CSE 484 / CSE M 584 - Homework 3

This homework is focused on a variety of topics from the last ~third of the quarter, with the goal of giving you some more hands-on experience with various tools.

Overview

- **Due Date:** Friday, June 1, 2018 (4:30pm).
- **Group or Individual:** Individual
- **How to Submit:** Submit to Canvas
- **Total Points:** 45 points across 4 parts

Part 1: Web Tracking (10 points)

Experiment with an anti-tracking browser add-on, such as Ghostery, Lightbeam, or Privacy Badger. Pick three websites (e.g., www.cnn.com, www.facebook.com, and www.weather.com -- though you may pick any sites), visit them with the add-on installed, and report on what you find.

What to Submit:

1. **(3 points):** Briefly describe (a few sentences) or sketch how third-party tracking allows advertisers or others to track users across multiple sites.
2. **(1 point):** Which add-on did you try?
3. **(6 points):** Include a screenshot of the add-on's output for each of the 3 pages you tested. How many trackers did you find on each page?

Part 2: Android Encryption (15 points)

In this part of the assignment, you will explore a vulnerability in how Android applications might use encryption. **Your goal:** Extract the encryption key from an Android application (without access to source code).

Download SimpleNotepad application here: SimpleNotepad.apk. This application lets users write notes, store them, and retrieve them. Because the developer knew that users might write private things in their notes, he/she decided to *encrypt* those notes before storing them on the device.

What to Submit:

1. **(5 points):** Find the encryption key used by SimpleNotepad, and briefly describe (one short paragraph) how and where you found it.

   Hint: You don't have any source code! :(But check out APKTool, which lets you decompile Android applications.
3. **(5 points):** Briefly describe (one paragraph) why it's a problem that SimpleNotepad's encryption key is hard-coded into the app, and explain what the developer of SimpleNotepad should have done instead.

4. **(5 points):** Identify at least one other way in which the developer of SimpleNotepad is not using best practices for encryption.

Part 3: Password Security (10 points)

Below we give you the entry for a password stored on a Linux machine. The password is weak. *Your goal:* find the password.

To do this, we recommend using either john or hashcat. We strongly recommend using Linux for this question (Use attu or a VM if you don't have a native Linux). You can likely install John the Ripper from the repository using apt-get or yum. The Linux package name is most likely john, e.g., for Ubuntu, run "sudo apt-get install john". Otherwise, you can download john and build it from source (you will have to use this option if you are using attu, during build specify the architecture as linux-x86-64): John the Ripper 1.8.0 (sources, tar.xz, 4.3 MB) and its signature

Here is the password entry, from a Linux machine:

```
charizard:$6$qQYokRkW$r7BpKNfSjOdGM6IIL/jeOWYJdQi10Uwf713Gxu6C.n7Qmusau
XG1fW73lHf9FbdGcm25fdLDTL3Xo4eR9ozsK.:17490:0:99999:7:::
```

What to Submit:

1. **(8 points)** The password
2. **(2 points)** Approximately how long it took the tool you used to crack the password

Part 4: Spectre (10 points)

Your task is to exploit the Spectre vulnerability on a target, vulnerable program. We provide you with a VMWare Virtual Machine instance with everything that you need to get started.

The default username for this VM is "spectre", and the password is "484exploitme". In the home directory, there's a folder called "spectre" that has all the materials:

- A README explaining all of the files
- The compiled victim code, in the file spectre-victim.o. There is a secret embedded in this object code. Read below for more rules on how to extract the secret from this file.
- A header file (spectre-victim.h) for the externally-facing parts of the compiled victim code
- The uncompiled c exploit code, in the file spectre-exploit.c
- A python file for running the exploit and generating pretty graphs
- A Makefile
- An executable named 'spectre-control' that can be run to verify that the system supports the Spectre exploit

The first thing you should do is run spectre-control. If spectre-control outputs the secret “The Magic Words are Squeamish Ossifrage.”, one character per line, then that means that your system, running this VM, is vulnerable to the Spectre attack and that you will be able to solve this homework assignment.

The next thing you should do is run “make” and then “python3 spectreAnalyze.py”. Note that you may need to explicitly specify the use of python3. You should see a bunch of graphs pop up, as well as output on the command line. The graphs reflect the attack program’s attempt at extracting the secret information embedded in spectre-victim.o. However, the attack is not yet successful -- you need to figure out how to make the attack successful. Once you have made the attack successful, the graphs will clearly show the cache read timing differences for each byte of the secret information.

Your next step is to familiarize yourself with spectre-exploit.c. This code will be compiled with spectre-victim.o. This code will call the victim function, aptly named victim_fuction(), in spectre-victim.o multiple times. INV_ATTACK_FREQ-1 out of INV_ATTACK_FREQ times, it will call the victim function in a way that trains the system’s branch predictor to follow a certain code path. And 1 out of INV_ATTACK_FREQ times it will be run with input that should not follow that certain code path … but it will follow that code path speculatively, if the branch predictor choses to do so, and then not commit the changes once it realizes that the prediction was incorrect.

This is the entirety of the code for spectre-victim.c:

```
/*
VICTIM CODE
*/

#include "spectre-victim.h"

unsigned int array1_size = 16;

uint8_t unused1[64];
uint8_t array1[120] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};
uint8_t unused2[64];
uint8_t array2[256 * 512];

char *secret = "This is some secret information!";

uint8_t temp = 0; /* Used so compiler won't optimize out victim_function() */

void victim_function(size_t x)
{
   if (x < array1_size)
```

```
    {
        temp &= array2[array1[x] * 512];
    }
}

int main()
{
    return 0;
}
```

You can compile it into spectre-victim.o and experiment with it directly, via “gcc spectre-victim.c -g -c -fPIC” and experiment with it directly, if you wish, but *make sure that your actual homework submission uses the spectre-victim.o file that we provide*.

Back to spectre-exploit.c, this code should work … except that there is some problem in the code following the “What might be problematic here?” comment. Your task: to figure out what that problem is, and fix the problem. The logic in that snippet of code is correct. But there’s something about the interplay between the compiled version of that code, the underlying architecture, and the Spectre vulnerability that is making the attack ineffective. Replace that snipped of code with code of your own design that implements the same logic. While INV_ATTACK_FREQ is a #define, you may just fix it to be whatever magic number you wish.

You may also adjust the CACHE_HIT_THRESHOLD and NUM_TRIALS #defines.

What to submit:

- Your spectre-exploit.c exploit code.
- The 7 byte secret that you needed to recover.
- A paragraph describing why the provided version of spectre-exploit.c did not work.
- A second paragraph describing why your modified spectre-exploit.c code does work
- The graphs output by your working exploit (screenshots are fine).
- Information about the machine that you did the test (e.g. provided VM running inside VMWare Player version 14, on a Windows 10 Enterprise with an Intel i7-2600 CPU at 3.4GHz and with 16GB RAM).

Note: Here is a git repository where you can download everything that you need in order to do this attack on your own Linux machine, assuming that the compiled spectre-victim.o file is of the appropriate type for your machine.
https://gitlab.cs.washington.edu/sbenaloh/Spectre-Final
However, since we cannot fully anticipate what might go wrong in such a configuration, we cannot help debug issues that might arise if you choose to do this. So we do encourage you to use the VM that we provide. However, if you’re interested in how this attack works on your native machine, please feel free to experiment!

What is not allowed:

- There are many ways to obtain the secret in spectre-victim.o other than by using the Spectre exploit. For example, you could disassemble the .o file, as you did for another part of this homework. For this part of the assignment, don't do that -- instead, use the Spectre exploit.
- You are welcome to read more about Spectre online, including by reading the original publication about Spectre: https://spectreattack.com/spectre.pdf. However, don't look at Appendix A, the sample code provided there. You're on the honor system here. We hope that everyone in the class can try to work through the puzzle provided in this assignment without jumping straight to the solution.