CSE 127 Discussion Week 4 - Side Channels

Agenda

PA2

- Due 2/1/22
- Side channel attacks
- Two parts:
  - Memory attacks
  - Timing attacks

Background

What do you mean “channel”?

- In this context, a channel is a means of conveying information
- For example:
  - Consider a function that checks passwords

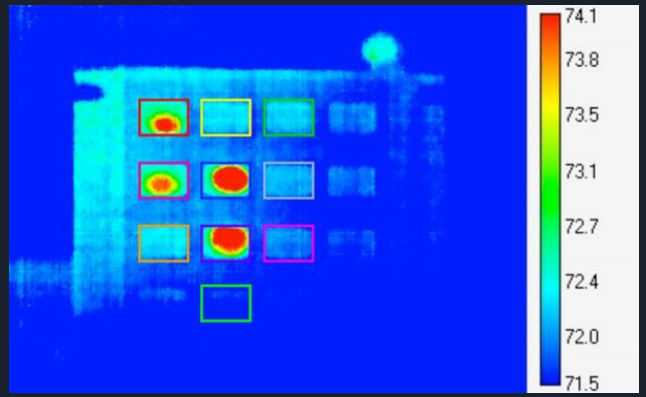
    `f(x): x -> { True, False }`
  - The intended information channel of the function is only the output, true or false
- Ideally, a user passes `x` to `f`, and may only observe `f(x)`

What's a side channel?

- In reality, `f` would need to be implemented on a physical device
- A side channel is any channel of information produced as a side-effect of conveying information along the primary/intended channel

Examples:

- Thermal
  - infrared pictures of pin pads can detect pressed keys



Other examples of side channels

- Timing
  - Is the output produced in the same amount of time for each input?
- Memory
  - Is memory accessed the same way in all cases?

PA2: Side Channel Attacks

Assignment Overview

- Two part assignment on side channels
  - memhack (memory-based side channel)
  - timehack (timing-based side channel)
- in both the goal is to programmatically guess the password checked by check_pass in sysapp.c
- Rubric:
  - Memhack (8pts) + Writeup (2pts)
  - Timehack (8pts) + Writeup (2pts)

Assignment Structure

- You are given a VM image with the starter files:
  - memhack.c
  - timehack.c
  - sysapp.c
- Sysapp.c is library code that both files import
  - It contains side channel vulnerabilities for you to exploit
  - DO NOT MODIFY THIS FILE

sysapp.c

- password to check (*`pass`) is passed by reference
- `check_pass` loops over characters checking against true password sequentially
- `correct_pass` is static in the given vm, but its value will change for grading so solution should generalize
- `delay` is added to make time hack more feasible
- solution should call `hack_system` on the password when it is found

```
void delay() {
    int j, q;
    for (j = 0; j < 100; j++) {
        q = q + j;
    }
}

int check_pass(char *pass) {
    int i;
    for (i = 0; i <= strlen(correct_pass); i++) {
        delay();  // artificial delay added for timehack
        if (pass[i] != correct_pass[i])
            return 0;
    }
    return 1;
};

void hack_system(char *correct_pass) {
    if (check_pass(correct_pass)) {
        printf("OK: You have found correct password: '%s'\n", correct_pass);
        printf("OK: Congratulations!\n");
        exit(0);
    } else {
        printf("FAIL: The password is not correct! You have failed\n");
        exit(3);
    };
};
```

memhack.c

- You are given a buffer of memory that will cause a segfault if the program tries to access certain bytes
- The code on the right (given in the starter) is an example method that shows you how to catch segfaults in your program.

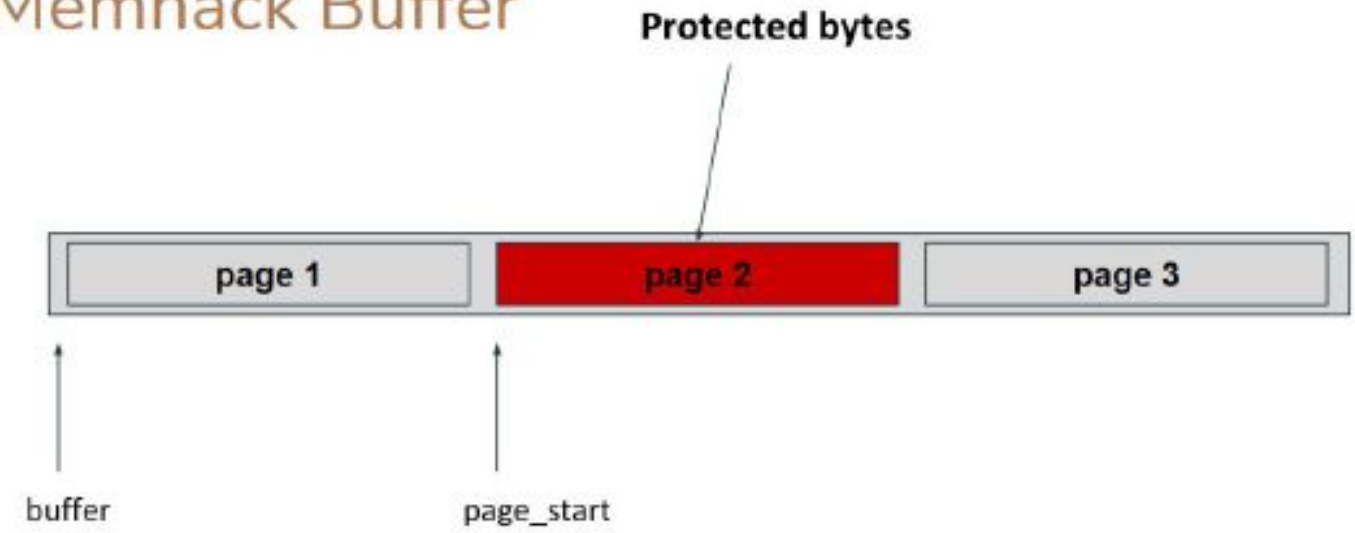
```c
int demonstrate_signals() {
    char *buf = page_start;

    // this call arranges that _if_ there is a SEGV fault in the future
    // (anywhere in the program) then control will transfer directly to this
    // point with sigsetjmp returning 1
    if (sigsetjmp(jumpout, 1) == 1)
        return 1; // we had a SEGV

    signal(SIGSEGV, SIG_DFL);
    signal(SIGSEGV, &handle_SEGV);

    // We will now cause a fault to happen
    *buf = 0;
    return 0;
}
```

Memhack Buffer



Things to Think About

- You have the ability to:
  - Set access rights to memory
  - Intercept all segfault signals
- Key features of the password checker:
  - Takes argument by reference
  - Checks characters sequentially
  - Short circuits on the first invalid character
- How can you utilize the above factors to create a side channel?

Hints

- Referencing protected memory bits will raise a fault
  - How can you use this to find a correct guess?
- Example: The password is "hello"

- check_pass(my_guess) causes a fault. Why?

- check_pass(my_guess) does not fault and returns 0. Why?

Catching faults

- Use sigsetjmp/siglongjmp
- sigsetjmp
  - Sets jump point for siglongjmp to jump to later
  - Returns 0 when you call it to set as the returning point
  - Returns non-zero value when it returns via a call to siglongjmp()
- Siglongjmp
  - Used to return to the point at which sigsetjmp was called
  - Avoid calling sigsetjmp in a helper function, because if the function sigsetjmp was called from returns before siglongjmp is called there is undefined behavior

Catching faults

- ```
  signal(SIGSEGV, SIG_DFL);
  signal(SIGSEGV, &handle_SEGV);
  ```

- This tells the system that whenever it hits a SIGSEGV segfault to call the function handle_SEGV()
- There are two calls because the documentation requires the signal to be set to default (SIG_DFL) before being set to a handler function
  ```

timehack.c

- Execution time of check_pass depends on how many characters you guess correctly
- Rdtsc returns processor cycle count as a long
  - Treat as a running timer
  - Use a timer by calling before and after check_pass, and find the difference in cycles
- There may be a lot of noise with each check_pass call, so you need multiple samples

Dealing with noise

- DON’T use printf’s in the code, they cause huge variances in execution time
- Use the median, not the mean for multiple trials
  - Outliers are every extreme, the mean will be affected
  - qsort may be helpful
- If time is not continuing to increase as you progress through characters, then you probably made an incorrect guess earlier

Good Luck!

Due Date: Tuesday, February 1st, 6:00PM