Assignment 1: Basic Fuzzing Framework (BFF)

Fall Semester 2017

Due: September 4, 8:00 a.m. Eastern Time

Corresponding Lecture: Lesson 3 (Random Testing)

Objective

The goal of this assignment is to gain familiarity with random testing and obtain hands-on experience with a fuzzing tool used in industry. Specifically, we will use CERT's Basic Fuzzing Framework (BFF) for fuzzing Linux applications. The BFF targets applications that consume files as input by mutating a set of seed files to probe for errors in handling improperly formatted or corrupted files. Of particular interest are errors discovered that could present a security vulnerability, which are labelled as such by the BFF.

Resources

1. About BFF:
   https://www.cert.org/vulnerability-analysis/tools/bff.cfm
2. About `pdfunite`:
   http://manpages.ubuntu.com/manpages/trusty/man1/pdfunite.1.html
3. About `pnmnorm`:
   http://manpages.ubuntu.com/manpages/precise/man1/pnmnorm.1.html

Setup

The BFF as well as the applications you will fuzz are pre-installed on the course VM. You only need to download, import, and start the VM to begin the assignment. Please refer to the course syllabus for information about the course VM.

Demonstration

First, follow these steps to see a demonstration of how to run a fuzzing campaign using the BFF. Open a terminal emulator and navigate to the `~/BFF` directory. Start the fuzzing campaign by executing the `batch.sh` shell script. This will run the default campaign, which fuzzes the ImageMagick utility with a variety of mutated input files.

NOTE: You may see OS warnings that applications have stopped working. You may safely close and ignore these warnings.

Throughout the test, you will see various mutated seed files successfully execute and some that will not. When a crash occurs, the test case causing the crash will be analyzed further and recorded. If a duplicate crash is encountered, the BFF will recognize the test case signature and avoid redundant analysis. Let the test run for five minutes, then stop the campaign by pressing CTRL + C in the terminal emulator.

Next, examine the results of the testing campaign. Navigate to the `~/BFF/results` directory in a file manager window. Inside, you should see a result folder for `convert_v5.2.0`. Within this folder, you will see the following (not an exhaustive list):

1. `bff.yaml`: A copy of the campaign file the BFF used to execute the fuzzing campaign.
2. `uniquelog.txt`: A log of the details of unique crashing test cases.
3. `/crashers`: A subfolder containing classification and analysis output per unique test case, including (not an exhaustive list):
    a. Output from Valgrind
    b. Output from Callgrind
    c. Log output from the test case minimizer
    d. A minimized input to reproduce the crash
4. `/seedfiles`: A subfolder containing the seed files used during fuzzing.

In this demonstration, you used the default fuzzer, a byte mutator that randomly replaces bytes in the seed files to generate test cases. There are a variety of other fuzzers available that you can select based on the application being tested. You can find descriptions of these fuzzers in the campaign file (`bff.yaml`) comments.

Assignment Instructions

In this assignment, you will configure the BFF to fuzz two Linux command line utilities, `pdfunite` and `pnmnorm`, and analyze the results. To configure your campaigns, you will need to edit the campaign file (`bff.yaml`) located in the `~/BFF/configs` directory. Seed files to use in fuzzing these utilities have been provided for you in the `~/BFF/seedfiles` directory.

Run the BFF on each utility for five minutes per fuzzer, using each of the following fuzzers:

1. bytemut
2. drop
3. insert
4. truncate

NOTE: The results of a fuzzing campaign will be added to existing data if it exists, so there is no need to clear the results directory when changing fuzzers.

After collecting your data, analyze the results of the generated output in order to answer the questions below.

Items to Submit

1. (10 points) In a compressed file (.zip format) named `pdfunite.zip`, include the following:

   a. The `bff.yaml` campaign file you created to run the bytemut fuzzing campaign on `pdfunite`.

   b. The final `uniquelog.txt` file documenting the unique crashing test cases for `pdfunite` across all four fuzzers.

2. (10 points) In a compressed file (.zip format) named `pnmnorm.zip`, include the following:

   a. The `bff.yaml` campaign file you created to run the bytemut fuzzing campaign on `pnmnorm`.

   b. The final `uniquelog.txt` file documenting the unique crashing test cases for `pnmnorm` across all four fuzzers.

3. (80 points) Record your responses to the following questions in a PDF file named `responses.pdf`. We will grade your analysis on its quality and brevity. (Use at most 100 words per individual question or subquestion.) Make sure to answer all elements of each question.

   1. Answer the following questions about the results of your fuzzing campaigns on `pdfunite`.

   a. How many unique crashing test cases did the BFF generate across all four fuzzers? Was the time budget (20 minutes) too little or too much? Explain your answer.

   b. Were any of the fuzzers ineffective at causing crashes? If so, which ones?

   c. Under which classification did each crashing test case fall? (Exploitable, Unknown, etc.)

   d. Based on your data, how should the QA team prioritize finding and fixing these bugs? Would expending resources to find and fix these bugs be a prudent choice? Explain why or why not.

   2. Answer the following questions about the results of your fuzzing campaigns on `pnmnorm`.

   a. How many unique crashing test cases did the BFF generate across all four fuzzers? Was the time budget (20 minutes) too little or too much? Explain your answer.

   b. Were any of the fuzzers ineffective at causing crashes? If so, which ones?

   c. Under which classification did each crashing test case fall? (Exploitable, Unknown, etc.)

   d. Based on your data, how should the QA team prioritize finding and fixing these bugs? Would expending resources to find and fix these bugs be a prudent choice? Explain why or why not.

3. Consider a format-agnostic file-copy application that reads an input file as a stream of bytes and writes these bytes to a new location on disk. Would the BFF be a good candidate to fuzz this application? If so, describe how you would configure an effective fuzzing campaign. If not, why not?

4. Consider the crmut fuzzer, which replaces carriage return bytes with random values to generate test cases. Give an example of an input file format that crmut would be effective in mutating to generate error-revealing inputs. Also give an example of an input file format that crmut would not be effective in mutating to generate error-revealing inputs. Justify your answers.

5. Consider how you might use the BFF to test a piece of software that you have worked on in your professional or academic career, or use frequently in everyday life.
    a. What is the general purpose of the software, and what input does it consume?
    b. What kind of seed files would you provide to the BFF?
    c. Which fuzzers would you include in the campaign, and which would you exclude? Explain your reasoning.
    d. Would any campaign settings require changes to test the software effectively?
    e. What would be the implications if the BFF were to find exploitable crashing test cases in the software?