

CIS5370-project2

prashant.ravi

January 2025

1 Demonstration

The Github repository contains a demo.mp4 to display the attack in video format and the followin image describes how one can attack the FLE executable.

[illegible]

Figure 1: Gaining Shell access as user on STACK.FLE executable

2 Question 1

Modified the time interceptor to override the gettimeofday syscall which gets loaded before libc due to LD_PRELOAD.

```
#define _GNU_SOURCE
#include <stdio.h>
#include <dlfcn.h>
#include <sys/time.h>
#include <time.h>

static int speed_factor = 1; // Adjust to modify speed
static struct timeval start_tv;
static struct timeval real_start_tv;
static int initialized = 0;

// Match the exact signature from /usr/include/sys/time.h
int gettimeofday(struct timeval *tv, void *tz) {
    static int (*real_gettimeofday)(struct timeval *, void *) = NULL;
```

```

if (!real_gettimeofday) {
    real_gettimeofday = dlsym(RTLD_NEXT, "gettimeofday");
}

if (!initialized) {
    real_gettimeofday(&real_start_tv, NULL);
    start_tv = real_start_tv;
    initialized = 1;
}

int ret = real_gettimeofday(tv, tz);

if (ret == 0) {
    // Compute elapsed time from real start time
    long elapsed_sec = tv->tv_sec - real_start_tv.tv_sec;
    long elapsed_usec = tv->tv_usec - real_start_tv.tv_usec;

    if (elapsed_usec < 0) {
        elapsed_sec--;
        elapsed_usec += 1000000;
    }

    // Apply speed factor
    elapsed_sec *= speed_factor;
    elapsed_usec *= speed_factor;

    if (elapsed_usec >= 1000000) {
        elapsed_sec += elapsed_usec / 1000000;
        elapsed_usec %= 1000000;
    }

    // Return modified time
    tv->tv_sec = start_tv.tv_sec + elapsed_sec;
    tv->tv_usec = start_tv.tv_usec + elapsed_usec;

    if (tv->tv_usec >= 1000000) {
        tv->tv_sec += tv->tv_usec / 1000000;
        tv->tv_usec %= 1000000;
    }
}

return ret;
}

```

3 How This Works Without Modifying libc

The standard libc is still loaded, but when the game calls `gettimeofday`, it gets redirected to our custom function before reaching libc. Uses `dlsym(RTLD_NEXT, "gettimeofday")` to call the real function. This ensures we still get the real time from the actual libc function. We modify the return value only after calling the real `gettimeofday`. libc remains untouched. We are not replacing libc itself. The game still links dynamically to the system's standard libc. Our shared library is just injected at runtime, affecting only this process. When the game starts, it normally calls `gettimeofday` from libc. With `LD_PRELOAD=./time_interceptor.so`, the dynamic linker redirects calls to our custom version. Calls the real `gettimeofday` via `dlsym(RTLD_NEXT, "gettimeofday")`. Modifies the time to speed up or slow down the game. Returns the modified time.

CIS5370-project2

prashant.ravi

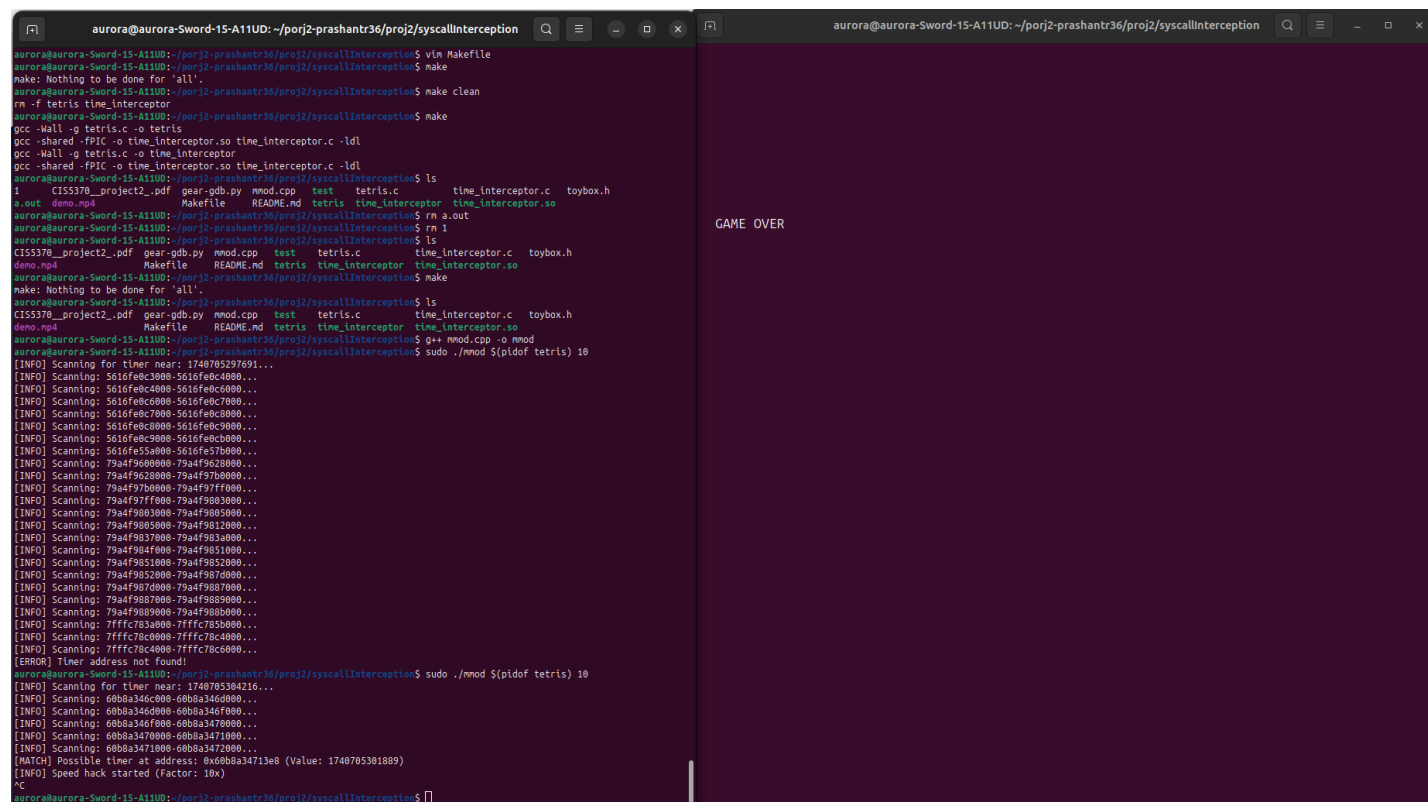
January 2025

1 Demonstration of Tetris speed using MemoryModifier

The Github repository contains a demo2.mp4 to display the attack in video format and the following image describes how one can attack the game score in Tetris.

To run the program alongside the game running in another terminal we dynamically modify the speedup factor at runtime.

```
$sudo ./mmod $(pidof tetris) 10
```



The image shows two terminal windows side-by-side. The left window is a terminal with a dark background and light text, showing the execution of a program named 'time_interceptor.c'. The program is compiled with 'gcc' and run with 'sudo ./mmod \$(pidof tetris) 10'. The output shows a list of memory addresses being scanned, with the first address being '0x00000000'. The right window is a terminal with a dark background and light text, showing the execution of the Tetris game. The game is running, and the text 'GAME OVER' is visible on the screen.

Figure 1: Attacking the Tetris game by writing a program that scans for the epoch address

2 Question 1

Modified the code to keep track of memory addresses that have been matched already and narrows down the search every time 's|value;' is manually entered or can automatically do this in a while loop to brute force

search all timestamps. Finally there will be only 1 matching memory address that can be updated from the command line which will reflect in the game as well. The previous iterative search was too slow and inaccurate because it doesn't remember the memory address matches across searches. The modification provided will search for the new values and also discard the previous matches that remained the same in the match.

As we can see in the demo video that the originally and then we search for this timer value in the memory of the Tetris game. We search for all the memory address of the time that have been modified with ± 5000 millesconds of the current time of day and then run a thread to constantly update the timer value with a desired speedup factor.

Below are the modifications made in the code that discards the excessive memory locations in the remain hashmap and retrieve the desired memory addresses that will be modified by the 'w 20' instructions from the command line.

3 Source Code changes

```
// Scan memory for possible timer addresses
uintptr_t find_timer_address(int pid, uint64_t expected_value) {
    std::string memfile = "/proc/" + std::to_string(pid) + "/mem";
    std::string mapsfile = "/proc/" + std::to_string(pid) + "/maps";

    std::ifstream maps(mapsfile);
    if (!maps.is_open()) {
        std::cerr << "Cannot open " << mapsfile << std::endl;
        return 0;
    }

    int fd = open(memfile.c_str(), O_RDONLY);
    if (fd < 0) {
        perror("open");
        return 0;
    }

    std::string line;
    while (std::getline(maps, line)) {
        uintptr_t start, end;
        char perms[5];

        if (sscanf(line.c_str(), "%lx-%lx %4s", &start, &end, perms) != 3)
            continue;

        // Scan only readable and writable memory regions
        if (perms[0] != 'r') continue;

        std::cout << "[INFO] Scanning: " << std::hex << start << "-" << end << std::dec << "..." <<
            std::endl;

        size_t size = end - start;
        std::vector<uint8_t> buffer(size);
        lseek(fd, start, SEEK_SET);
        read(fd, buffer.data(), size);

        // Scan for timestamp (allow a wider range)
        for (size_t i = 0; i < size - sizeof(uint64_t); i++) {
            uint64_t *val = reinterpret_cast<uint64_t *>(&buffer[i]);
            if (*val >= expected_value - 5000 && *val <= expected_value + 5000) {
```

```

        uintptr_t addr = start + i;
        std::cout << "[MATCH] Possible timer at address: 0x" << std::hex << addr
                    << std::dec << " (Value: " << *val << ")\n";
        close(fd);
        return addr; // Return first match
    }
}

close(fd);
return 0;
}

// Modify the timer value dynamically
void modify_timer(int pid, uintptr_t address, double speed_factor) {
    std::string memfile = "/proc/" + std::to_string(pid) + "/mem";

    int fd = open(memfile.c_str(), O_RDWR);
    if (fd < 0) {
        perror("open");
        return;
    }

    uint64_t start_time, new_time;
    pread(fd, &start_time, sizeof(start_time), address); // Read original time

    std::cout << "[INFO] Speed hack started (Factor: " << speed_factor << "x)\n";

    while (true) {
        uint64_t real_time = std::chrono::duration_cast<std::chrono::milliseconds>(
            std::chrono::system_clock::now().time_since_epoch()
        ).count();
        new_time = start_time + ((real_time - start_time) * speed_factor);

        // Write the modified time back
        pwrite(fd, &new_time, sizeof(new_time), address);

        // Sleep briefly to avoid excessive CPU usage
        std::this_thread::sleep_for(std::chrono::milliseconds(10));
    }

    close(fd);
}

```
