Improve Security of File System for Containers inBlackBox

Yongmao Luo, Zijian Zhang, Xincheng Xie

CONTENT

01 Background

02 Proposal & Beta Impl

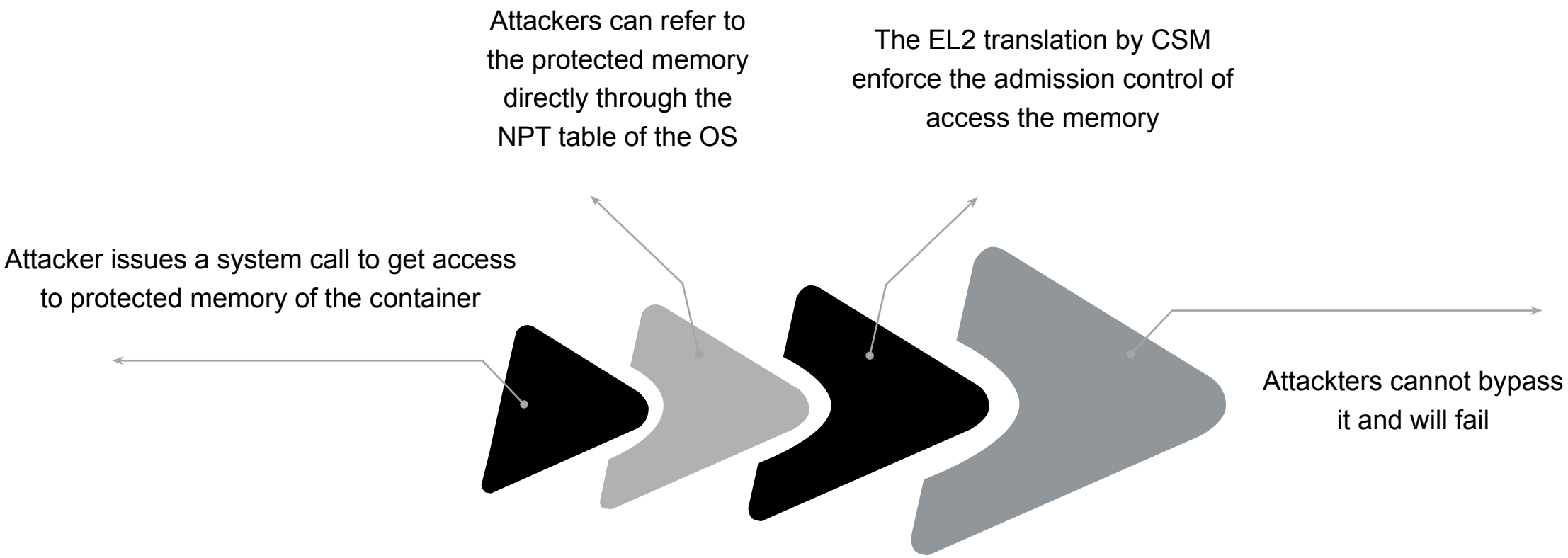
03 Demo

04 Result & Plan

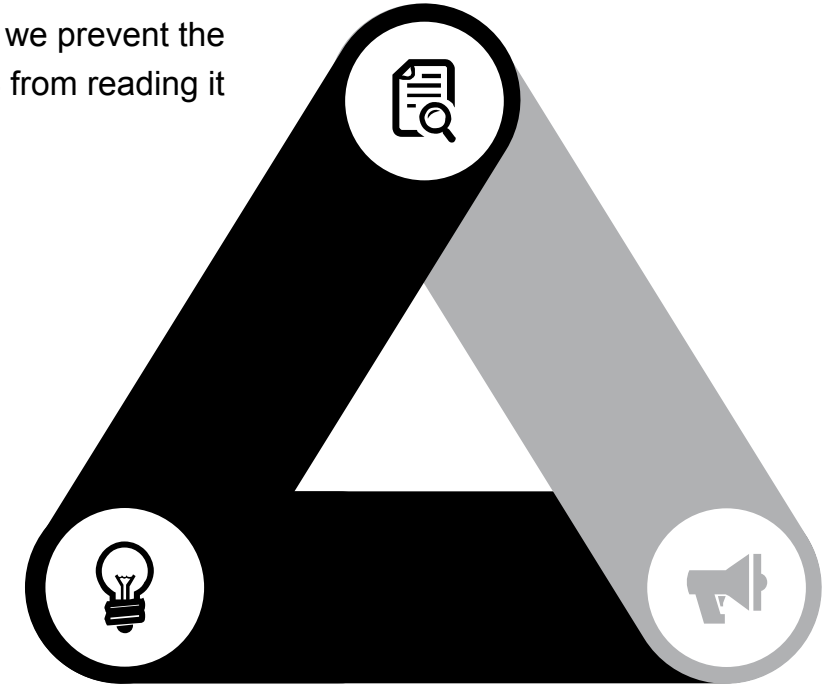
01

Background

Security provided by BlackBox



Security provided by BlackBox



Confidentiality

Yes, because we prevent the untrusted OS from reading it

Integrity

Yes, because we prevent the untrusted OS from writing it

Availability

No, because we rely on the OS functionality to manage resources

Files within a container can only be accessed through an OS's I/O facilities making access to a container's files inherently untrustworthy without additional protection. A userspace encrypted file system could potentially be used to provide transparent protection of file I/O, but this would likely signif-

02

Proposal & Beta Implementation

Encryption & Protected Memory

What to encrypt?

Newly created files

Newly created files by containers —
Private Files of Containers

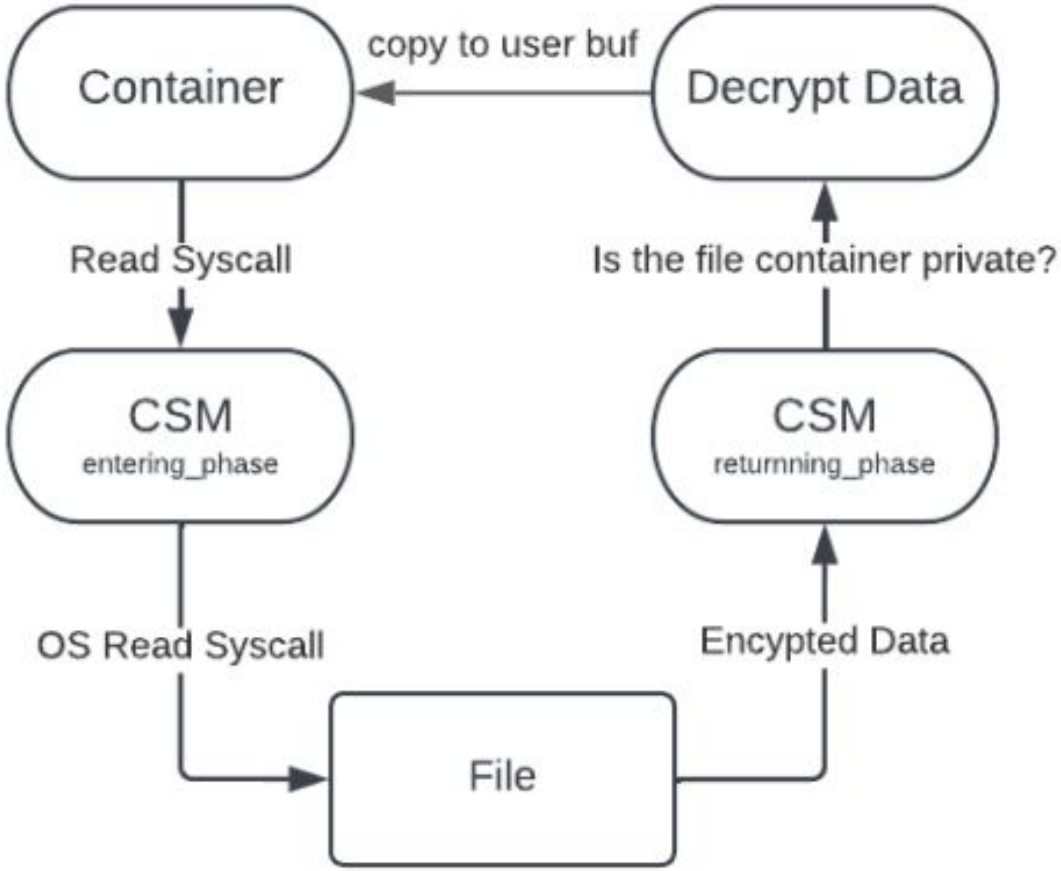


Figure 4: Read

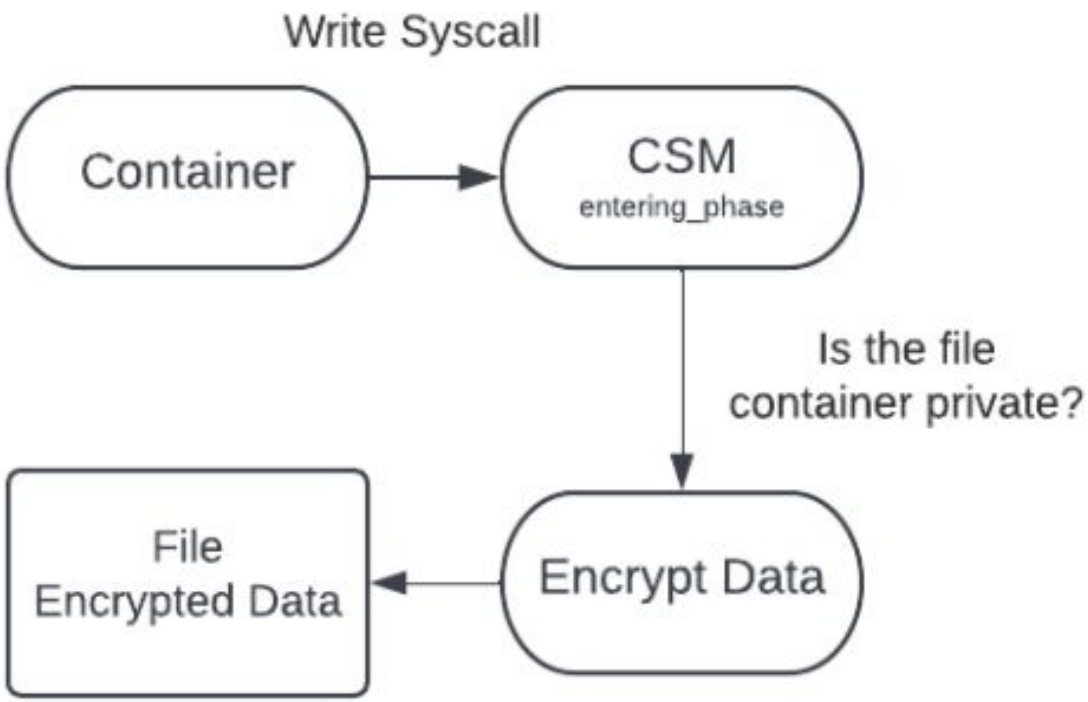
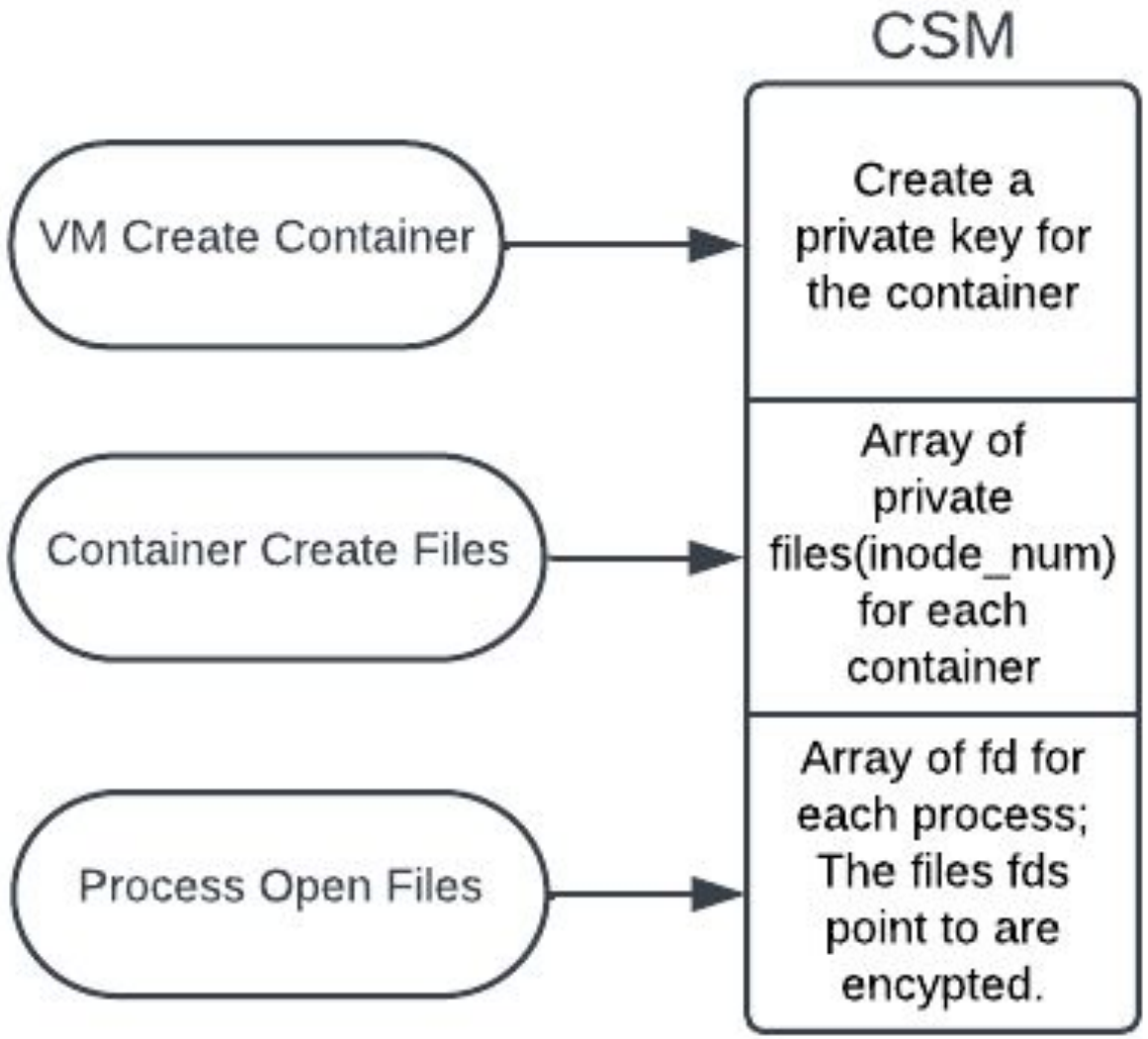


Figure 3: Write


CSM
VM Create Container
Create a private key for the container
Container Create Files
Array of private files(inode_num) for each container
Process Open Files
Array of fd for each process; The files fds point to are encypted.

```c
int hvc_process_fork(uint32_t parent_eidx, uint32_t new_eidx, struct task_struct *child);
int hvc_process_exec(uint32_t caller_eidx, uint32_t new_eidx, struct task_struct *exec_tsk);
int hvc_process_clone(uint32_t caller_eidx, struct task_struct *thread);
void hvc_process_exit(void);
void hvc_restarting_syscall(int scno);

void hvc_settid(u64 tid_addr, u64 tid);

void hvc_file_create(uint64_t eidx, char *path, unsigned long i_ino);
void hvc_file_open(uint64_t eidx, char *path, unsigned long i_ino, int fd);
void hvc_file_unlink(uint64_t eidx, char *path, unsigned long i_ino);
void hvc_file_close(uint64_t eidx, char *path, int fd);
```

```
/* Negative dentry, just create the file */
if (!dentry->d_inode && (open_flag & O_CREAT)) {
    file->f_mode |= FMODE_CREATED;
    audit_inode_child(dir_inode, dentry, AUDIT_TYPE_CHILD_CREATE);
    if (!dir_inode->i_op->create) {
        error = -EACCES;
        goto out_dput;
    }
    error = dir_inode->i_op->create(dir_inode, dentry, mode,
                                    open_flag & O_EXCL);
    if (error)
        goto out_dput;
    else if (eidx) {
        char *tmp_path = kmalloc(4096, GFP_KERNEL);
        char *path = dentry_path_raw(dentry, tmp_path, 4096);
        printk(KERN_DEBUG "lookup_open_create eidx = %lld file_path = %s i_ino = %ld\n",
        hvc_file_create(eidx, path, dentry->d_inode->i_ino);
        kfree(tmp_path);
    }
    fsnotify_create(dir_inode, dentry);
}
if (unlikely(create_error) && !dentry->d_inode) {
    error = create_error;
    goto out_dput;
}
```

```c
case HVC_FILE_CREATE: {
    u64 eidx = hr->regs[1];
    char *path = hr->regs[2];
    unsigned long i_ino = hr->regs[3];
    struct el2_enclave_task *e_task = get_current_enclave_task();
    struct el2_enclave_info* e_info = current_el2_enclave_info();
    int i;

    for (i = 0; i < MAX_ENCRYPTED_FILES; i++) {
        if (e_info->encrypted_ino_array[i] == -1) {
            e_info->encrypted_ino_array[i] = i_ino;
            printf("HVC_FILE_CREATE eidx = %lld file_path = %s i_ino = %ld\n", eidx, __el2_va(path), i_ino);
            printf("HVC_FILE_CREATE e_info->encrypted_ino_array[%d] = %d\n", i, i_ino);
            break;
        }
    }
    break;
}
```

03

Demo

https://youtu.be/qwg2D7RWX-c

04

Result & Plan

Result

1. Correctness.
   a. Processes within a container can correctly read the files
   b. Processes outside a container cannot correctly read the file content and fail to write.
   c. Compromised OS cannot correctly read the file content

2. Performance.
   a. file system benchmark: fio
   b. real application: LevelDB
   c. expectation: lower performance than unencrypted, but not significantly worse.

Basic Data Structures:

1. track files belongings
2. storing ephemeral keys for each container

Hypercalls for manipulating such data structures

1
First Week

Improve memory Utilization by deleting encrypted files generated by processes in container when closing the container

Adding cryptography library to CSM to improve the security of encrypted files

2
Second Week

Manipulate Read_write_wrapper to encrypt and decrypt data accordingly

3
Third Week

encrypt/decrypt in a more precise granularity

Testing to ensure the functionality

4
Fourth Week

Testing & final paper

5
Fifth Week

THANK YOU
For Your Listening