bpf_map_lookup(&access_map, path);
10
11     /* example check: prohibit access to private files */
12     if (val) return -EACCES;
13
14     return 0; /* allow operation */
15 }
```

SandFS: Example

```
1  int sandfs_open(void *args) {
2
3      /* get mode */
4      u32 mode;
5      ret = sandfs_bpf_read(args, PARAM1, &mode, sizeof(u32));
6      if (ret) return ret;
7
8      /* example check: file creation not supported */
9      if (mode & O_CREAT) return -EPERM;
10
11     /* example enforcement: rewrite arg to force RDONLY mode */
12     mode = O_RDONLY;
13     ret = sandfs_bpf_write(args, PARAM1, &mode, sizeof(u32));
14     if (ret) return ret;
15
16     return 0; /* allow access */
17 }
```

SandFS: Implementation

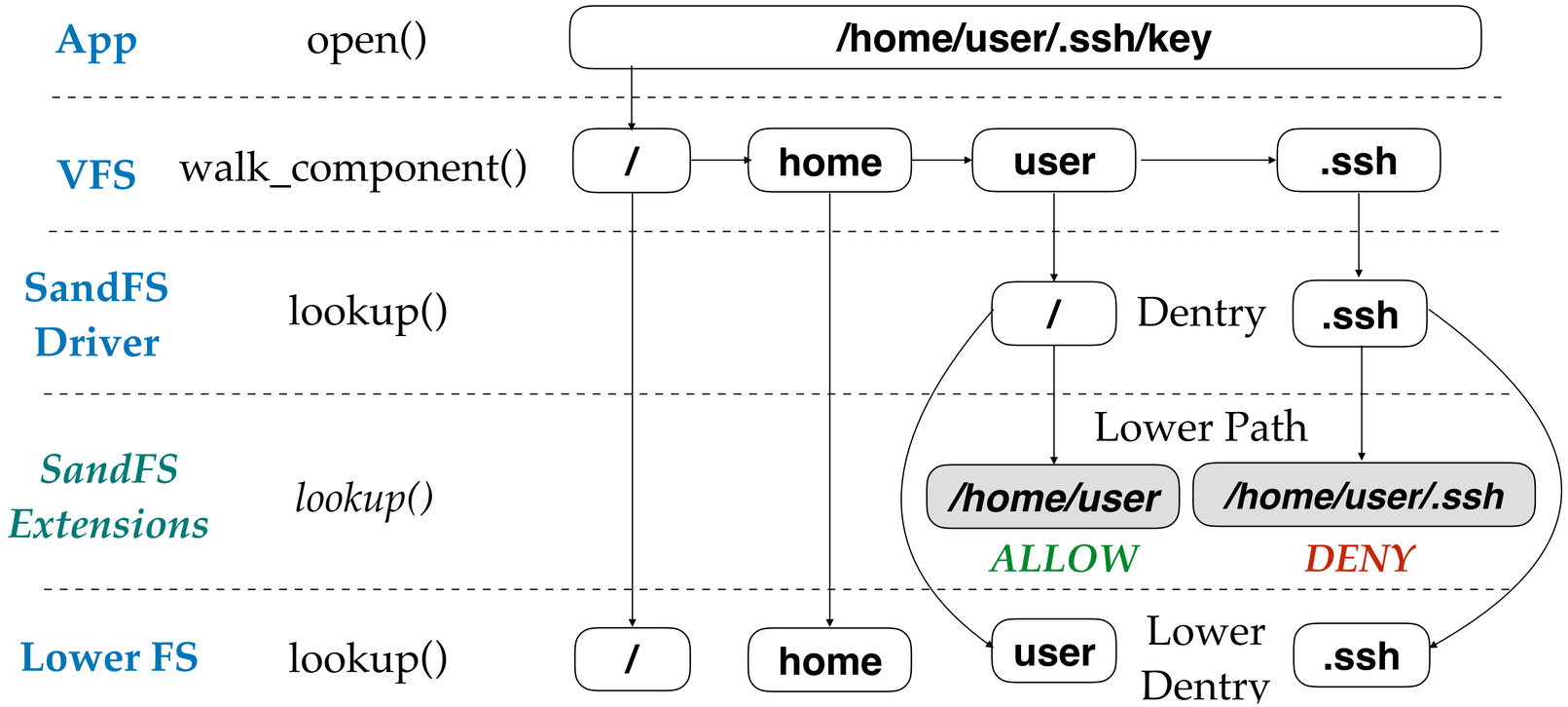
- SandFS driver based on WrapFS

  - Stackable file system wrapper layer
    - Does not perform I/O
    - Forwards request to lower FS (e.g., Ext4)

  - Limit num of stackable layers (no stack overflow)
  - Invoke SandFS extensions to enforce policies

SandFS: Workflow

Works directly with kernel objects, no TOCTTOU



Outline

- Motivation
- Introduction
- Key enabling technology: eBPF
- Architecture
- Implementation
- Workflow
- **Evaluation**
- Use Cases

SandFS: Evaluation

Intel Quad-Core i5-3550, 16GB RAM, SSD (EXT4)

Benchmark	**Time Taken (seconds)**		
	Native (Ext4)	**SandFS**	**Overhead (%)**
Compress (tar.gz) Linux Kernel 4.17	**61.05**	**63.84**	**4.57**
Decompress (tar.gz) Linux Kernel 4.17	**5.13**	**5.63**	**9.75**
Compile (make -j4) Linux Kernel 4.17 (tinyconfig)	**27.15**	**29.67**	**9.28**

SandFS: Use cases

- Restricting access to private user data
  - e.g., hide .ssh keys
- Building secure applications
  - e.g., compartmentalize (Chrome browser)
- Hardening containers
  - e.g., stack layers of SandFS for custom checks

SandFS

- Source code available on GitHub.
  - https://sandfs.github.io
- Academic paper published
  - “A Lightweight and Fine-grained File System Sandboxing Framework” in APSys '18
- Related work with eBPF
  - “when eBPF meets FUSE” in OSS NA’18, LPC’18

Thank You!

ashish.bijlani@gatech.edu

www.linkedin.com/in/ashishbijlani


THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE