Week 1 exercise: Fuzzing

The target of this week’s exercise is to use a fuzzer to generate broken data, to feed those to a target program, and potentially find issues.

We will be using a fuzzer created at the University of Oulu called Radamsa. It is a file-based fuzzer that takes in a set of valid example files ("valid cases") and output an arbitrary number of invalid files. Radamsa doesn't inject the bad data to the target software; this is something that would need to be done by its user. In this exercise, injection will be manual. In real life, you would need to automate it.

Note that if you have an antivirus software installed, it may react to your fuzz test cases by deleting or quarantining them, as they may flagged as false positives. Also, if you do this on a managed computer such as your work desktop, this may cause a remote virus alarm to sound at your IT department. This is obviously not optimal. However, I also do not recommend disabling your antivirus software, so the solution really is to use an operating system that has no (need for an) antivirus software.

Obtaining Radamsa

Obtain Radamsa as a current Git snapshot, or a pre-built binary (version 0.3) from https://code.google.com/p/ouspg/wiki/Radamsa.

If you are running GNU/Linux or Windows, the pre-built binary may work. Otherwise, you would need to compile it from sources. See https://code.google.com/p/ouspg/wiki/Radamsa#Building_Radamsa but you might not want to run the installation as root - just put the resulting binary in bin/ into your path.

For the exercise, you are free to try to fuzz whatever program, with whatever input your program takes. If you don’t have anything special that you know you want to torture, I recommend you try fuzzing an image viewer. This is because:

1. It is easy to obtain valid case data for an image viewer.
2. It is easy to find broken image viewer that provide satisfying results. (Although getting software to crash is not a requirement for the exercise, it feels good.)

Obtaining valid case data

This example assumes you are fuzzing an image viewer.

Acquire properly licensed images from somewhere, for example, the Internet. If you want to fuzz JPEG images, then you should only acquire JPEGs.
In serious fuzzing work, you would at this stage perform corpus distillation. This means that you should determine which of the valid case JPEGs you have acquired actually cause processing that is different from the rest; you could try to find a set of JPEGs that maximises code coverage. This is something you don’t need to do for this weekly exercise, but if you want to, you would want to run your target program (or some other image viewer) under a dynamic analysis tool such as Valgrind. If you’re interested in this, ask the lecturer at the weekly exercise session.

You should find at least a couple of valid case sets of around 20 valid cases each. (If you would do serious stuff, you’d obtain thousands of valid cases and boil them down to something less, and then use subsets of that “high quality” corpus to run the fuzzer.)

**Obtaining fuzz cases**

Take a moment to familiarise yourself with the Radamsa command line. As the idea of the exercise is not to cut and paste commands, I don’t give you the right command here, but meditate on [https://code.google.com/p/ouspg/wiki/Radamsa#Output_Options](https://code.google.com/p/ouspg/wiki/Radamsa#Output_Options), type

```
radamsa --help
```

on the command line and try to figure out how you can create a few hundred fuzz test cases out of the two sets of ~20 valid cases you have.

Again, in real life, you would aim towards thousands or tens of thousands of fuzz cases, but you would also think about automation.

**Injecting fuzz cases to an image viewer**

**WARNING: DO NOT USE YOUR “PRODUCTION” IMAGE VIEWER OR IMAGE MANAGEMENT APPLICATION WHERE YOU HAVE IMPORTANT PICTURES ARCHIVED.** This includes your mobile phone, that might automatically create index thumbnails in your “gallery” app. It is possible that you corrupt the image viewer’s metadata database or thumbnail cache, and you can lose everything. Always use a test setup and throwaway data. And never use someone else’s system or account for testing.

Most image viewers will happily thumbnail a directory full of images, so you have multiple options:

1. Open your favourite / default image viewer and point it to the directory.
2. Obtain a very old image viewer and point it to the directory (gives you a stronger chance of seeing fireworks).
3. Copy the pictures on a memory card and put that into a crappy el cheapo phone and see whether its Gallery app stands. (Old Nokias are actually pretty robust. For a reason.) Again, there is a small (but existing) risk that you will brick your phone or it will require a
factory reset, so do not use a phone that you rely on or that has important data inside. If your phone doesn’t have a removable memory card, don’t use it for this stunt.

You will see a lot of broken images, and with luck, you see crashes, hangs, or something else.

If you can run the image viewer from a command line, and you’re interested in the details, you could run it under a debugger and get an exploitability assessment. Again, this is not required for this exercise, but help is available in the exercise face-to-face meeting if you’d like to venture here. Or, have a look at https://www.cert.org/vuls/discovery/triage.html.

To return

You need to provide a very short explanation (5 lines or less) that explains what you fuzzed and how. This explanation should contain the Radamsa command line; the number of valid cases and how you selected them; how many fuzz cases you created, and how you injected them to the target; and what the target was.

In addition, you need to return one (or both, if you like) of these for evaluation:

1. Either one example of a fuzz case that causes a crash or hang with a software that you tested. Put the test case into a zip file before submission, so it won’t get rendered by anything. Also explain what software (version, OS) it was that it crashes.
2. Or a screenshot of an application being run with the test data. This is also why we are recommending using an image viewer; there is something to take a screenshot from.