RV College of Engineering®

Operating System

Go, change the world

Topic: Implementing Custom Encryption System Call

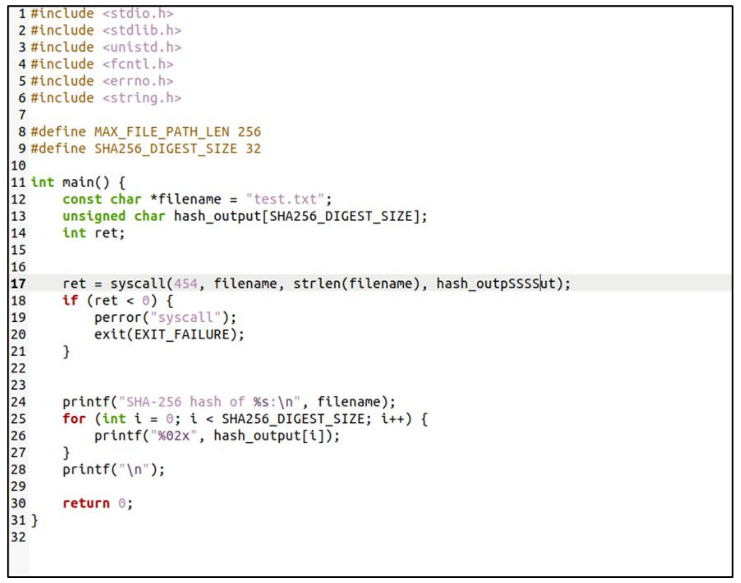
Objectives:

The kernel is the core code of the operating system. Usually when the computer starts the bootloader and the kernel are the first pieces of code to be loaded into the machine.

The kernel has control over the entire system, its memory, peripheral devices, processor and so on. User programs and applications can utilize the resources that the operating system makes available through system calls.

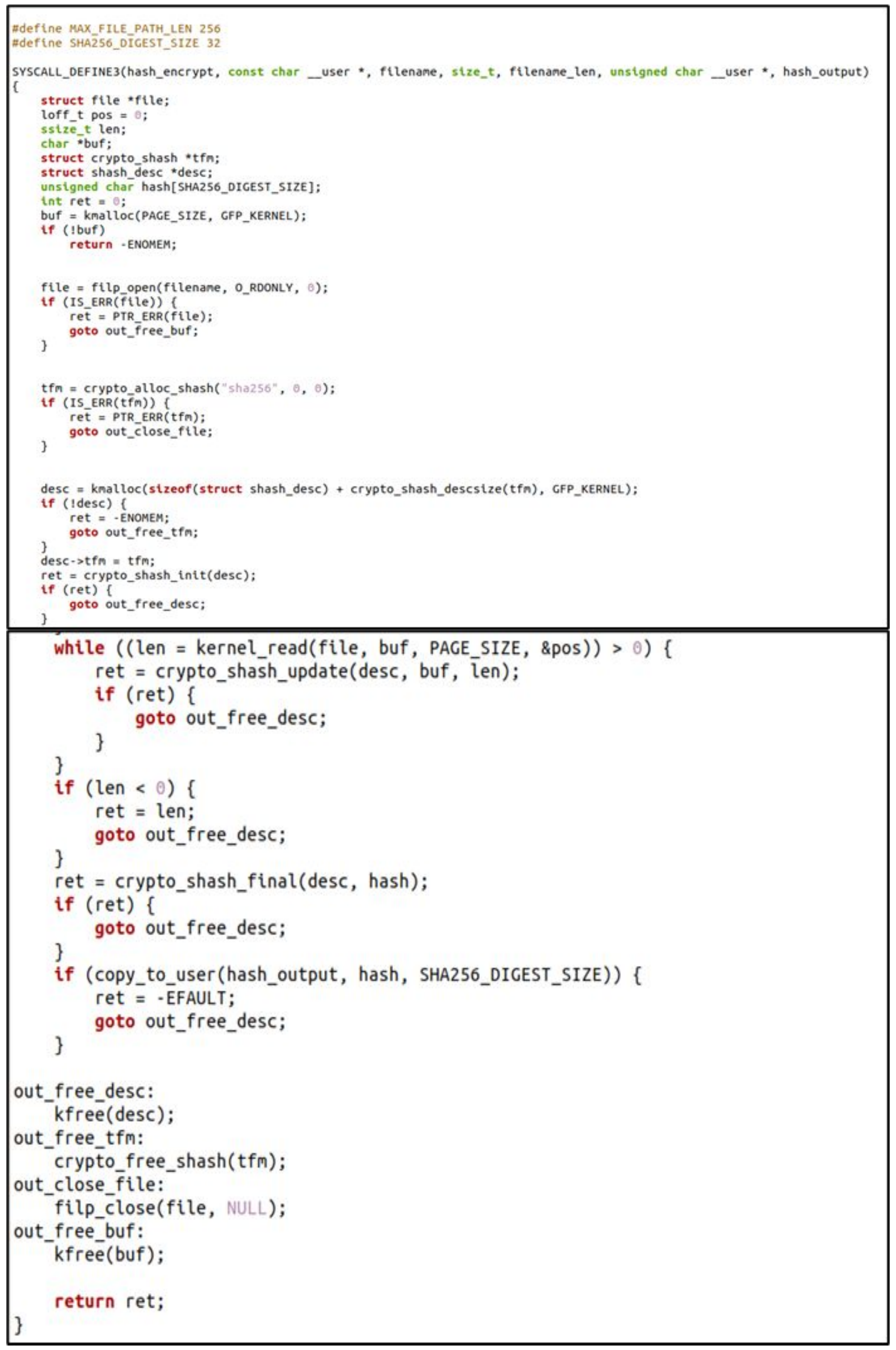
System calls are listed in the system call table and are executed in the privileged space of the kernel mode.

User Space Program Snippet



```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>

#define MAX_FILE_PATH_LEN 256
#define SHA256_DIGEST_SIZE 32

int main() {
    const char *filename = "test.txt";
    unsigned char hash_output[SHA256_DIGEST_SIZE];
    int ret;


    ret = syscall(454, filename, strlen(filename), hash_outpSSSSut);
    if (ret < 0) {
        perror("syscall");
        exit(EXIT_FAILURE);
    }


    printf("SHA-256 hash of %s:\n", filename);
    for (int i = 0; i < SHA256_DIGEST_SIZE; i++) {
        printf("%02x", hash_output[i]);
    }
    printf("\n");

    return 0;
}

```

Kernel Space System Call Implementation



```c
#define MAX_FILE_PATH_LEN 256
#define SHA256_DIGEST_SIZE 32

SYSCALL_DEFINE3(hash_encrypt, const char __user *, filename, size_t, filename_len, unsigned char __user *, hash_output)
{
    struct file *file;
    loff_t pos = 0;
    ssize_t len;
    char *buf;
    struct crypto_shash *tfm;
    struct shash_desc *desc;
    unsigned char hash[SHA256_DIGEST_SIZE];
    int ret = 0;
    buf = kmalloc(PAGE_SIZE, GFP_KERNEL);
    if (!buf)
        return -ENOMEM;


    file = filp_open(filename, O_RDONLY, 0);
    if (IS_ERR(file)) {
        ret = PTR_ERR(file);
        goto out_free_buf;
    }


    tfm = crypto_alloc_shash("sha256", 0, 0);
    if (IS_ERR(tfm)) {
        ret = PTR_ERR(tfm);
        goto out_close_file;
    }


    desc = kmalloc(sizeof(struct shash_desc) + crypto_shash_descsize(tfm), GFP_KERNEL);
    if (!desc) {
        ret = -ENOMEM;
        goto out_free_tfm;
    }
    desc->tfm = tfm;
    ret = crypto_shash_init(desc);
    if (ret) {
        goto out_free_desc;
    }
    while ((len = kernel_read(file, buf, PAGE_SIZE, &pos)) > 0) {
        ret = crypto_shash_update(desc, buf, len);
        if (ret) {
            goto out_free_desc;
        }
    }
    if (len < 0) {
        ret = len;
        goto out_free_desc;
    }
    ret = crypto_shash_final(desc, hash);
    if (ret) {
        goto out_free_desc;
    }
    if (copy_to_user(hash_output, hash, SHA256_DIGEST_SIZE)) {
        ret = -EFAULT;
        goto out_free_desc;
    }

out_free_desc:
    kfree(desc);
out_free_tfm:
    crypto_free_shash(tfm);
out_close_file:
    filp_close(file, NULL);
out_free_buf:
    kfree(buf);

    return ret;
}
```

Hashing:

Hashing is a fundamental concept in computer science and cryptography. It involves taking an input (or 'message') and applying a mathematical function to generate a fixed-size string of bytes, typically of a fixed length. This output is often referred to as the 'hash value' or 'hash code'. Hashing is widely used in various applications, including data storage, security, and digital signatures, among others.

Conclusions:

In conclusion, the development and implementation of a system for transparent file-level encryption and decryption within the operating system kernel represent a significant advancement in data security and privacy. By leveraging custom kernel modules, cryptographic algorithms, and system call interception mechanisms, the system enhances file security without imposing additional complexity on user-space applications.

The system's architecture seamlessly integrates encryption and decryption mechanisms into the kernel, ensuring that file operations are performed securely and transparently. Through rigorous testing and evaluation, the system demonstrates its effectiveness in protecting sensitive data from unauthorized access and tampering, while maintaining compatibility with existing file systems and applications.

Team: Vibhav Simha G (1RV22CS230),
Sriram D S (1RV22CS203)

Guidance under: Dr Jyoti Shetty , Professor, Dept of CSE, RVCE
Dr. Apoorva , Associate Professor, Dept of CSE, RVCE