Online Markov Chain-driven Fuzzing

Fuzzing is a method of software testing that uncovers difficult to discover bugs in code by generating and running thousands of tests per second on an application.

One of the most important components of an effective fuzzer is the test generator, since we want to generate a large, varied set of inputs that span the entire possible input space of a program. If we want to fuzz a program that consumes a well-defined and understood input format, like a SQL parser, we can provide an EBNF grammar to a generator like Dharma (https://github.com/MozillaSecurity/dharma), which generates random inputs within a specified structure.

If we don't have a formal definition of an input structure, then we need to resort to other methodologies to generate interesting tests. If we have a large corpus of example inputs (for example, if we were fuzzing an HTTP server, and had a large collection of valid HTTP requests) one method of generating a large number of inputs could be via a Markov Chain generator trained on the corpus of examples.

Task 1: Markov input stream generator

Your first task is to build a server that consumes a large set of example inputs in a specific format, and provides a stream of new inputs, generated by a markov chain.

You should assume that the set of example inputs will be provided as flat files in a folder. So an invocation of this server via the command line may look like:
`./my_server --corpus-location ./corpus --port 8080.`

Where every file in `./corpus` is an example of an input. The user should be able to make a GET request to this server, and receive the following JSON blob: `{ "input": "<b64>" }` where the contents of *input* is a base64-encoded blob generated from a markov chain of all of the text in the `corpus` directory.

The details of this are largely up to you - you can use whatever programming language and framework you're most comfortable with to build the server. The specifics of how you build a Markov chain and use it to generate inputs is also up to you. There are Markov chain libraries available in most programming languages that can get you started, but we'll let you be creative when it comes to deciding how to tokenize the text, which knobs to tweak, and so on. We won't be judging your submission based on how "good" the generated inputs are since it is very difficult to accurately determine what a "good" algorithm looks like, given the current scope.

To test your server, you should use the Mozilla xml corpus (https://github.com/strongcourage/fuzzing-corpus/tree/master/xml/mozilla), a large set of files used to seed the fuzzing of XML parsers.

Task 2: Live updates

Another important feature of fuzzers is the ability to learn from past tests. If a specific input covers previously untested code, it should be added to the pool of inputs used by the markov chain generator so it has a wider selection of inputs to work with.

Extend the server you built in the previous test to allow for updates to the corpus of inputs the markov chain generator uses. The specifics of how you do this are up to you, but an example of how to do this may be to add an endpoint that allows users to POST a base64-encoded blob of text.

Optional task 3: Mutational fuzzing

Many fuzzers produce subtle bugs in code by “mutating” inputs that the application handles without crashing. These mutations can come in many forms, from flipping a single bit in the input, to splicing two different inputs together, or even replacing words with a known interesting combination (like replacing a word in a SQL query with “DROP TABLES;”).

How would you go about extending this server to mutate the inputs it generates? Can you think of any interesting mutation strategies or ways to corrupt inputs given the data you have?

There are no hard requirements for this task - if you choose to think about it, you could write up some thoughts in your README, or provide a small code sample, but there’s no need to implement something in great detail.

Submitting:

When you start off, initialize a new git repository, and use git how you normally would to develop the data generator. Be sure to add a README documenting dependencies, how to run the code, the routes you built and anything else we should know before running this. Keep in mind we’ll be running this on our machines, so make sure all the requirements & dependency files are updated appropriately.

When you’re ready to submit, push your code up to a private GitHub repository, and add the following GitHub users: everestmz, andofcourse, alexanderguy, peterfuzz and modalton. Send us an email when you’re finished, and we’ll clone the code and run it/give it a review.