Assignment 7, Fall 2018
CS4630, Defense Against the Dark Arts
Spectre Variants 1 and 2

Purpose

In this assignment you will perform experiments with Spectre variant 1 (data cache information leak) and spectre variant 2 (branch target buffer poisoning).

You will be leaking a secret string from our Spectre variant 1 proof of concept binary and you will get hands on experience implementing (in assembly code) the software-based *retpoline* mitigation for Spectre variant 2 published by Intel in a Spectre variant 2-vulnerable program.

Due

This assignment is due on **Tuesday, 06-NOV-2018 at 11:59 pm**

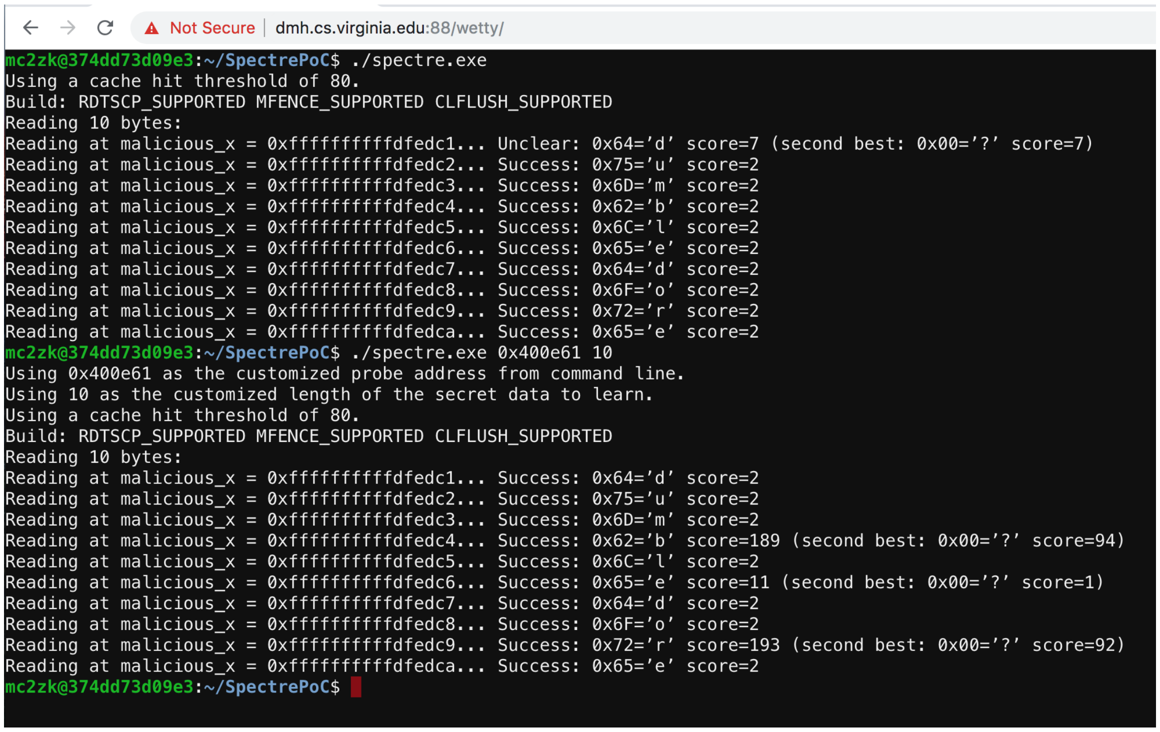
1 Spectre variant 1

NOTE: Work on solving this part of the assignment on your CS 4630 VM

In Spectre variant 1, an attacker mistrains the branch predictor to mispredict the direction of a branch. The atttacker takes advantage of speculative execution to cause secret information to be read.

For this part of the assignment, we have provided a Spectre variant 1 proof of concept binary, `spectre.exe` which leaks secret data, `username`.

Use the reverse engineering tools that you've learned about in the class such as `gdb, ghex,` (or other hexadecimal editors) and `objdump` to gather the information you need to cause the program to leak another secret value, `password` contained within the binary.



```
mc2zk@374dd73d09e3:~/SpectrePoC$ ./spectre.exe
Using a cache hit threshold of 80.
Build: RDTSCP_SUPPORTED MFENCE_SUPPORTED CLFLUSH_SUPPORTED
Reading 10 bytes:
Reading at malicious_x = 0xffffffffffdfedc1... Unclear: 0x64='d' score=7 (second best: 0x00='?' score=7)
Reading at malicious_x = 0xffffffffffdfedc2... Success: 0x75='u' score=2
Reading at malicious_x = 0xffffffffffdfedc3... Success: 0x6D='m' score=2
Reading at malicious_x = 0xffffffffffdfedc4... Success: 0x62='b' score=2
Reading at malicious_x = 0xffffffffffdfedc5... Success: 0x6C='l' score=2
Reading at malicious_x = 0xffffffffffdfedc6... Success: 0x65='e' score=2
Reading at malicious_x = 0xffffffffffdfedc7... Success: 0x64='d' score=2
Reading at malicious_x = 0xffffffffffdfedc8... Success: 0x6F='o' score=2
Reading at malicious_x = 0xffffffffffdfedc9... Success: 0x72='r' score=2
Reading at malicious_x = 0xffffffffffdfedca... Success: 0x65='e' score=2
mc2zk@374dd73d09e3:~/SpectrePoC$ ./spectre.exe 0x400e61 10
Using 0x400e61 as the customized probe address from command line.
Using 10 as the customized length of the secret data to learn.
Using a cache hit threshold of 80.
Build: RDTSCP_SUPPORTED MFENCE_SUPPORTED CLFLUSH_SUPPORTED
Reading 10 bytes:
Reading at malicious_x = 0xffffffffffdfedc1... Success: 0x64='d' score=2
Reading at malicious_x = 0xffffffffffdfedc2... Success: 0x75='u' score=2
Reading at malicious_x = 0xffffffffffdfedc3... Success: 0x6D='m' score=2
Reading at malicious_x = 0xffffffffffdfedc4... Success: 0x62='b' score=189 (second best: 0x00='?' score=94)
Reading at malicious_x = 0xffffffffffdfedc5... Success: 0x6C='l' score=2
Reading at malicious_x = 0xffffffffffdfedc6... Success: 0x65='e' score=11 (second best: 0x00='?' score=1)
Reading at malicious_x = 0xffffffffffdfedc7... Success: 0x64='d' score=2
Reading at malicious_x = 0xffffffffffdfedc8... Success: 0x6F='o' score=2
Reading at malicious_x = 0xffffffffffdfedc9... Success: 0x72='r' score=193 (second best: 0x00='?' score=92)
Reading at malicious_x = 0xffffffffffdfedca... Success: 0x65='e' score=2
mc2zk@374dd73d09e3:~/SpectrePoC$
```

Figure 1: Running `spectre.exe`. Note that you can pass an address and length on the commandline

Submit the following to Collab:

1. **Screenshot of your working information leak**. The screenshot should include your UVA computing ID somewhere within the image.

2. In a file named **`spectre-results-analysis.txt`**, provide a brief writeup describing how you gathered the information you needed to leak the information, including the tools used, information sought, how you used that information to launch a successful information leak.

2 Spectre variant 2

Spectre variant 2 (branch target injection) exploit targets the branch target buffer (BTB). Indirect branches such as indirect `jmp` or indirect `call` instructions occur when the destination of the branch is not contained in the instruction itself, but rather in a register or a memory location. The BTB is a special-purpose caching mechanism which stores branch target addresses and uses previously-executed branches (*branch history*) to predict future branch target (destination) addresses.

In Spectre variant 2, the branch predictor is mistrained such that speculative execution continues at a location that should not be reached during legitimate program execution. This is called *indirect branch poisoning* or *branch target injection*.

Several mitigations have been proposed for preventing indirect branch poisoning. Intel and AMD extended the ISA to contain three control mechanisms (see https://spectreattack.

com/spectre.pdf). However, these controls require a microcode patch and OS or BIOS support for use.

In this assignment, you will be working with an alternative prevention mechanism proposed by Google called *retpolines*. A *retpoline* is a code sequence that replaces indirect branches with return instructions. This code sequence also makes sure that the return instruction is predicted to a benign endless loop through the return stack buffer. The actual branch target destination is reached by pushing it on the stack and returning via the `ret` instruction.

https://software.intel.com/security-software-guidance/api-app/sites/default/files/Retpoline-A-Branch-Target-Injection-Mitigation.pdf describes the Spectre variant 2 exploit, explains the principle behind *retpolines*, and explains the *retpoline* implementation. This document is also provided to you on Collab for reference.

2.1 Spectre Variant 2 Assignment Details

NOTE: Due to VM timing issues, for this portion of the assignment you must use `http://dmh.cs.virginia.edu:88/wetty`. Please keep in mind the UVA Honor Code and resist the temptation to look at other students' solutions.

We have given you on Collab (and on the DMH machine) two files: a `Makefile` and a vulnerable program, `assignment-unsafe.s`.

For this part of the assignment, you will do the following:

1. Identify the Spectre variant 2 vulnerable code in `assignment-unsafe.s`
2. Create a new assembly code program, `assignment-safe.s` in which you replace the vulnerable code with a software *retpoline* as described in the Intel document.
3. Collect and write up an analysis timing data between `assignment-unsafe` and `assignment-safe`. For the purposes of this assignment, collect timing data from 5 runs of each program. Because timing is important, you will be writing, building, and running your experiments on a CS department machine (http://dmh.cs.virginia.edu:88/wetty) instead of the class reference VM. The web-based tty interface gives you a commandline terminal window. We have installed three text editors: `vim`, `nano`, and `emacs`. You may use any of these text editors to edit your files.

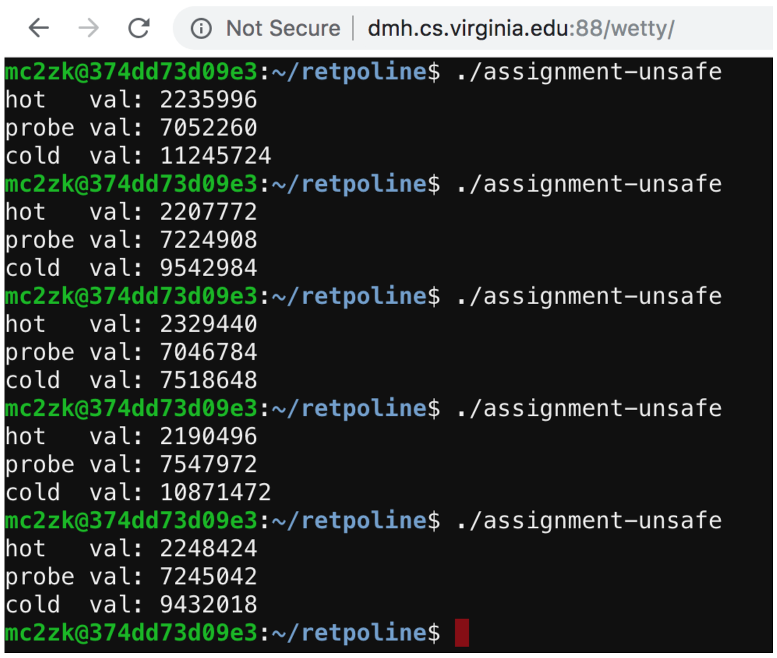
2.1.1 Understanding `assignment-unsafe`

We have placed a copy of the `Makefile` and `assignment-unsafe.s` in each of your home directories on http://dmh.cs.virginia.edu:88/wetty in the `asst7` folder.

Your login credentials are your UVA computing ID for both username and password.

The makefile contains the build instructions. To build `assignment-unsafe`, simply type `make` You can edit this makefile later to include a rule to build your *retpoline*-protected `assignment-safe.s` to save repeated typing.

When you run `assignment-unsafe`, you see output similar to that shown in Figure **??**. Run `assignment-unsafe` 5 times and record the results in your `spectre-results-analysis.txt` file for later analysis.



```
mc2zk@374dd73d09e3:~/retpoline$ ./assignment-unsafe
hot   val: 2235996
probe val: 7052260
cold  val: 11245724
mc2zk@374dd73d09e3:~/retpoline$ ./assignment-unsafe
hot   val: 2207772
probe val: 7224908
cold  val: 9542984
mc2zk@374dd73d09e3:~/retpoline$ ./assignment-unsafe
hot   val: 2329440
probe val: 7046784
cold  val: 7518648
mc2zk@374dd73d09e3:~/retpoline$ ./assignment-unsafe
hot   val: 2190496
probe val: 7547972
cold  val: 10871472
mc2zk@374dd73d09e3:~/retpoline$ ./assignment-unsafe
hot   val: 2248424
probe val: 7245042
cold  val: 9432018
mc2zk@374dd73d09e3:~/retpoline$
```

Figure 2: Running `assignment-unsafe`. Timing numbers are reported for `hot` (in cache), `probe` (should not be in cache, but might be), and `cold` (should not be in cache). Note that the timing numbers vary from run to run.

2.1.2 Implementing *retpolines*

Read through the Intel *Retpoline* White Paper document to see examples of the assembly code implementation of *retpolines*. Make a copy of `assignment-unsafe.s` named `assignment-safe.s`, then modify `assignment-safe.s` to replace the indirect branch code with a *retpoline*. Build and run it 5 times, again collecting the timing information in the file `spectre-results-analysis.txt` for analysis.

2.1.3 Analyzing results

Now that you have implemented *retpolines* and gathered timing information for both `assignment-unsafe` and `assignment-safe`, it's time to examine the results you've gathered.

Here are some comparative results gathered from other real hardware:

Hardware	Effect	probe val unprotected fetch time	probe val protected fetch time	Slowdown factor	Memory speed
Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz	pronounced	approx. 3652852	approx. 12131068	about 3.3x factor	34.1 GB/s
Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz	effects are there, but not as visible	approx. 8549019	approx. 13958326	about a 1.63 factor	25.6 GB/s
Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz	pronounced	approx. 4332110	approx. 19979450	about a 4.6x factor	59.7 GB/s
Intel(R) Xeon(R) CPU E5645 @ 2.40GHz	pronounced	approx. 3071262	approx. 10089712	about a 3.28x factor	32 GB/s
Intel(R) Xeon(R) CPU X7550 @ 2.00GHz	effects are there, but not as visible	approx. 10316529	approx. 15005052	about a 1.45x factor	unknown

Items to Submit - Summary

There are several items to submit to the course Collab site. All files should include your name and UVA computing ID (if in code, as a comment). This assignment is due on **Tuesday, 06-NOV-2018 at 11:59 pm** .

1. Spectre variant 1 screenshot of your successful information leakage of `password` using `spectre.exe`'s commandline parameters.
2. Your `assignment-safe.s` file which implements the retpolines. It must be called `assignment-safe`
   This file should contain your implementation of the software retpoline described in the Intel document.
3. `spectre-results-analysis.txt`, your results and analysis file. This file should contain the following:
   (a) Spectre variant 1: Your Spectre variant 1 description of your how you performed the information leakage.
   (b) Spectre variant 2: your timing results for 5 timing runs each of the unprotected executable and of the executable you produce from your retpoline implementation.
   (c) Spectre variant 2: An analysis of the average relative timing difference between reported unsafe vs. safe probe_val fetch times.
   (d) An explanation of your understanding of the results you got.

It is mandatory that you use the file names given to ease the task of grading many different submissions of this assignment. Throughout the semester, you will be given file names. All assignments will be submitted using the class Collab Website.