CSE 127 Discussion Week 4 – Side Channels

Agenda

- PA2 – released 19th April, due 28th April
- Based on side channel attacks
  - Memory attacks
  - Timing attacks

What is a side channel attack

- A side-channel attack is **a security exploit that aims to gather information from or influence the program execution of a system by measuring or exploiting indirect effects of the system or its hardware** -- rather than targeting the program or its code directly.

PA2 : Side Channel Attacks

Assignment Overview

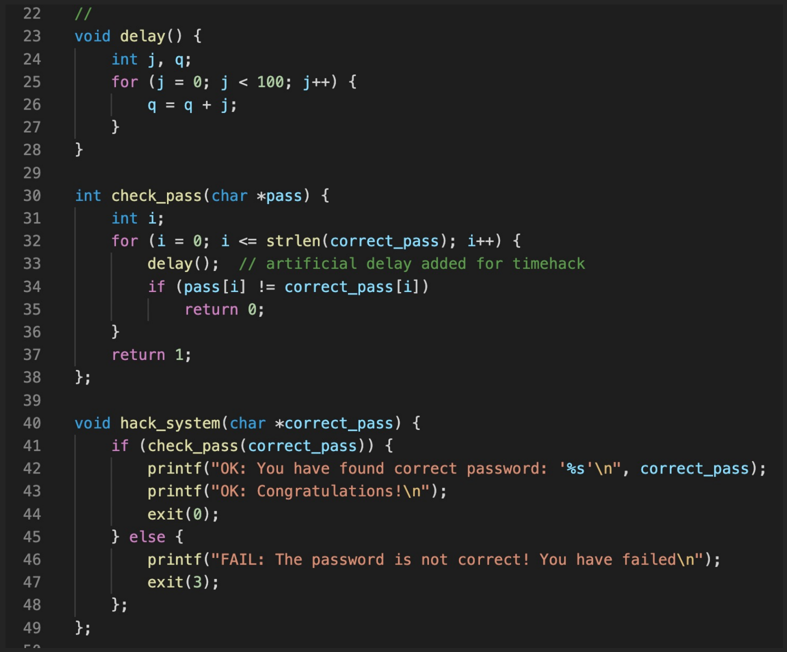
- Two-part assignment on side channels
  - memhack(memory-based side channel attack)
  - timehack(timing-based side channel attack)
- In both of these parts goal is to programmatically guess the password checked in check_pass in sysapp.c
- Rubric:
  - memhack(10pts)
  - timehack(10pts)

Starter code

- Starter code contains files memhack.c, timehack.c, sysapp.c
- Modify memhack.c, timehack.c.
- DO NOT MODIFY sysapp.c

Sysapp.c

- password is passed by reference to check_pass which loops over all characters against true password
- correct_pass is static in starter code but will change while grading, so generalize the solution.
- Delay is added to make time hack more feasible
- Solution should call hack_system when correct password is passed



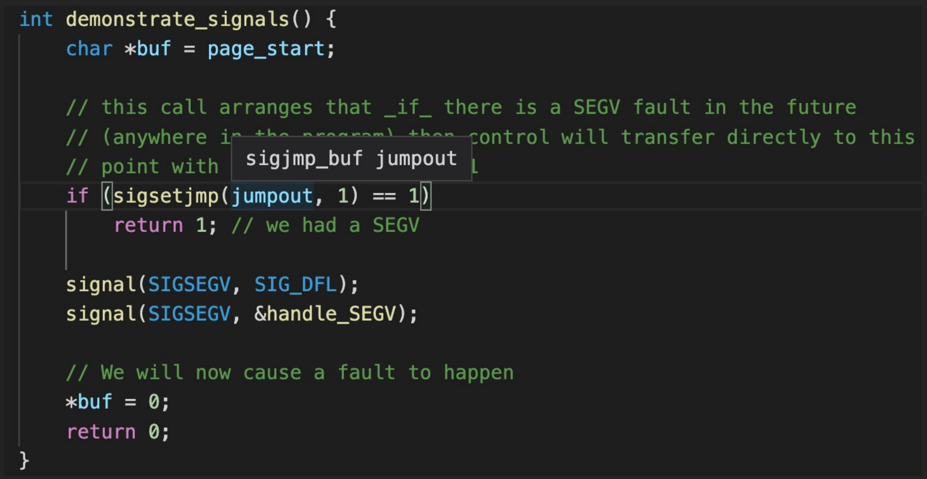
```c
//
void delay() {
    int j, q;
    for (j = 0; j < 100; j++) {
        q = q + j;
    }
}

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 which will cause a seg fault if the program tries to access certain bytes.
- The code on the right demonstrates how you can catch seg faults in the program.



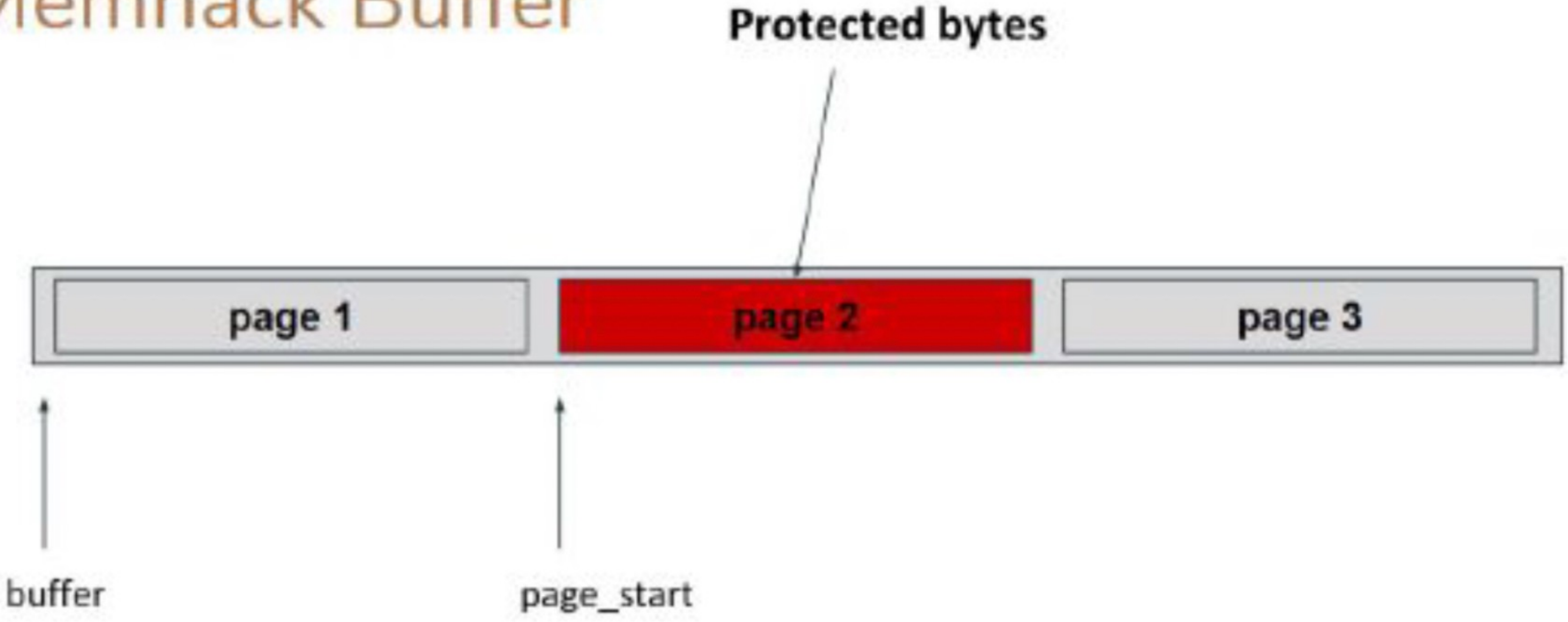
```
int demonstrate_signals() {
    char *buf = page_start;

    // 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  sigjmp_buf jumpout  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;
}
```

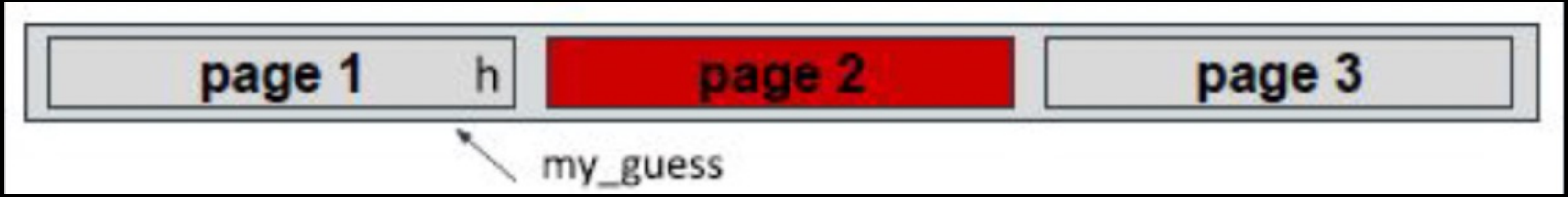
Memhack Buffer



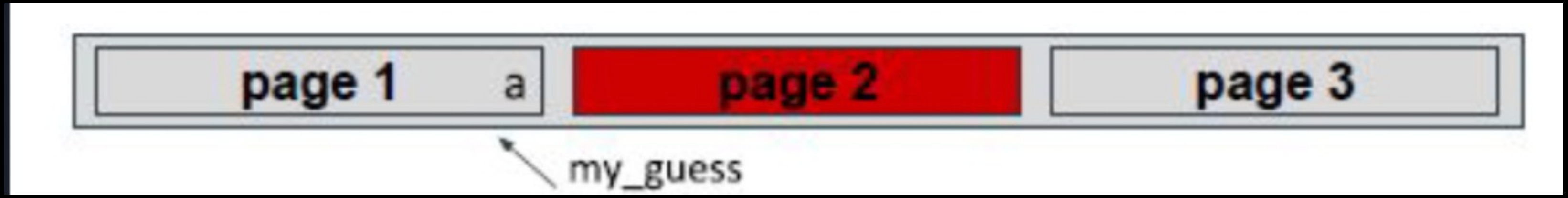
Hints

- You have ability to set access rights to memory and intercept seg faults.
- Password checker takes arg by reference, checks characters sequentially and short circuits on first invalid character
- Referencing protected bytes will cause a seg fault

- For example, if correct password is “hello”



- check_pass(my_guess) causes a fault. Why?



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

Catching Faults

- signal(SIGSEGV, SIG_DFL);
- signal(SIGSEGV, &handle_SEGV);
- This tells the system that whenever it hits a SIGSEGV fault, call the function handle_SEGV.
- SIG_DFL is the default handler, which the documentation requires us to do before being set to handler.
- Use sigsetjmp, siglongjmp to catch faults

timehack.c

- Execution time of check_pass depends on how many characters you have guessed correctly.
- rdtsc returns processor cycle count , use this as a time by calling it before and after check_pass
- There might be lots of noise with each check_pass call, so take multiple samples.

Hints

- Don’t use printf’s in the code, they cause huge variances in exec time.
- Take multiple samples, take the median not the mean as outliers might be extreme. Qsort might be helpful.
- If time is not continuing to increase as you progress through characters , then you probably made an incorrect guess guess earlier.

Good Luck!