Module 07: Trusted Execution Environments

Week 14: Extracting Secrets using side-channels

Shweta Shinde

shweta.shivajishinde@inf.ethz.ch

Hi all! Welcome to the last Information Security Lab for this semester. This lab is part of the Module-07 on *Trusted Execution Environments*, or TEEs in short.

Instructions for Online Lab

We begin with a few pointers and instructions. As you are aware, the lab sessions will be held online via Zoom. We ask that all students join the Zoom session via the following link:

`https://ethz.zoom.us/j/91450896871`

The first 30-ish minutes of the Zoom session will consist of a general presentation and discussion about the lab objectives and evaluation criteria. Afterwards, you may choose to stay on in the lab session to discuss specific problems you might be having with the lab. You are also welcome to post your questions and to join ongoing discussions on the Moodle forum at the following link:

`https://moodle-app2.let.ethz.ch/course/view.php?id=13172`

During the Zoom session, we will be hosting “break-out rooms". These are meant to facilitate one-on-one discussions with one of the TAs. During such discussions, you can share your screen and bring to our attention specific implementation issues that you might be facing. Please be aware that when you are sharing a screen with the session, you are responsible for the content that is presented to any other students in the breakout. In other words, *please be mature*.

Overview

This lab is focused on studying the leakage of information via side-channels. The goal is to study and analyze the sources of leakage and leverage them to build an analysis tool that extracts program secrets.

Your primary task in this assignment is to think like an attacker. Here is the scenario: the victim program checks if an input matches a secret password. You cannot directly access the password. The program does not have any logical or memory vulnerabilities. So the only way you can try to crack the password is by looking at side-channels. You have to implement such password crackers.

Please note that we expect you to use Python-3 for this lab. We will not accept submissions/solutions coded in any other language, including older versions of Python. *So please make sure that your submissions are Python-3 compatible.*

Getting Started

You are provided with a VM, it has several tools and infrastructure required for the task. Please boot the VM and get familiar with the following layout and parts:

```
isl
├── t1
│   ├── {password_checker_1
│   ├── password_checker_1.c
│   ├── password.txt
│   ├── run.sh
│   └── samples
└── t2
    ├── password_checker_2
    ├── password_checker_2.c
    ├── passwords
    ├── password.txt
    └── run.sh
```

Victim Program. We provide two victim programs and corresponding binaries.

`password_checker_1.c` & `password_checker_2.c`: These two source file located in `/home/sgx/isl/t1` and `/home/sgx/isl/t2` folder respectively. These are two C programs that compare a password guess (input string) with the secret password. They outputs whether the guess was correct or wrong. Please read the program source code carefully to understand some of the assumptions about the inputs and the secrets. Assume that these programs are free of any memory vulnerabilities (e.g., buffer overflows).

`run.sh`: Script to statically compile the password checker programs to produce corresponding executable binaries. Note that the programs are statically linked to a math library (`-lm`) and are compiled with optimizations turned off (`-O0`) and debug information enabled (`-g`).

`password_checker_1`, `password_checker_2`: Corresponding executable binary produced using `run.sh`.

Secrets. The programs operate on a secret password.

`password.txt`: Both `password_checker_1` and `password_checker_2` loads the secret password from this file. Thus, the secret is not embedded in the source code or the binary.

Attacker Tools. As an attacker, a few tools remain available at your disposal. For an assignment, it is challenging to setup real Intel SGX machines and ask students to extract side-channel information about enclaves. Instead, we have simplified it a little bit. We give you all the information that the attacker can observe using several sources of side-channels. Your task is to simply analyze this information and extract the secret. Below we describe how we collect this information.

`Pin`: A dynamic binary instrumentation framework provided by Intel. The Pin framework provides a developer friendly way to write several dynamic program analysis tools. We will be using the Pin framework to mimic a real attacker that wants to extract the information about our victim program. The framework is already built and installed in the VM, you will not be required to change anything in the framework. The tool is located at `/home/sgx/pin-2.14-71313-gcc.4.4.7-linux/`, the `pin.sh` script runs the pin command.

`SGX_Trace`: This is a custom tool that we wrote on top of Pin framework. When a program is executed with this tool, it records important information about the execution. Specifically, it records the address and opcode of each instruction that is executed in the program. Further, it also records the data addresses that were accessed (read/write) for each instruction. The tool is located in the `/home/sgx/pin-2.14-71313-gcc.4.4.7-linux/source/tools/SGXTrace` folder, the `SGXTrace.cpp` file contains the tool source code. To run the SGXTrace tool, first enter the `/home/sgx/pin-2.14-71313-gcc.4.4.7-linux/source/tools/SGXTrace` directory. Then run the following command:

```
../../../pin.sh -t ./obj-intel64/SGXTrace.so -o <output-file>
-trace 1 -- <victim-program>
```

For example, you can trace the `ls /home` command as follows:

```
sgx@sgx-VirtualBox:
~/pin-2.14-71313-gcc.4.4.7-linux/source/tools/SGXTrace$
../../../pin.sh -t ./obj-intel64/SGXTrace.so -o ~/isl/ls.txt
-trace 1 -- ls /home
```

You will see the following output:

```
================================================
This application is instrumented by SGXTrace Pintool
PID: 7028
See file /home/sgx/isl/ls.txt for analysis results
================================================
sgx
```

`trace_samples`: We tested the victim program on a bunch of secret passwords. For each password, we made several guesses. We recorded all these executions using our SGX_Trace tool. So, we were able to simulate an attacker

who doesn't know the password, but can make several guesses. For each guess, the attacker can observe an *execution trace* as collected by the SGX_Trace tool and the final output (either `correct` or `wrong`) printed on the console. We have provided several such trace samples for different passwords. For each password (e.g., samples/1/password.txt), there are several traces over different inputs in the `traces` folder (e.g., sample/1/traces/magicbeans.txt, sample/1/traces/magicbeant.txt). The password.txt file contains the secret password. Each trace file is named as the input what was used to generate the trace. For example, the trace for the command `./password_checker_1 magicbeans` is in file sample/1/traces/magicbeans.txt.

`Trace Format`: The trace file has a lot of information which may seem overwhelming in the beginning. Here is an example that explains the trace format:

```
E:0x7f71b3133102:C:7:add qword ptr [rax+0x8], rcx
R:0x7f71b3350e60:C
W:0x7f71b3fdea60:D
```

The first line is (E), i.e. this line is about the current EIP, followed by (0x7f71b3133102) the instruction address in the virtual memory, followed by C, which means this is from the code section, and then (7) the opcode, followed by the disassembly of the instruction (add qword ptr [rax+0x8], rcx). Line 2 tells us that the instruction caused a read (R) at the address (0x7f71b3350e60) in the code section (C). Line 3 tells us that the same instruction also caused a write (W) at the address (0x7f71b3fdea60) in the data section (D). Please take a look at the binary assembly, input values, and corresponding trace entries to understand the details.

Goal

Given the same secrets, the victim program exhibits different execution behavior for different inputs. For instance, consider that the secret (password) is `abracadabra`. When we execute a password checker program with an input password guess of `guesspassword`, the program output will tell us that the input is incorrect. Thus, the output tells a little bit about the secret. In this case, it tells us that the secret password is not `guesspassword`. However, the attacker can try several inputs and observe other aspects of the execution. By analyzing how the program behavior changes with the change in the input, the attacker can extract more information about the secret.

In the case of the TEE threat model, the attacker can be the OS, the hypervisor, non-enclave programs, or malicious enclaves co-resident on the same SGX machine. As we will see in the class, this new threat model exposes several new avenues of information leakage, mainly via side-channels. In this lab, you are presented with a corpus of such information that can be collected via side-channels in SGX. However, such low-level knowledge about a programs execution may or may not lead to a complete leakage of program secrets. The attacker has to find clever

ways to make sense of the information collected via side-channel(s).

In this lab, we will learn how to analyze the information collected via side-channels, compare it across several executions over varying inputs, and then devise attack strategies. In the process, we will study the reduction in the attacker's uncertainty about the program secret. Finally, we will put on the defender's hat and propose mechanisms to thwart the information leakage by protecting against the side-channels.

Dos and Don'ts

Password Checker. Do not move or change or modify the password checker source code or binary. Similarly, please do not change the provided sample execution traces and passwords. Any change will affect the program addresses and will not match the pre-recorded samples. You can examine the source code and the binary (say in read-only mode). Please be mindful of this especially when you test your solution on the traces or generate your own traces.

Pin & SGX_Trace Tool. We have provided these tools for your reference, in case you want to understand how the programs are being traced and what does each line in the trace mean. Please do not change these tools when you are coding or testing your solution. You can use the above tools to trace some existing programs (e.g., `ls`) or test programs that you code to understand the details of the trace collection. That way, you can generate your own traces and get some insights into the sample traces that we have provided.

Trace & Trace Samples. To assess your solution for Task T1, we will first test it on these sample traces. Then we will test it on extensive sample traces (generated using other passwords). So make sure your solution computes the correct password at least on the sample traces. If you want to do more extensive testing, you can generate your own sample traces on passwords of your choice.

Coding Tasks

[50 points] T1: Write a password cracking program that extracts the secret password by analyzing the execution traces of `password_checker_1` binary provided to you. Please code in Python 3 as a single script file. Name your code script as `module_7_password_cracker_t1.py`.

- 20 points if the password cracker works on the 5 sample traces provided with the VM, 4 points per trace.
- 30 points if the password cracker works on our reserved traces (not provided with the VM), 6 points per trace.

- You should only use the sample traces provided to you. Your solution for this task is not allowed to execute the `password_checker_1` binary or the tracer.
- Your script is not allowed to directly access the password file or read the password from the program memory. On our evaluation system we will ensure that the script can only access the trace files. For each set of sample traces, if your script fails to print the password, takes too long (more than 1 minutes) to finish, or has errors in its execution, we will run it again for a total of 3 times. If in these 3 attempts it does not print the password, you will not get the points. Thus, it is important for your script to be reliable: remember the cracker should run outside GDB or any other tool that you may use during testing.
- The exact commands that we will run is: `python3 module_7_password_cracker_t1.py <trace_folder>`. This command should print out the password for that folder of traces and a status flag (either `complete` or `partial`. These two strings must be separated by a space. For example: python3 module_7_password_cracker_t1.py sample/1/traces should print `magicbeans complete` if you are sure that the complete password is `magicbeans`. If you think that the given traces do not have sufficient information to recover the entire password, you can print the partial password that you are able to recover. For example, if you recover only partial password, say `magic`, then you should print `magic partial` The first output of the command `python3 module_7_password_cracker_t1.py sample/1/traces` should match the content in the file `sample/1/password.txt`.
- For a set of traces, you will get full points only if your script successfully prints out the correct password and status flag.

  Any attempt to tamper with the test passwords or trace files will be heavily penalized.

[25 points] T2: Write a password cracking program that extracts the secret password for `password_checker_2` binary by generating your own execution traces and then analyzing them. Please code in Python 3 as a single script file. Name your code script as `module_7_password_cracker_t2.py`.

- 10 points if the password cracker works on the sample passwords provided with the VM, 2 point per password.
- 15 points if the password cracker works on our reserved passwords (not provided with the VM), 3 points per password.
- You are allowed to execute the `password_checker_2` binary and the tracer program in your solution to generate your own traces for your solution (this is not allowed for Task T1).
- Rest of the rules and output format are same as for Task T1.
- For each set of sample traces, if your script fails to print the password, takes too long (more than 2 minutes) to finish, or has errors in its execution, we will run it again for a total of 3 times. If in these 3 attempts it does not print the

password, you will not get the points. Thus, it is important for your script to be reliable: remember the cracker should run outside GDB or any other tool that you may use during testing.

Questions

Please setup the VM, read through the `password_checker_1.c` and `password_checker_2.c` programs, and look at the sample traces before proceeding to the following questions. Please submit your answers in a PDF file produced using the LaTeX template `module_7_solution.tex` provided to you. Follow the instructions in the LaTeX template and format your solutions accordingly.

[5 points] Q1: If the attacker has access to no side-channel (i.e., it does not have any knowledge about the program execution), at most how many attempts will be required to guess the password correctly? You can assume that the attacker has access to the program source code, and hence is aware of the assumptions about the password. Please state the number of attempts required and your reasoning. (3 points for number of attempts, 2 points for reasoning)

[5 points] Q2: For Task T1, if you as an attacker were allowed to select the input i.e., guess the password:

(a) for all possible passwords accepted by the program, can you extract the secret password with a single guess? (2 points)

(b) If so, what is the exact attack strategy to craft the guess and analyze the trace? (3 points)

[5 points] Q3: For Task T2, for all possible passwords accepted by the program, at most how many attempts will be required to guess the password correctly? You can assume that the attacker has access to the program source code, and hence is aware of the assumptions about the password. Please state the number of attempts required and your reasoning. (2 points for number of attempts, 3 points for reasoning)

[4 points] Q4: Imagine that the `password_checker_1` program was being executed inside Intel SGX enclave. A real-world attacker cannot directly access the password file. To crack the password, the attacker wants to extract some information, similar to what we provided to you in the sample execution traces. State what exact information should the attacker extract and the corresponding side channels that they can use. Please limit to maximum two types of information, one sources for each. (2 point each for stating type of information, 2 point each for stating the corresponding side-channel)

[6 points] Q5: Pick one type of information + side-channel you stated in Q4, describe 2 concrete defenses to prevent that leakage. (3 points per defense scheme)

Solution Template

To submit your answers to Questions Q1-Q5 listed above, use the solution LaTeX template (`module_7_TEEs_solution.tex`) provided on Moodle. We have also provided a sample PDF (`module_7_TEEs_solution.pdf`) that will be produced using the template. Please make sure that you add your Student ID to the template, fill in your answers in the given boxes, and submit a PDF file with the right name (see submission format instructions below for more details).

Discussions

Shweta Shinde (shweta.shivajishinde@inf.ethz.ch), Gianluca Lain (gilain@student.ethz.ch), and Samuele Piazzetta (spiazzetta@student.ethz.ch) will monitor the Moodle discussions. We encourage you to ask clarification questions on this public forum. That way everyone will benefit from clarifications we provide and it reduces duplicate questions.

If you have a question that requires explaining your solution, please send a private email to the three of us. Note that we will only answer questions that genuinely warrant a response. As a rule of thumb, we will not provide hints or help you to debug your code.

Submission Format

Your completed submission for this module should consist of a single archive file named `module_7_submission_[insert LegiNo].zip`, containing *two* Python files named: `module_7_password_cracker_t1.py` and `module_7_password_cracker_t2.py`; and a single PDF file named `module_7_solution.pdf`. Submit a zip file containing the above three files in a flat structure: your zip archive should only contain the PDF file and the python files. You are expected to upload your submission to Moodle.

In conclusion, *happy cracking & coding*!