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, December 8, 2017, 8pm.
- **Group or Individual:** Individual
- **How to Submit:** Submit to the Catalyst dropbox: https://catalyst.uw.edu/collectit/dropbox/franzi/40881.
- **Total Points:** 65 points across 5 parts

Part 1: Enigma Video (10 points)

This part of the assignment is a replacement for the pre-Thanksgiving class period. You should not spend more than 50 minutes on it.

This part of the assignment aims to give you exposure to a current topic in the broader field of computer security and privacy. Take a look at the programs for the first two years of the USENIX Enigma conference, which is intended to be something like the "TED Talks of security":
https://www.usenix.org/conference/enigma2016/conference-program
https://www.usenix.org/conference/enigma2017/conference-program

What to Submit:

Choose a video that sounds interesting (all talks are 20 minutes long), watch it, and answer the following questions:

1. **(1 point):** Which talk did you watch (please include title, speaker name, and link)?
2. **(5 points):** In 1-3 sentences, what was the core message or lesson that you took away from the talk?
3. **(4 points):** What 2 questions would you like to have asked the speaker?

Part 2: 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 3: 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 4: 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 5: Fuzz Testing (20 points)

Throughout the course, we have been manually analyzing code for weaknesses and designing targeted exploits for each program. In this problem, you will gain experience using a fuzzing tool developed by Microsoft (and described in David Molnar's guest lecture) to find vulnerabilities for you. For more details on how to use the fuzzing service and what it provides, and to supplement our instructions below, see here: https://docs.microsoft.com/en-us/security-risk-detection/.

Getting an Account

To receive access to the fuzzing service, you must provide us with an email address linked to a Microsoft account via the form sent via the course mailing list. *The sooner you submit this form, the sooner we can get you access and deal with any possible access issues!*

Detailed Instructions

For this assignment, you'll be trying out the tool on a calendar application that has intentional bugs planted in it. Follow the instructions below to build the calendar app and run the fuzzing service on it.

Note: Our goal is *not* for you to have a spend a lot of time dealing with configuration issues, but rather to learn how to use this tool so that you can apply it to other projects in the future. So if any step below starts taking you a seemingly non-trivial amount of time to figure out, please check in with us -- ideally on the Catalyst forum, so that others can benefit from the resolution.

~~1. Log in to https://demofuzz.visualstudio.com.~~
~~2. Navigate to the code tab and familiarize yourself with the BuildCalendar source code.~~
~~3. Proceed to the Build and Release tab. You should see a BuildCalendar item. Click on that item and "Queue new build" to spin up a machine in Azure to build the project. (You can keep the default options provided.)~~
~~4. Once your build is complete, from the Builds page, click your build to open a full report. Download the "Artifacts" from the build in a zip.~~
~~5. From the repo ("Code" tab), download the seeds file named "valid" (in msrd-demo/win-native-std/seeds). These will be used to create the test scenarios in the fuzzer.~~

Updated instructions below!

1. Familiarize yourself with the source code of the CalendarReader sample app, which contains planted bugs, available here.
2. Download a build of this application from here.
3. Download the seed files named "valid" from here. They are also in the repo here. These will be used to create the test scenarios in the fuzzer.
4. Log in to https://www.microsoftsecurityriskdetection.com/
5. Create a new job. This will provision a VM in Azure where you will create your fuzzing job.
6. Once the remote machine has become available, download and open the RDP file.

    i. *For non-Windows users, install your favorite RDP client for this step. We've confirmed that this works with Microsoft's RDP app for Mac, for example. However, you may wish to perform the following steps on a Windows machine if available to you.*
7. Upload the CalendarReader build and seed files (downloaded in steps 2 and 3 above) to the VM.
    i. *On Windows, you can copy and paste from your local machine. On a Mac using Microsoft's RDP app, follow these instructions to share a drive, and then relaunch the RDP connection via the RDP app: https://apple.stackexchange.com/questions/7949/how-can-i-copy-files-from-my-mac-to-pc-via-microsoft-remote-desktop.*
8. Fill out the fuzzing wizard following the guidelines it gives you (be sure to save and unzip the executable and seeds outside of the User directory, e.g., in "C://"). You do not need to change any of the parameters when the wizard asks you multiple choice questions.
9. When validating your setup at the end, you may receive an error for a missing dll. It turns out your target program needs to have all dependencies installed so the fuzzer can run it. To fix this, in the VM or on your local machine, download **Microsoft Visual C++ Redistributable for Visual Studio 2017 x86** from https://visualstudio.com/downloads/ and install the package on your VM.
10. Validate the job again and submit if no errors persist. The RDP connection will automatically close, and you can track the job back on https://www.microsoftsecurityriskdetection.com/.
11. The fuzzer will run for 2 weeks to exhaust millions of test cases. In practice, in this case, you will find most or all bugs in the sample program within 1-2 hours.

What to Submit

1. **(7 points)** Choose one of the results files to download. Submit the severity, type, hash, and callstack of that result, along with the zip file you download. We will also check the output of your job server-side.
2. Answers to the following questions (don't forget to check out the documentation at https://docs.microsoft.com/en-us/security-risk-detection/, among other possible sources of answers):
    a. **(1 point)** Briefly (in ~1 sentence), what is fuzzing?
    b. **(2 points)** Name at least 2 pros of fuzzing.
    c. **(2 points)** Name at least 2 cons of fuzzing (e.g., types of bugs it isn't good for, or things it doesn't help you with).
    d. **(2 points)** In the above process, what are the seed files, and why are they needed?
    e. **(2 points)** If we had asked you to reproduce one of the crashes on a local version of the program, given the output of the fuzzing service, how would you have done that?
    f. **(2 points)** If you had to repeat fuzzing on a new version of this application, how would you use the output of the previous job to re-fuzz it more effectively?
    g. **(2 points)** Fuzzing is useful for both attackers and defenders. If you were using it as an attacker, what would you do given the output of the fuzzing service?
    h. **(Optional)** Do you have any feedback or other thoughts to share about your use of this service?

Optional / Extra Credit
(at least 5 points, at our discretion, depending on how in depth you go with this):
Try applying the fuzzing service to an open source codebase of your choice! Please submit:

1. The name and link of the repository you're testing,
2. The output you receive from the fuzzing tool, and
3. If applicable, a short writeup of any interesting bugs or potential vulnerabilities you find (this step may require installing Visual Studio and actually checking out the bugs in the source).