FuzzyS2E How-to Documentation

In this documentation, we will show how to use FuzzyS2E to find bugs in software. Specifically, this documentation focuses on Linux software.

1. What is FuzzyS2E

S2E is built on top of symbolic execution and even though it has solved the path explosion problem to a large extent, but because the complex constraint solving problem is still not well handled in its solvers, like STP, so it may get stuck in some cases. While fuzzing test is a great way to find GREAT seed paths, so FuzzyS2E utilizes this advantage to help S2E avoid this problem, and in return, fuzzing test can be helped with the output of S2E to promote its path discovering ability.

FuzzyS2E is built on top of S2E and AFL, which is an open-sourced fuzzing test system. FuzzyS2E tries to exchange concrete testcases between S2E and AFL, when we start the testing procedure, S2E first starts its concolic testing procedure with a random concrete input testcase. Then as the testing goes on, S2E can generate several testcases for different paths, which can be used as the input of AFL, and then AFL starts its fuzzing test and generates more testcases based on these initial inputs. Note that during the whole procedure, AFL can dynamically exchange concrete testcases with S2E so that they can help each other. The detail technical documentation will be shown in another documentation soon.

2. How to test Linux software with FuzzyS2E

2.1 Configure for S2E

Our testing focuses on binary software testing and you can first compile your source code to binary executable file or you can choose to test binary software directly. In order to start your testing, you should first prepare with basic VM image for S2E as shown in S2E's documentation.

After that, you need to complete the config-file of FuzzyS2E, in this phase, you have to make sure that each plug-in of FuzzyS2E should be configured correctly. Because FuzzyS2E runs in concolic mode, so you have to specify the KLEE arguments as follows. Note that whether to specify DFS search is not very important.

```
s2e = {
    kleeArgs = {
        "--use-concolic-execution=true",
        "--use-dfs-search=true",
    }
}
```

FuzzyS2E uses three main plug-ins to finish its testing, i.e. **AutoShFileGenerator, FuzzySearcher** and **ForkController.** Let us do with each plug-in in detail.

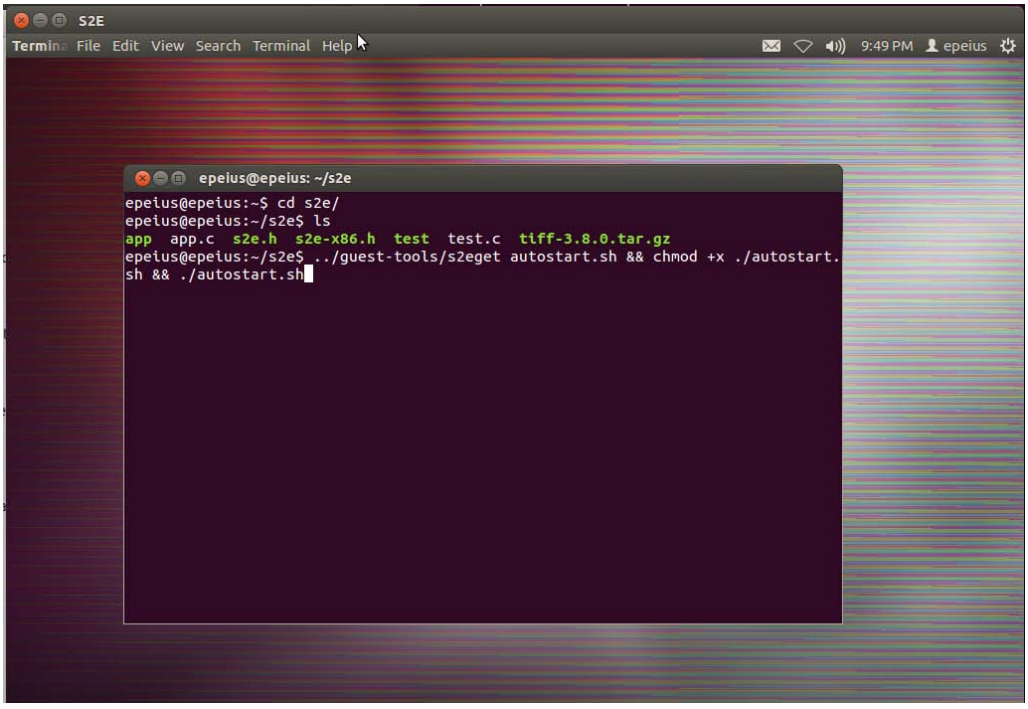
[AutoShFileGenerator]

AutoShFileGenerator tries to generate a shell script so that the testing and its iteration can start automatically to avoid tedious manual work. An example configure file of AutoShFileGenerator is shown as follows.

```
command_str = '#!/bin/bash \n /guest/path/to/s2eget aa.txt && cp aa.txt /tmp/aa.txt && /guest/path/to/s2ecmd symbfile /tmp/aa.txt && LD_PRELOAD=/guest/path/to/init_env.so /guest/path/to/app --select-process-code /tmp/aa.txt',
command_file = "/host/path/to/autostart.sh",
```

"**command_str**" infers the detail content of shell script. In order to exchange concrete testcase with AFL, which means FuzzyS2E has to exchange files with host OS dynamically during the testing, you have to get the testcase ("/guest/path/to/s2eget aa.txt") and copy it to a ramdisk ("cp aa.txt /tmp/aa.txt") and then mark the testcase as symbolic ("/guest/path/to/s2ecmd symbfile /tmp/aa.txt") and finally start the binary target code ("LD_PRELOAD=/guest/path/to/init_env.so /guest/path/to/app --select-process-code /tmp/aa.txt"). All this has been written to a shell script file in host as "**command_file**"("/host/path/to/autostart.sh").

Then FuzzyS2E's guest will get this automatically generated shell script file ("/host/path/to/autostart.sh") though **HostFiles** plug-in and some command line in guest's shell. The guest's shell command line is show in the following figure.



[FuzzySearcher]

FuzzySearcher is the core search plugin for FuzzyS2E, it tries to schedule all the execution states and communicate with AFL. An example configure of FuzzySearcher is shown as follows.

```
symbolicfilename = "aa.txt",
inicasepool = "/host/path/to/initialtestcasepool",
curcasepool = "/host/path/to/curtmp",
aflRoot="/host/path/to/afl-1.96b",
aflOutput="/host/path/to/afloutput",
aflBinaryMode=true,
aflAppArgs="/host/path/to/app @@",
mainModule="app";
autosendkey_enter = true,
autosendkey_interval = 10,
MaxLoops = 50,
```

"**symbolicfilename**" infers to the testcase name in AutoShFileGenerator.
"**inicasepool**" infers to the directory which stores the initial testcase for S2E in host OS.
"**curcasepool**" infers to the directory which stores the current testcase for S2E in host OS.
"**aflRoot**" infers to the root directory of AFL in host OS.
"**aflOutput**" infers to the output directory of AFL in host OS.
"**aflBinaryMode**" represents whether we want to test binary-only software, please set this always to be **TRUE**, because we focus on binary software testing.
"**aflAppArgs**" infers to the whole command line arguments for target binary and you should replace the input file argument with "@@", as documented in AFL.
"**mainModule**" infers to our target binary name.
"**autosendkey_enter**" will automatically send "enter" to guest OS if set to be **TRUE,** otherwise, you have to send this key manually before each iteration.
"**autosendkey_interval**" infers to the time interval before we send "enter" to guest OS.
"**MaxLoops**" specifies the stop condition, and we currently stop the testing procedure when it reaches to maximum iteration numbers.

[ForkController]
S2E can control the fork in several levels of testing grain, like "select-process-code(target binary only)", "select-process-user(target process in user mode)", but ForkController can give a more fine grained fork control, it can restrict the fork in a code region, An example configure of ForkController is shown as follows.

```
forkRanges ={
    r01 = {0x8048000, 0x8049000},
},
```

"**forkRanges**" infers to the code regions that you want FuzzyS2E to fork in.

A sample complete configure file can be written as follows.

```
s2e = {
   kleeArgs = {
      "--use-concolic-execution=true",
      "--use-dfs-search=true",
   }
}

plugins = {
   "BaseInstructions",
   "HostFiles",
   "RawMonitor",
   "ModuleExecutionDetector",
   "CodeSelector",
   "AutoShFileGenerator",
   "FuzzySearcher",
   "ForkController",
}

pluginsConfig = {}

pluginsConfig.HostFiles = {
   baseDirs = {"/path/to/host ", "/host/path to/cur"}
}
pluginsConfig.CodeSelector = {
}
pluginsConfig.RawMonitor = {
      kernelStart = 0xc0000000,
}

pluginsConfig.ModuleExecutionDetector = {
   trackAllModules=false,
   configureAllModules=false,
}

pluginsConfig.ForkController = {
   forkRanges ={
      r01 = {0x8048000, 0x8049000},
   },
}

pluginsConfig.FuzzySearcher = {
   symbolicfilename = "aa.txt",
   inicasepool = "/host/path/to/initialtestcasepool",
   curcasepool = "/host/path/to/curtmp",
   aflRoot="/host/path/to/afl-1.96b",
   aflOutput="/host/path/to/afloutput",
   aflBinaryMode=true,
   aflAppArgs="/host/path/to/app @@",
   mainModule="app";
   autosendkey_enter = true,
   autosendkey_interval = 10,
   MaxLoops = 50,
}
```

```
pluginsConfig.AutoShFileGenerator={
   command_str = '#!/bin/bash \n /guest/path/to/s2eget aa.txt && cp aa.txt /tmp/aa.txt && /guest/path/to/s2ecmd symbfile /tmp/aa.txt && LD_PRELOAD=/guest/path/to/init_env.so /guest/path/to/app --select-process-code /tmp/aa.txt',
   command_file = "/host/path/to/autostart.sh",
}
```

2.2 Configure for AFL

We have added and modified some codes in AFL so that it can exchange some runtime information with S2E, so you have to do some initial configure to AFL, that is **CREATE** "/tmp/aflbitmap" file in host OS.

Then start FuzzyS2E as what we do in S2E.

Have fun.