CSE 127: Side Channels

- Sourav Anand (Spring 2019)

Side Channels attacks

What are side channel attacks?

Side Channels attacks

What are side channel attacks?

- We see programs which takes inputs and take outputs.
- We saw vulnerabilities which modifies input in order to get secret data.
- Critical information can be revealed in how it is produced.

HW 3

The goal of this assignment is to gain hands-on experience exploiting side channels. This assignment will demonstrate how side channels can be just as dangerous as the control flow vulnerabilities you exploited in Assignment 2

HW3

- Contains 2 side channel exploits.

HW3

- Contains 2 side channel exploits.
- First exploit uses the implementation to generate a signal on correct guess.

HW3

- Contains 2 side channel exploits.
- First exploit uses the implementation to generate a signal on correct guess.
- Second exploit uses the time to match the passwords to generate the password.

HW3

- Contains 2 side channel exploits.
- First exploit uses the implementation to generate a signal on correct guess.
- Second exploit uses the time to match the passwords to generate the password.

HW3 will be released today and will be due by next Friday.

Assignment Infrastructure

- Contains a library sysapp.c which has the password and the function which checks the password (containing the vulnerability).

Assignment Infrastructure

- Contains a library sysapp.c which has the password and the function which checks the password (containing the vulnerability). ***Should not be modified***

Assignment Infrastructure

- *Sysapp.c* is a library which has the password and the functions which checks the password (containing the vulnerability). ***Should not be modified***
- *Memhack.c* is the file where you have to implement the memory based side channel exploit.
- *Timehack.c* is the file where you have to implement the timing based side channel exploit.

Assignment Infrastructure

- *Sysapp.c* is a library which has the password and the functions which checks the password (containing the vulnerability). ***Should not be modified***
- *Memhack.c* is the file where you have to implement the memory based side channel exploit.
- *Timehack.c* is the file where you have to implement the timing based side channel exploit.
- You should use the cfbox VM provided with HW2 to work on this assignment.

Sysapp.c

- Check_pass() takes a char pointer and matches the password character by character.
- Adds a function call to delay() which performs just some random computation to add time delay for checking each character. It is needed to see a significant difference of timing difference.
- hack_system() is should be called at the end to finally verify the guessed password

Memory-Based Side Channel

- a buffer is allocated such that the page starting at page start is protected (i.e., accessing it will cause a segmentation fault, or SEGV) and the previous 32 characters are allocated.
- demonstrate_signals() is an example function which shows how referencing any memory in the protected page will produce a fault as well as how to catch that fault in your program.

Buffer

We allocate a buffer of size 3*PAGE_SIZE. This is because malloc does not gives page aligned buffer

```
//   Page:   1111111111111111222222222222222233333333333333334444444
//                ^ buffer        ^page_start                ^ end of buffer
//   Prot:   ++++++++++++++++----------++++++++++++++++++++++++
```

Password type

- Max 32 characters
- ASCII symbols from ASCII 33 (“!”) to ASCII 126 (“~”) can be used in the password.

Exploit hint

- Referencing protected memory will raise the fault.
- Capture the fault and use this as a hint of correct guess.
- For the first guess you store the password such that its first character is one byte before page start and then iterate between possible choices for the first character .

Timing based side channel

- The execution time of check_pass() depends on how many characters you guess correctly.

Timing based side channel

- The execution time of check_pass() depends on how many characters you guess correctly.
- rdtsc() returns processor cycle count

Timing based side channel

- The execution time of check_pass() depends on how many characters you guess correctly.
- rdtsc() returns processor cycle count
- Insert a call to rdtsc() before the call to check pass() and afterwards and get the difference.

Timing based side channel

- The execution time of check_pass() depends on how many characters you guess correctly.
- rdtsc() returns processor cycle count
- Insert a call to rdtsc() before the call to check pass() and afterwards and get the difference.
- The time taken to run check_pass() can also contain noise (Why?)

Hints for implementation

- Be careful in using printf’s as it can affect the time you are trying to capture.

Hints for implementation

- Be careful in using printf’s as it can affect the time you are trying to capture
- IDEA: store all the info you want to print for debugging and print it at the end

Hints for implementation

- Be careful in using printf’s as it can affect the time you are trying to capture.
- IDEA: store all the info you want to print for debugging and print it at the end.

- There can be noise in the timing data collected.

Hints for implementation

- Be careful in using printf’s as it can affect the time you are trying to capture.
- IDEA: store all the info you want to print for debugging and print it at the end.

- There can be noise in the timing data collected.
- IDEA: Run multiple trials and gather the data.

Hints for implementation

- Be careful in using printf's as it can affect the time you are trying to capture.
- IDEA: store all the info you want to print for debugging and print it at the end.

- There can be noise in the timing data collected.
- IDEA: Run multiple trials and gather the data. **How should you get the timing form the data?**

Hints for implementation

- Be careful in using printf’s as it can affect the time you are trying to capture.
- IDEA: store all the info you want to print for debugging and print it at the end.

- There can be noise in the timing data collected.
- IDEA: Run multiple trials and gather the data. Use Median (and not average).

Hints for implementation

- Be careful in using printf’s as it can affect the time you are trying to capture.
- IDEA: store all the info you want to print for debugging and print it at the end.

- There can be noise in the timing data collected.
- IDEA: Run multiple trials and gather the data. Use Median (and not average).

- If time is not continuing to increase as you progress through characters, then you probably made a bad guess earlier. Backtrack

Questions?