Agenda

• PA3 – due Wednesday Nov 9
• 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.
**side channel**: any channel of information produced as a side-effect of conveying information along the primary/intended channel

---

*Figure 2: The Diebold plastic ATM keypad with rubber keys, model 19-019062-001M REV1.*
**side channel:** any channel of information produced as a side-effect of conveying information along the primary/intended channel
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.c(8pts) + writeup(2pts)
  - timehack.c(8pts) + writeup(2pts)
- Writeup:
  - high level idea and explanation of how your code works
Assignment Overview

- Timing based
  - Is the output produced in the same amount of time for each input?

- Memory based
  - Is memory accessed the same way in all cases?
• 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. Test with different correct_pass
- 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",
                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.
Memhack Buffer

Protected bytes

page 1  page 2  page 3

buffer  page_start
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?
Write a helper function similar to this, or modify this starter function.

Cause a segfault inside this function, how will this information help you?

If segfault returns 1, if does not segfault return 0.
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
• 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

if (check_pass(guess)) {
    printf("Password Found!\n");
hack_system(guess);
};

- At an index, try each character
- Calculate how long the check_pass will take
- Do multiple samples to get better estimate of time
- The maximum time-taking one would be the right character for this index
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 earlier.
Good Luck!