



Host-based Firmware Analyzer User Guide

This paper is for informational purposes only. THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

Intel, the Intel logo, Intel. leap ahead. and Intel. Leap ahead. logo, and other Intel product name are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2019 Intel Corporation. All rights reserved.

Contents

1	Introduction	1
1.1	Overview	1
1.2	Terms	1
1.3	Related Information	2
1.4	Conventions Used in this Document	2
1.4.1	Typographic Conventions.....	2
2	Prerequisite	4
2.1	Third-Party Tools	4
2.1.1	Windows	4
2.1.1.1	AFL	4
2.1.1.2	DynamoRIO	4
2.1.1.3	LLVM.....	4
2.1.1.4	Debugging Tools for Windows.....	5
2.1.1.5	Peach	5
2.1.2	Linux	5
2.1.2.1	AFL	5
2.1.2.2	KLEE	6
2.1.2.3	LLVM.....	6
2.1.2.4	Peach	6
2.1.2.5	LCOV.....	7
2.2	Setup EDKII Environment	7
3	Quick Start Guide	8
3.1	Setup Host-based Firmware Analyzer Environment	8
3.2	Graphical User Interface	9
3.2.1	Overview	9
3.2.1.1	Introduction	9
3.2.1.2	Config File	10
3.2.2	Run Test Case.....	10
3.2.2.1	Run Test with AFL.....	10
3.2.2.2	Run Test with libFuzzer.....	13
3.2.2.3	Run Test with Peach + Sanitizer	14
3.2.2.4	Generate Test Seed by KLEE	17
3.3	Commandline Interface	18
3.3.1	Run Test Case.....	18
3.3.1.1	Run Test with AFL.....	18
3.3.1.2	Run Test with libFuzzer.....	24
3.3.1.3	Run Test with Peach + Sanitizer	27
3.3.1.4	Generate Test Seed by KLEE	29
3.3.1.5	Run Test with ErrorInjection.....	32
Appendix A	List of All Examples.....	36

FIGURES

Figure 1. Main GUI.....	9
Figure 2. AFL Running Status	11
Figure 3. AFL Summary Report.....	12
Figure 4. Debug Report for Crashes	12
Figure 5. Debug Report for Hangs	12
Figure 6. Code Coverage Report	13
Figure 7. libFuzzer Output.....	14
Figure 8. Peach+Sanitizer Running Status	15
Figure 9. Peach Catch an Error	16
Figure 10. KLEE Running Status	17
Figure 11. KLEE Seeds	18
Figure 12. Windows AFL Running Status	21
Figure 13. Code Coverage Report 1.....	21
Figure 14. Code Coverage Report 2.....	21
Figure 15. AFL Running Status	22
Figure 16. AFL Summary Report.....	23
Figure 17. Debug Report for Crashes.....	23
Figure 18. Debug Report for Hangs	24
Figure 19. Code Coverage Report 1.....	24
Figure 20. Code Coverage Report 2.....	24
Figure 21. libFuzzer Output.....	26
Figure 22. Peach+Sanitizer Running Status.....	29
Figure 23. KLEE Running Status	31
Figure 24. Generated “.ktest” Files.....	31
Figure 25. Generated “.seed” Files	31
Figure 26. KLEE Code Coverage Report	32
Figure 27. Code Coverage without Running ErrorInjection.....	34
Figure 28. Code Coverage after Running ErrorInjection	35

Revision History

Revision Number	Description	Revision Date

1

Introduction

1.1 Overview

Host-based Firmware Analyzer is a tool that focuses on the pure software logic. Users can use it to implement security test in OS, which can improve test efficiency and extensibility greatly. This documents introduces the detail steps to run test cases in it, before which this document also explains how to setup environment that it requires.

1.2 Terms

The following terms are used throughout this document to describe varying aspects of input localization:

OS

Operation system.

Fuzzer

Tools that can do fuzzing test.

AFL

American fuzzy lop. It is a security-oriented fuzzer that employs a novel type of compile-time instrumentation and genetic algorithms to automatically discover clean, interesting test cases that trigger new internal states in the targeted binary.

libFuzzer

In-process, coverage-guided, evolutionary fuzzing engine.

Peach

Peach accomplishes fuzzing test by separating out modeling of the data and state systems being fuzzed and the actual fuzzing engine.

KLEE

KLEE is a symbolic virtual machine built on top of the LLVM compiler infrastructure, and available under the UIUC open source license.

Clang

A compiler which is default for Mac OS X.

Sanitizer

AddressSanitizer (or ASan) is an open source programming tool by Google that detects memory corruption bugs such as buffer overflows or accesses to a dangling pointer (use-after-free).

Test seeds

The input files which can walk through almost all test logic and is used to be an initial template for fuzzer.

Test cases

The test cases under HBFA/UefiHostFuzzTestCasePkg/TestCase.

Test binary

The binary file built from test case and can be run in OS.

1.3 Related Information

The following publications and sources of information may be useful to you or referred to by this document:

- *Wikipedia*, a free-access, free content Internet encyclopedia, supported and hosted by the non-profit Wikimedia Foundation. Those who can access the site and follow its rules can edit most of its articles, <https://www.wikipedia.org/>.
- *American fuzzy lop (afl)*, the official website of AFL, from where you can find all official documents about AFL, <http://lcamtuf.coredump.cx/afl/>.
- *libFuzzer*, the official page of libFuzzer, <https://llvm.org/docs/LibFuzzer.html>.
- *Peach Fuzzer*, provides tutorials information on how to get started fuzzing with Peach 3, <http://community.peachfuzzer.com/v3/PeachQuickStart.html>.
- *KLEE*, documentation for KLEE master branch, <http://klee.github.io/>.
- *Clang*, Getting Started: Building and Running Clang, give the shortest path to checking out Clang and demos a few options, http://clang.llvm.org/get_started.html.

1.4 Conventions Used in this Document

This document uses typographic and illustrative conventions described below.

1.4.1 Typographic Conventions

Typographic Convention

Typographic Convention Description

Plain Text

The normal text typeface is used for the vast majority of the descriptive text in a specification.

[Plain Text \(blue\)](#)

Any [plain text](#) that is underlined and in blue indicates an active link to the cross-reference.

Bold

In text, a **Bold** typeface can be used as a running head within a paragraph.

Typographic Convention

Typographic Convention Description

Italic

In text, an *Italic* typeface can be used as emphasis to introduce a new term or to indicate a manual or specification name.

BOLD Monospace

Computer code, example code segments, and all prototype code segments use a **BOLD Monospace** typeface with a dark red color. These code listings normally appear in one or more separate paragraphs, though words or segments can also be embedded in a normal text paragraph.

Bold Monospace

Words in a **Bold Monospace** typeface that is underlined and in blue indicate an active hyperlink to the code definition for that function or type definition. Click on the word to follow the hyperlink.

\$(VAR)

This symbol VAR defined by the utility or input files.

Italic Monospace

In code or in text, words in *Italic Monospace* indicate placeholder names for variable information that must be supplied (i.e., arguments).

Note: Due to management and files size considerations, only the first occurrence of the reference on each is an active link. Subsequent references on the same page will not be actively linked to the definition and will use the standard, none-underlined **BOLD Monospace** typeface. Find the first instance of the name (in the underlined **BOLD Monospace** typeface) on the page and click on the word to jump to the function or type definition.

2

Prerequisite

This chapter explains how to download the third-party software and EDKII environment needed by the Host-based Firmware Analyzer run process.

2.1 Third-Party Tools

2.1.1 Windows

2.1.1.1 AFL

User can download latest winafl package from <https://github.com/googleprojectzero/winafl> and setup AFL environment.

After AFL setup done, please add AFL installed location to system environment variable:

```
set AFL_PATH=<AFL_PATH>
```

For example:

```
set AFL_PATH=C:\winafl
```

As AFL test should use DynamoRIO, please also refer to 2.1.1.2 and prepare DynamoRIO environment.

2.1.1.2 DynamoRIO

Download DynamoRIO release from <https://github.com/DynamoRIO/dynamorio/wiki/Downloads> and setup DynamoRIO environment.

After DynamoRIO setup done, please add DynamoRIO installed location to system environment variable:

```
set DRIO_PATH=<DynamoRIO_PATH>
```

For example:

```
set AFL_PATH=C:\DynamoRIO-Windows-x.x.x-x
```

2.1.1.3 LLVM

Download LLVM 8.0.0 windows pre build binary from <http://releases.llvm.org/download.html> and setup LLVM environment.

Note: Please install 64bit exe for X64 build and 32bit exe for IA32 build.

After LLVM setup done, please add LLVM installed location to system environment variable:

```
set LLVM_PATH=<32_LLVM_PATH>
set LLVMx86_PATH =<64_LLVM_PATH>
For example:
set LLVM_PATH=C:\Program Files\LLVM
set LLVMx86_PATH=C:\Program Files (x86)\LLVM
```

And then, add %LLVM_PATH%\bin, %LLVM_PATH%\lib\clang\8.0.0\lib\windows and %LLVMx86_PATH%\lib\clang\8.0.0\lib\windows to system environment variable PATH.

2.1.1.4 Debugging Tools for Windows

Download Debugging Tools for Windows from <https://docs.microsoft.com/zh-cn/windows-hardware/drivers/debugger/debugger-download-tools> and setup Debugging Tools for Windows environment.

2.1.1.5 Peach

Download Peach windows pre build binary from <http://www.peach.tech/resources/peachcommunity/> and setup Peach environment.

After Peach setup done, please add Peach installed location to system environment variable.

```
set PEACH_PATH=<PEACH_PATH>
For example:
set PEACH_PATH=C:\PEACH
```

And then, add %PEACH_PATH% to system environment variable PATH.

As some test will use CLANG as tool chain, please also refer to 2.1.1.3 and prepare LLVM 8.0.0 environment.

As windows Peach should use Debugging Tools for Windows to monitor program running information, please also refer to 2.1.1.4 and install Debugging Tools for Windows.

2.1.2 Linux

Note: For the following third-party tools, if there is a requirement to modify ~/.bashrc, please reopen Terminal to make the modification take effect.

2.1.2.1 AFL

AFL is available at: <http://lcamtuf.coredump.cx/afl/>. After downloading latest package, follow docs/QuickStartGuide.txt in AFL package to quickly make AFL.

After AFL setup done, please add below content at the end of ~/.bashrc:

```
export AFL_PATH=<AFL_PATH>
export PATH=$PATH:$AFL_PATH
For example:
export AFL_PATH=/home/tiano/Env/afl-2.52b
```

```
export PATH=$PATH:$AFL_PATH
```

And you need to run these commands as root (these command need to be ran every time you reboot your OS):

```
echo core >/proc/sys/kernel/core_pattern  
cd /sys/devices/system/cpu  
echo performance | tee cpu*/cpufreq/scaling_governor
```

2.1.2.2 KLEE

KLEE is available at <http://klee.github.io/>, user can choose to follow <http://klee.github.io/build-llvm38/> to setup KLEE environment.

Note: In above link, please remove "llvm-3.8-tools" when you use command in step. 2, and we suggest you to choose "STP" as your solver which is verified on our platform in step. 3.

After KLEE setup done, please add below content at the end of ~/.bashrc:

```
export KLEE_BIN_PATH=<KLEE_BUILD_DIR>/bin  
export KLEE_SRC_PATH=<KLEE_SRC_DIRECTORY>
```

For example:

```
export KLEE_BIN_PATH=/home/tiano/Env/klee_build_dir/bin  
export KLEE_SRC_PATH=/home/tiano/Env/klee
```

2.1.2.3 LLVM

Download LLVM 8.0.0 linux pre build binary from <http://releases.llvm.org/download.html> and setup LLVM environment.

After LLVM setup done, please add LLVM installed location at the end of ~/.bashrc:

```
export CLANG_PATH=<LLVM_CLANG_BUILD_DIR>/bin  
export ASAN_SYMBOLIZER_PATH=$CLANG_PATH/llvm-symbolizer
```

For example:

```
export CLANG_PATH=/home/tiano/ Env/clang+llvm-8.0.0-x86_64-linux-gnu-  
ubuntu-16.04/bin  
export ASAN_SYMBOLIZER_PATH=$CLANG_PATH/llvm-symbolizer
```

2.1.2.4 Peach

Before get Peach package, you need to install mono in your OS:

```
sudo apt-get install mono-complete
```

Then, you can download Peach package from 'Project Activity' in

<https://sourceforge.net/projects/peachfuzz/>.

After unzipping Peach package, please add below content at the end of ~/.bashrc:

```
export PEACH_PATH=<PEACH_PATH>  
export PATH=$PATH:$PEACH_PATH
```

For example:

```
export PEACH_PATH=/home/tiano/Env/peach  
export PATH=$PATH:$PEACH_PATH
```

As some test will use CLANG as tool chain, please also refer to 2.1.2.3 and prepare LLVM 8.0.0 environment.

2.1.2.5 LCOV

LCOV is available at <http://tp.sourceforge.net/coverage/lcov.php>, you can get LCOV with below command in Linux:

```
sudo apt-get install lcov
```

2.2 Setup EDKII Environment

You can download the EDKII source tree from <https://github.com/tianocore/tianocore.github.io/wiki/EDK-II>. Please refer to the <https://github.com/tianocore/tianocore.github.io/wiki/Getting-Started-with-EDK-II> and setup EDKII environment.

3

Quick Start Guide

This chapter assumes readers have already setup environment correctly and can build EDKII successfully. This guide gives detailed steps and instruction to run test cases in Host-based Firmware Analyzer.

3.1 Setup Host-based Firmware Analyzer Environment

Step 1: Get Host-based Firmware Analyzer source code.

Step 2: After getting HBFA package, please remember to add HBFA path into your PACKAGES_PATH when you want to build your tree along with HBFA:

```
export WORKSPACE=<YOUR_WORKSPACE_PATH>
export PACKAGES_PATH=<HBFA_PATH>:<YOUT_PACKAGES_PATH>
```

For example:

```
export WORKSPACE=/home/tiano/edk2
export PACKAGES_PATH=/home/tiano/HBFA:/home/tiano/edk2
```

Step 3: Setup build environment by calling edksetup.sh or edksetup.bat:

For example:

```
. edk2/edksetup.sh
```

Step 4: Setup HBFA environment by calling HBFAEnvSetup.py under UefiHostTestTools:

For example:

```
python HBFA/UefiHostTestTools/HBFAEnvSetup.py
```

Step 5: After setting up environment, here are two options to start up test cases: 1) Graphical User Interface (Chapter 3.2.1); 2) Commandline Interface (Chapter 3.2.2).

Note: If you want to run test cases with GUI, please according to <https://www.wxpython.org/pages/downloads/> to install wxpython to your python.

3.2 Graphical User Interface

3.2.1 Overview

3.2.1.1 Introduction

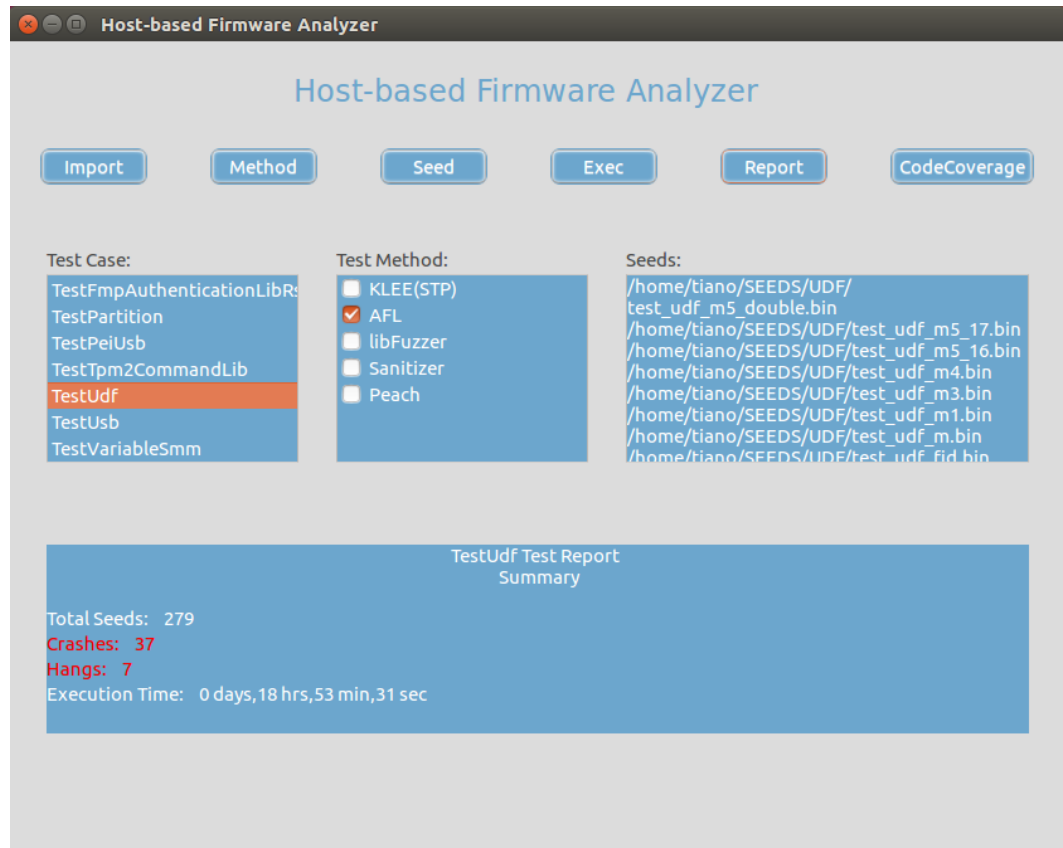


Figure 1. Main GUI

Above picture is main GUI for 'Host-based Firmware Analyzer' tool.

'**Import**' button is used to import test cases , after clicking it, there will be 5 test cases (TestTpm2CommandLib, TestUdf, TestPartition, TestUsb, TestBmpSupportLib) displaying on GUI. For example, user can choose 'TestUdf'.

'**Method**' button is used to select different test methods, after clicking it, there will be 5 test methods (KLEE(STP), AFL, libFuzzer, Sanitizer, Peach) displaying on GUI. For example, user can choose 'AFL'.

'**Seed**' is used to select seeds as test inputs. When click seed button, it will open browser window for user to choose the seed.

'**Exec**' is used to kick off running test cases.

'**Report**' is used to display summary and report.

'**CodeCoverage**' button is used to get code coverage.

3.2.1.2 Config File

Before running Host-based Firmware Analyzer GUI tool, user can config some info in `HBFA/UefiHostTestTools/HBFAGUI/Env.conf`, such as:

1) 'OutputPath' is output folder for results, if not change, it will run with default ramdisk path: `/dev/shm`.

Note: we suggest you to set this path on a ramdisk, because a high number of read and write operations will harm harddisk.

2) 'Arch', 'BuildTarget' for [afl], [klee], [peach_sanitizer], [libFuzzer], if not change them, it will be ran with default value.

Note: for [peach_sanitizer] and [libFuzzer], arch must be X64, otherwise, module will fail to be built.

3.2.2 Run Test Case

A pop-up GUI will display after running this command:

```
python HBFA/UefiHostTestTools/HBFAGUI/main.py
```

3.2.2.1 Run Test with AFL

General Steps:

Step 1: Click '**Import**', choose test cases.

Step 2: Click '**Method**', choose 'AFL'.

Step 3: Click '**Seed**' to choose test seeds, user can use prepared seeds or seeds generated by KLEE (refer to 3.2.2.4).

Step 4: Click '**Exec**', a new pop-up terminal will display.

Step 5: User can stop it by 'Ctrl C' in this terminal.

Step 6: After finishing running AFL, Click '**Report**' button to check summary report, and click 'Crashes Case Number' or 'Hangs Case Number' in summary report. User can get detailed info about crashes or hangs, such as case name, file name, line number, error message and stack information: click seed name, it will display the position of directory contain seeds; click file name, it will display source code file; and click '+', detail stack information will display.

Step 7: On GUI, click '**CodeCoverage**' button. It will display code coverage report of all seeds generated by AFL.

Note: All reports are generated in the `afl_XXX(XXX means test case name)` folder under 'OutputPath' you set in 3.2.1.2.

Example:

Step 1: Click '**Import**', choose 'TestUdf'.

Step 2: Click '**Method**', choose 'AFL'.

Step 3: Click '**Seeds**', choose test seeds prepared under
HBFA/UefiHostFuzzTestCasePkg/Seed/UDF/Raw/FileSystem

Step 4: Click '**Exec**', a new pop-up terminal will display, in this window, user can get summary info of test status, and more detailed information please refer to https://github.com/nccgroup/TriforceAFL/blob/master/docs/status_screen.txt.

```
american fuzzy lop 2.52b (TestUdf)

-- process timing --
run time : 0 days, 18 hrs, 53 min, 31 sec
last new path : 0 days, 0 hrs, 0 min, 0 sec
last uniq crash : 0 days, 0 hrs, 0 min, 0 sec
last uniq hang : 0 days, 4 hrs, 0 min, 52 sec
-- cycle progress --
now processing : 15 (5.38%)
paths timed out : 0 (0.00%)
-- stage progress --
now trying : bitflip 1/1
stage execs : 4.56M/8.39M (54.31%)
total execs : 122M
exec speed : 1169/sec
-- fuzzing strategy yields --
bit flips : 131/37.7M, 29/37.7M, 14/37.7M
byte flips : 2/4.72M, 1/1662, 5/1938
arithmetics : 23/85.3k, 11/100k, 4/80.7k
known ints : 9/4304, 5/20.9k, 4/52.4k
dictionary : 0/0, 0/0, 6/6580
havoc : 1/5816, 0/0
trim : 29.03%/11.2k, 99.97%

-- overall results --
cycles done : 0
total paths : 279
uniq crashes : 37
uniq hangs : 7

-- map coverage --
map density : 1.12% / 1.86%
count coverage : 2.85 bits/tuple

-- findings in depth --
favored paths : 90 (32.26%)
new edges on : 111 (39.78%)
total crashes : 9451 (37 unique)
total tmouts : 300 (7 unique)

-- path geometry --
levels : 2
pending : 272
pend fav : 87
own flnds : 253
imported : n/a
stability : 100.00%

[cpu001: 61%]
```

Figure 2. AFL Running Status

Step 5: After running for a period of time or 'cycles done' turns green, stop it by 'Ctrl C' in terminal.

Step 6: Click '**Report**' button, and summary report will display on GUI and a summary report web page is like Summary Report figure as below. Click 'Crashes Case Number' or 'Hangs Case Number', detailed info will display.

Host-based Firmware Analyzer TestUdf Summary Report		
Highlight: This is Host-based Firmware Analyzer TestUdf Summary Report offered the crashes and hangs number.		
Date: 2018-10-30 13:01:35		
ItemName	ItemValue	
Generated Seeds Total Number:	279	
Generated Crashes Number:	37	
Generated Hangs Number:	7	
TestCases Execution Time:	0 days,18 hrs,53 min,31 sec	

Generated by: Host-based Firmware Analyzer

Figure 3. AFL Summary Report

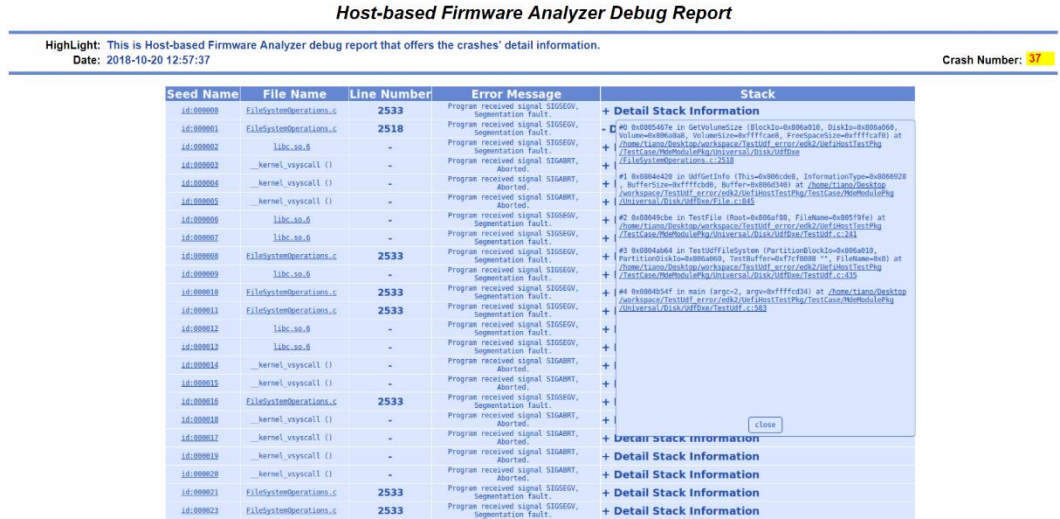


Figure 4. Debug Report for Crashes

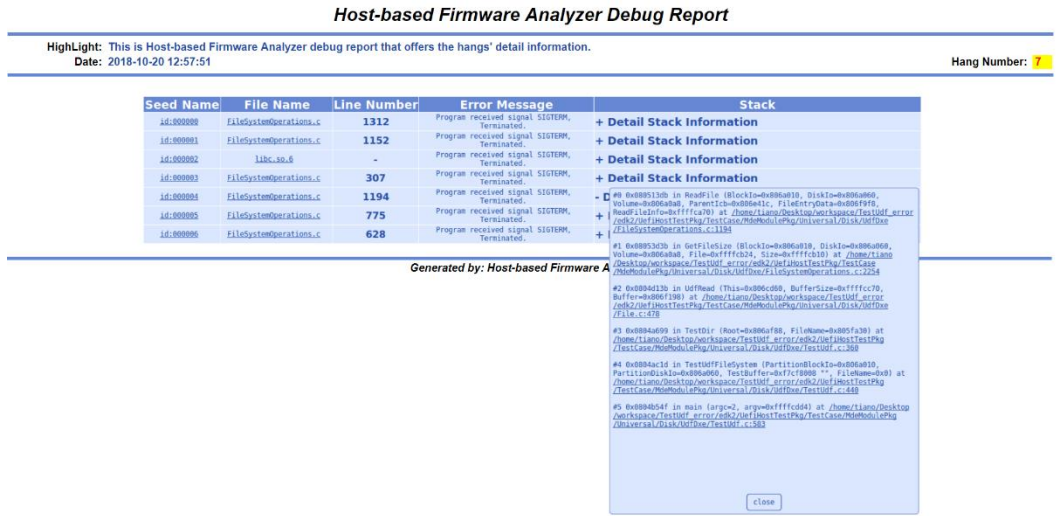
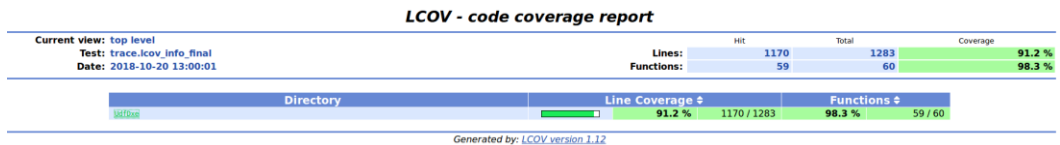


Figure 5. Debug Report for Hangs

Step 7: On GUI, click '**CodeCoverage**' button, LCOV – code coverage report will display in web page as shown below:



LCOV - code coverage report					
Current view: top level - UdfDxe					
Test: trace.lcov_info_final					
Date: 2018-10-20 13:00:01					
			Lines:	1170	1283
			Functions:	59	60
			Coverage		
			91.2 %		
			98.3 %		
Filename	Line Coverage	Functions			
File.c	96.2 %	256 / 266	100.0 %	11 / 11	
File.h	95.1 %	77 / 81	100.0 %	4 / 4	
FileOperations.c	92.5 %	637 / 689	100.0 %	33 / 33	
TestFile.h	75.0 %	18 / 24	100.0 %	2 / 2	
TestFile.c	81.6 %	182 / 223	90.0 %	9 / 10	

Generated by: LCOV version 1.12

Figure 6. Code Coverage Report

3.2.2.2 Run Test with libFuzzer

General Steps:

Step 1: Click '**Import**', choose test case.

Step 2: Click '**Method**', choose 'libFuzzer'.

Step 3: Click '**Seed**' to choose test seeds, user can use prepared seeds or seeds generated by KLEE (refer to 3.2.2.4).

Step 4: Click '**Exec**', a new pop-up terminal will display.

Step 5: User can stop it by 'Ctrl C' in new terminal.

Step 6: After finishing running libFuzzer, Click '**Report**' button to check summary report, and click 'Failure Type Number' in summary report. User can get detailed info about failures.

Step 7: On GUI, click '**CodeCoverage**' button. It will display code coverage report of all seeds generated by libFuzzer.

Note: All reports are generated in the libfuzzer_xxx(xxx means test case name) folder under 'OutputPath' you set in 3.2.1.2.

Example:

Step 1: Click '**Import**', choose 'TestPartition'.

Step 2: Click '**Method**', choose 'libFuzzer'.

Step 3: Click '**Seeds**', choose seeds prepared under
HBFA/UefiHostFuzzTestCasePkg/Seed/UDF/Raw/Partition

Step 4: Click '**Exec**', a new terminal will display:

```

INFO: Seed: 364996928
INFO: Loaded 1 modules (91 inline 8-bit counters): 91 [0x7a8330, 0x7a838b),
INFO: Loaded 1 PC tables (91 PCs): 91 [0x581bf8,0x5821a8),
INFO: 36 files found in /home/tiano/EAST/UefiHostTestCasePkg/Seed/UDF/Raw/
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 1048576 bytes
INFO: seed corpus: files: 36 min: 4b max: 1048576b total: 23140600b rss: 39Mb

=====
==4155==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 144 byte(s) in 1 object(s) allocated from:
#0 0x51ec5f (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x51ec5f)
#1 0x5610f4 (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x5610f4)
#2 0x5621bd (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x5621bd)
#3 0x55ef3d (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x55ef3d)
#4 0x430628 (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x430628)
#5 0x433894 (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x433894)
#6 0x435e13 (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x435e13)
#7 0x436732 (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x436732)
#8 0x42916b (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x42916b)

```

Figure 7. libFuzzer Output

Step 5: After libFuzzer stop running itself of running for a period of time and stop it by 'Ctrl C' in terminal.

Step 6: Click '**Report**' button, and summary report will display on GUI and a summary report web page (similar to 'Figure 3. AFL summary report') is generated. Click 'Failure Type Number', detailed info will display (similar to 'Figure 4. Debug Report for Crashes').

Step 7: On GUI, click '**CodeCoverage**' button, code coverage report (similar to 'Figure 6. Code Coverage Report') will display.

3.2.2.3 Run Test with Peach + Sanitizer

General Steps:

Step 1: Click '**Import**', choose test case.

Step 2: Click '**Method**', choose 'Sanitizer' and 'Peach'.

Step 3: Click '**Exec**', a new pop-up terminal will display.

Step 4: Stop it by 'Ctrl C' in terminal.

Step 5: After finishing running Peach + Sanitizer, Click '**Report**' button to check summary report , and click 'Failure Type Number' in summary report. User can get detailed info about failures.

Step 6: On GUI, click '**CodeCoverage**' button. It will display code coverage report of all seeds generated by peach + sanitizer.

Note: All reports are generated in the peach+sanitizer_xxx(xxx means test case name) folder under 'OutputPath' you set in 3.2.1.2.

Example:

Step 1: Click '**Import**', choose 'TestUsb'.

Step 2: Click '**Method**', choose 'Sanitizer' and 'Peach'.

Step 3: Click '**Exec**', a new pop-up terminal will display, user can see the running status as below:

```
[376,-,-] Performing iteration
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
==14388==LeakSanitizer has encountered a fatal error.
==14388==HINT: For debugging, try setting environment variable LSAN_OPTIONS=verbosity=1:log_threads=1
==14388==HINT: LeakSanitizer does not work under ptrace (strace, gdb, etc)
[Inferior 1 (process 14388) exited with code 01]
* Fuzzing: TPM2_PCR_EVENT_RESPONSE.DataElement_2.digests.DataElement_0
* Mutator: DataElementDuplicateMutator
* Fuzzing: TPM2_PCR_EVENT_RESPONSE.DataElement_2.digests
* Mutator: DataElementDuplicateMutator
* Fuzzing: TPM2_PCR_EVENT_RESPONSE.DataElement_2.count
* Mutator: DataElementSwapNearNodesMutator
* Fuzzing: TPM2_PCR_EVENT_RESPONSE.DataElement_1.paramSize
* Mutator: DataElementDuplicateMutator
* Fuzzing: TPM2_PCR_EVENT_RESPONSE.ParameterSize
* Mutator: DataElementDuplicateMutator
* Fuzzing: TPM2_PCR_EVENT_RESPONSE.DataElement_2.digests.digests
* Mutator: DataElementDuplicateMutator

[377,-,-] Performing iteration
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
==14395==LeakSanitizer has encountered a fatal error.
==14395==HINT: For debugging, try setting environment variable LSAN_OPTIONS=verbosity=1:log_threads=1
==14395==HINT: LeakSanitizer does not work under ptrace (strace, gdb, etc)
[Inferior 1 (process 14395) exited with code 01]
* Fuzzing: TPM2_PCR_EVENT_RESPONSE.DataElement_2
* Mutator: DataElementDuplicateMutator
* Fuzzing: TPM2_PCR_EVENT_RESPONSE.DataElement_1.paramSize
* Mutator: NumericalVarianceMutator
* Fuzzing: TPM2_PCR_EVENT_RESPONSE.ParameterSize
* Mutator: DataElementRemoveMutator

[378,-,-] Performing iteration
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
```

Figure 8. Peach+Sanitizer Running Status

And Sanitizer will also catch memory corruption bugs and output stack information:


```

==11743==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x602000000032 at pc 0x00000052c
2b8 bp 0x7fffffffce80 sp 0x7fffffffce78
READ of size 2 at 0x602000000032 thread T0
#0 0x52c2b7 (/home/tiano/work/SecurityTest/edk2/Build/UefiHostTestPkg/DEBUG_CLANG8/X64/TestUs
b+0x52c2b7)
#1 0x52982f (/home/tiano/work/SecurityTest/edk2/Build/UefiHostTestPkg/DEBUG_CLANG8/X64/TestUs
b+0x52982f)
#2 0x7ffff6ee582f (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)
#3 0x41a728 (/home/tiano/work/SecurityTest/edk2/Build/UefiHostTestPkg/DEBUG_CLANG8/X64/TestUs
b+0x41a728)

0x602000000032 is located 1 bytes to the right of 1-byte region [0x602000000030,0x602000000031)
allocated by thread T0 here:
#0 0x4e945f (/home/tiano/work/SecurityTest/edk2/Build/UefiHostTestPkg/DEBUG_CLANG8/X64/TestUs
b+0x4e945f)
#1 0x52d1e4 (/home/tiano/work/SecurityTest/edk2/Build/UefiHostTestPkg/DEBUG_CLANG8/X64/TestUs
b+0x52d1e4)
#2 0x52bc21 (/home/tiano/work/SecurityTest/edk2/Build/UefiHostTestPkg/DEBUG_CLANG8/X64/TestUs
b+0x52bc21)
#3 0x52c009 (/home/tiano/work/SecurityTest/edk2/Build/UefiHostTestPkg/DEBUG_CLANG8/X64/TestUs
b+0x52c009)

SUMMARY: AddressSanitizer: heap-buffer-overflow (/home/tiano/work/SecurityTest/edk2/Build/UefiHost
TestPkg/DEBUG_CLANG8/X64/TestUsb+0x52c2b7)
Shadow bytes around the buggy address:
 0x0c047fff7fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x0c047fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x0c047fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x0c047fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x0c047fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c047fff8000: fa fa 00 fa fa fa[01]fa fa fa fa fa fa fa fa fa
 0x0c047fff8010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x0c047fff8020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x0c047fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x0c047fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x0c047fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
Addressable: 00
Partially addressable: 01 02 03 04 05 06 07
Heap left redzone: fa
Freed heap region: fd
Stack left redzone: f1
Stack mid redzone: f2
Stack right redzone: f3
Stack after return: f5
Stack use after scope: f8
Global redzone: f9
Global init order: f6
Poisoned by user: f7
Container overflow: fc
Array cookie: ac
Intra object redzone: bb
ASan internal: fe
Left alloca redzone: ca
Right alloca redzone: cb
Shadow gap: cc
==11743==ABORTING

```

Figure 9. Peach Catch an Error

Step 4: After running for a period of time and stop it by 'Ctrl C' in terminal.

Step 5: Click '**Report**' button, and summary report will display on GUI and a summary report web page (similar to 'Figure 3. AFL summary report') is generated. Click 'Failure Type Number', detailed info will display(similar to 'Figure 4. Debug Report for Crashes').

Step 6: On GUI, click '**CodeCoverage**' button, code coverage report will display(similar to 'Figure 6. Code Coverage Report').

3.2.2.4 Generate Test Seed by KLEE

General Steps:

Step 1: Click '**Import**', choose test case.

Step 2: Click '**Method**', choose 'KLEE(STP)'.

Step 3: Click '**Exec**', a new pop-up terminal will display.

Step 4: Wait until KLEE finish running, or user can stop it by 'Ctrl C' in terminal.

Step 5: Click '**Seed**' on GUI, seeds generated by KLEE with suffix '.seed' will be displayed in a Buffer folder.

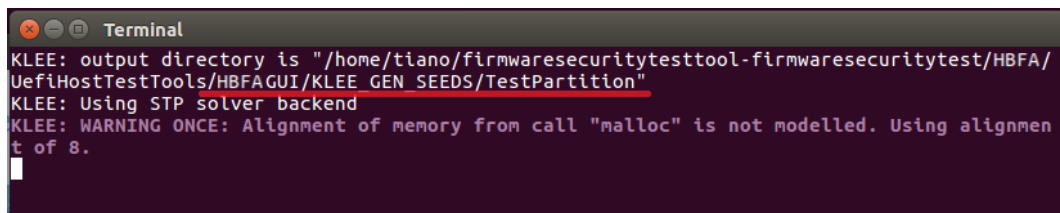
Note: User can run AFL and libFuzzer with seeds generated by KLEE.

Example:

Step 1: Click '**Import**', choose test case 'TestPartition'.

Step 2: Click '**Method**', choose 'KLEE(STP)'.

Step 3: Click '**Exec**', a new terminal will display, like '**Figure 12. KLEE is running**', in this picture, `HBFAGUI/KLEE_GEN_SEEDS/TestPartition` is new generated directory to store KLEE generated seeds.



```
Terminal
KLEE: output directory is "/home/tiano/firmwaresecuritytesttool-firmwaresecuritytest/HBFA/
UefiHostTestTools/HBFAGUI/KLEE_GEN_SEEDS/TestPartition"
KLEE: Using STP solver backend
KLEE: WARNING ONCE: Alignment of memory from call "malloc" is not modelled. Using alignmen
t of 8.
```

Figure 10. KLEE Running Status

Step 4: Wait until KLEE finish running, or stop it by 'Ctrl C' in terminal.

Step 5: Click '**Seed**' on GUI, it will change seeds with suffix '.ktest' to with suffix '.seed' automatically, path of seeds with suffix '.seed' is like below:

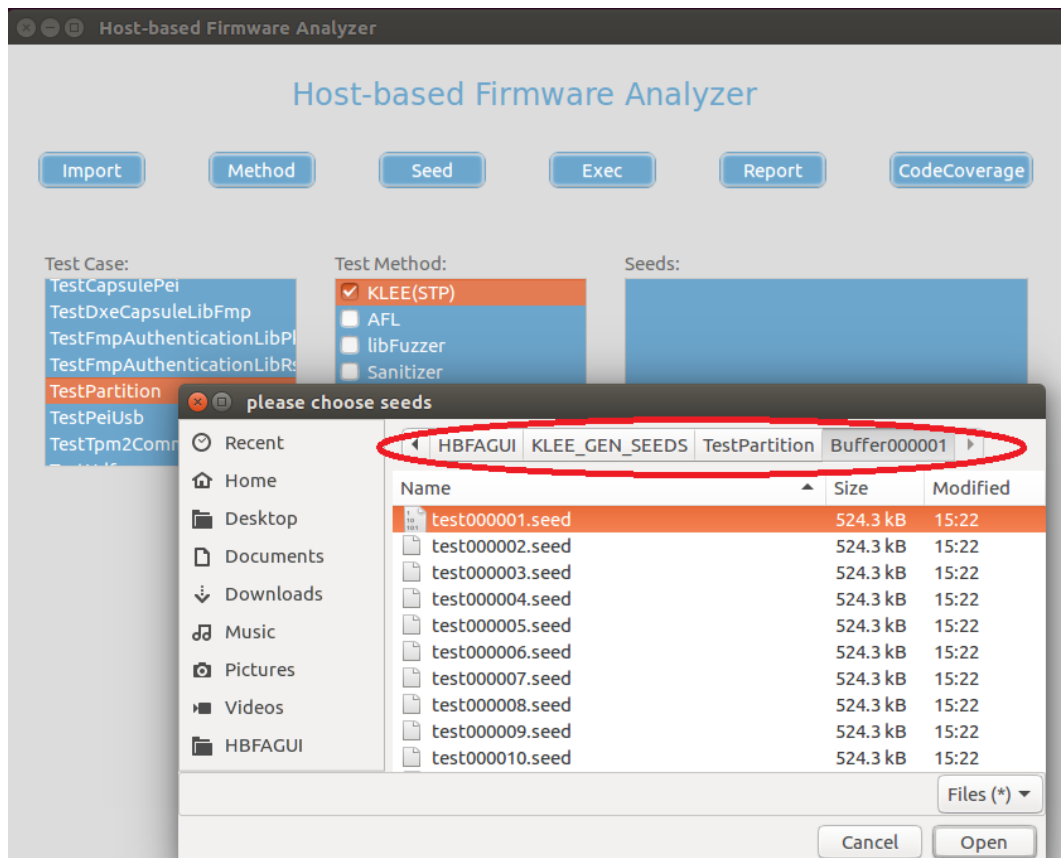


Figure 11. KLEE Seeds

Step 6: Run AFL with seeds with suffix '.seed' generated by KLEE, refer to 3.2.2.1.

3.3 Commandline Interface

3.3.1 Run Test Case

3.3.1.1 Run Test with AFL

This chapter explains how to run test with AFL. EDK II and Host-based Firmware Analyzer environment (Refer to chapter 2.2 and 3.1) should be set up ready before using this script.

General Steps:

Execute script RunAFL.py under `HBFA/UefiHostTestTools` package to build test module and run test with AFL.

Command:

```
python <SCRIPTPATH> -a <TARGETARCH> -b <BUILDTARGET> -m <MODULEFILE> -i
<INPUTSEED> -o <OUTPUT>
```

Usage: python RunAFL.py [options][argument]

Copyright (c) 2019, Intel Corporation. All rights reserved.

Options:

--version show program's version number and exit
-h, --help show this help message and exit
-a TARGETARCH, --arch=TARGETARCH
 ARCHS is one of list: IA32, X64, ARM or AARCH64, which
 overrides target.txt's TARGET_ARCH definition.
-b BUILDTARGET, --buildtarget=BUILDTARGET
 Using the TARGET to build the platform, overriding
 target.txt's TARGET definition.
-m MODULEFILE, --module=MODULEFILE
 Build the module specified by the INF file name
 argument.
-i INPUTSEED, --input=INPUTSEED
 Test input seed path.
-o OUTPUT, --output=OUTPUT
 Test output path for AFL.

Note: We suggest you to set OUTPUT path on a ramdisk, because a high number of read and write operations will harm harddisk.

Step 2: Generate Summary Report

Execute script ReportMain.py under `HBFA/UefiHostTestTools/Report` package to generate the summary report for AFL test.

Command:

```
python <SCRIPTPATH> -e <MODULEBIN> -i <AFLRESULTPATH> -r <REPORTPATH>
```

Usage: python ReportMain.py [options][argument]

Copyright (c) 2019, Intel Corporation. All rights reserved.

Options:

--version show program's version number and exit
-h, --help show this help message and exit
-e MODULEBIN, --execbinary=MODULEBIN
 Test module binary file name.
-i RESULTPATH, --input=RESULTPATH
 Test result path for test method.
-r REPORTPATH, --report=REPORTPATH
 Generated report path.
-t TESTMETHODS, --testmethods=TESTMETHODS
 Test method's name. Must be one of [afl, peach,
 libfuzzer]. Will be auto detected for default.
-s SLEEPTIME, --sleep=SLEEPTIME
 In run time mode, # of seconds to sleep between
 checking for new seed files

Step 3: Generate Code Coverage Report

Execute script GenCodeCoverage.py under `HBFA/UefiHostTestTools/Report` package to generate the code coverage report.

Command:


```
python <SCRIPTPATH> -e <MODULEBIN> -d <SEEDPATH> -r <REPORTPATH> -t
<TESTINIPATH>
```

Copyright (c) 2019, Intel Corporation. All rights reserved.

Options:

```
--version          show program's version number and exit
-h, --help         show this help message and exit
-e MODULEBIN, --execbinary=MODULEBIN
                   Test module binary file name.
-d SEEDPATH, --dir=SEEDPATH
                   Test output seed directory path.
-t TESTINIPATH, --testini=TESTINIPATH
                   Test ini files path for ErrorInjection, only for
                   ErrorInjection.
-r REPORTPATH, --report=REPORTPATH
                   Generated code coverage report path.
```

Example on Windows:

For AFL test, we use TestUdf.inf as an example:

Step 1:

Run following command to kick off the test:

```
python C:\Users\tiano\HBFA\UefiHostTestTools\RunAFL.py -m
UefiHostFuzzTestCasePkg\TestCase\MdeModulePkg\Universal\Disk\UdfDxe\TestU
df.inf -a IA32 -b DEBUG -i
C:\Users\tiano\HBFA\UefiHostFuzzTestCasePkg\Seed\UDF\Raw\FileSystem -o
C:\Users\tiano\AFL_SEED
```

After running the script, you will get AFL running status screen like below:

```
WinAFL 1.15 based on AFL 2.43b (TestUdf.exe)
+-- process timing -----+-- overall results -----+
|   run time : 0 days, 0 hrs, 0 min, 56 sec   | cycles done : 0      |
| last new path : none seen yet               | total paths : 18    |
| last uniq crash : none seen yet             | uniq crashes : 0    |
| last uniq hang : none seen yet              | uniq hangs : 0     |
+-- cycle progress -----+-- map coverage -----+
| now processing : 2 (11.11%)                 | map density : 0.35% / 1.87% |
| paths timed out : 0 (0.00%)                 | count coverage : 1.86 bits/tuple |
+-- stage progress -----+-- findings in depth -----+
| now trying : bitflip 1\1                   | favored paths : 13 (72.22%) |
| stage execs : 3990/6.29M (0.06%)            | new edges on : 16 (88.89%) |
| total execs : 5688                         | total crashes : 0 (0 unique) |
| exec speed : 150.9/sec                     | total tmouts : 0 (0 unique) |
+-- fuzzing strategy yields -----+-- path geometry -----+
| bit flips : 0/0, 0/0, 0/0                 | levels : 1          |
| byte flips : 0/0, 0/0, 0/0                | pending : 18        |
| arithmetics : 0/0, 0/0, 0/0               | pend fav : 13       |
| known ints : 0/0, 0/0, 0/0                | own finds : 0        |
| dictionary : 0/0, 0/0, 0/0                | imported : n/a       |
| havoc : 0/0, 0/0                          | stability : 99.92%   |
| trim : 25.00%/1521, n/a                   |                      |
+-----+-- [cpu: 0%] -----+
```

Figure 12. Windows AFL Running Status

In this window, you can get the summary information of the test status, you can get the meaning of each item by referring to https://github.com/nccgroup/TriforceAFL/blob/master/docs/status_screen.txt.

Step 2:

After stopping running AFL with 'Ctrl C', run following command to get the summary report:

```
python C:\Users\tiano\HBFA\UefiHostTestTools\Report\ReportMain.py -r
C:\Users\tiano\TestUdf -i C:\Users\tiano\AFL_SEED -e
C:\Users\tiano\Build\UefiHostFuzzTestCasePkg\DEBUG_CLANGWIN\IA32\TestUdf.
exe
```

Note: CLANGWIN binary need to be built before getting report.

Step 3:

Run following command to get the code coverage report:

```
python C:\Users\tiano\HBFA\UefiHostTestTools\Report\GenCodeCoverage.py -e
C:\Users\tiano\Build\UefiHostFuzzTestCasePkg\DEBUG_VS2015x86\IA32\TestUdf
.exe -d C:\Users\tiano\AFL_SEED\queue -r C:\Users\tiano\TestUdf
```

After running the script, you will get the code coverage report in folder `C:\Users\tiano\TestUdf\CodeCoverageReport` like below:

LCOV - code coverage report					
Current view: top level		Hit		Total	Coverage
Test: trace.lcov_info_final		Lines:	1170	1283	91.2 %
Date: 2018-10-20 13:00:01		Functions:	59	60	98.3 %
Directory	Line Coverage %	Functions %			
UdfDxe	91.2 %	98.3 %	1170 / 1283	59 / 60	
Generated by: LCOV version 1.12					

Figure 13. Code Coverage Report 1

Click directory name "UdfDxe" in html report, then you can get the detail code coverage info like below:

LCOV - code coverage report					
Current view: top level - UdfDxe		Hit		Total	Coverage
Test: trace.lcov_info_final		Lines:	1170	1283	91.2 %
Date: 2018-10-20 13:00:01		Functions:	59	60	98.3 %
Filename	Line Coverage %	Functions %			
File.c	96.2 %	100.0 %	256 / 266	11 / 11	
File.h	95.1 %	100.0 %	77 / 81	4 / 4	
FileOperations.c	92.5 %	100.0 %	637 / 689	33 / 33	
TestUdf.c	75.0 %	100.0 %	18 / 24	2 / 2	
TestUdf.h	81.6 %	90.0 %	182 / 223	9 / 10	
Generated by: LCOV version 1.12					

Figure 14. Code Coverage Report 2

Example on Linux:

For AFL test, we use TestUdf.inf as an example:

Step 1:

Run following command to kick off the test:

```
python /home/tiano/HBFA/UefiHostTestTools/RunAFL.py -m
UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Universal/Disk/UdfDxe/TestU
df.inf -a IA32 -b DEBUG -i
/home/tiano/HBFA/UefiHostFuzzTestCasePkg/Seed/UDF/Raw/FileSystem -o
/dev/shm/TestUdf
```

After running the script, you will get AFL running status screen like below:

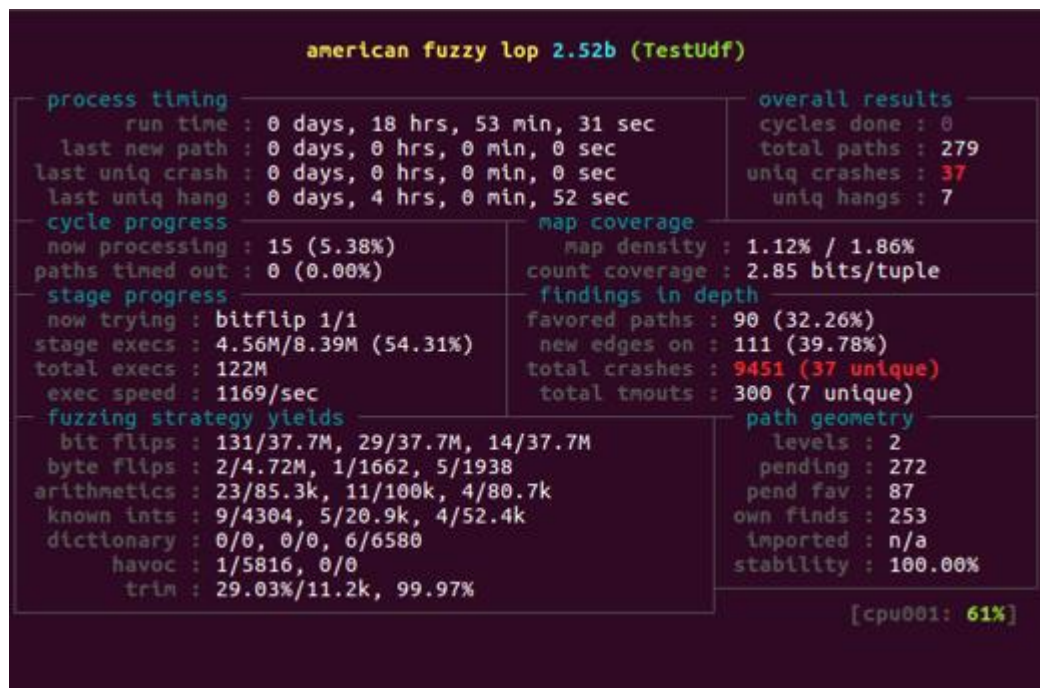


Figure 15. AFL Running Status

In this window, you can get the summary information of the test status, you can get the meaning of each item by referring to https://github.com/nccgroup/TriforceAFL/blob/master/docs/status_screen.txt.

Step 2:

After stopping running AFL with 'Ctrl C', run following command to get the summary report:

```
python /home/tiano/HBFA/UefiHostTestTools/Report/ReportMain.py -r
/home/tiano/AFL/TestUdf -i /dev/shm/TestUdf -e
/home/tiano/edk2/Build/UefiHostFuzzTestCasePkg/DEBUG_GCC5/IA32/TestUdf
```

After running the script you can get the summary report in folder `/home/tiano/AFL/TestUdf` like below:

Host-based Firmware Analyzer TestUdf Summary Report

HighLight: This is Host-based Firmware Analyzer TestUdf Summary Report offered the crashes and hangs number.
Date: 2018-10-30 13:01:35

ItemName	ItemValue
Generated Seeds Total Number:	279
Generated Crashes Number:	32
Generated Hangs Number:	7
TestCases Execution Time:	0 days,18 hrs,53 min,31 sec

Generated by: Host-based Firmware Analyzer

Figure 16. AFL Summary Report

Click the "ItemValue" of "Generated Crashes Number", then you can get the DEBUG report with crashes' detail information like below:

Host-based Firmware Analyzer Debug Report

HighLight: This is Host-based Firmware Analyzer debug report that offers the crashes' detail information.
Date: 2018-10-20 12:57:37

Crash Number: 37

Seed Name	File Name	Line Number	Error Message	Stack
id:000000	FilesystemOperations.c	2533	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information
id:000001	FilesystemOperations.c	2518	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information
id:000002	libc.so.6	-	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information
id:000003	_kernel_vsyscall ()	-	Program received signal SIGABRT, Aborted.	+ Detail Stack Information
id:000004	_kernel_vsyscall ()	-	Program received signal SIGABRT, Aborted.	+ Detail Stack Information
id:000005	_kernel_vsyscall ()	-	Program received signal SIGABRT, Aborted.	+ Detail Stack Information
id:000006	libc.so.6	-	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information
id:000007	libc.so.6	-	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information
id:000008	FilesystemOperations.c	2533	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information
id:000009	libc.so.6	-	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information
id:000010	FilesystemOperations.c	2533	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information
id:000011	FilesystemOperations.c	2533	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information
id:000012	libc.so.6	-	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information
id:000013	libc.so.6	-	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information
id:000014	_kernel_vsyscall ()	-	Program received signal SIGABRT, Aborted.	+ Detail Stack Information
id:000015	_kernel_vsyscall ()	-	Program received signal SIGABRT, Aborted.	+ Detail Stack Information
id:000016	FilesystemOperations.c	2533	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information
id:000017	_kernel_vsyscall ()	-	Program received signal SIGABRT, Aborted.	+ Detail Stack Information
id:000018	_kernel_vsyscall ()	-	Program received signal SIGABRT, Aborted.	+ Detail Stack Information
id:000019	_kernel_vsyscall ()	-	Program received signal SIGABRT, Aborted.	+ Detail Stack Information
id:000020	_kernel_vsyscall ()	-	Program received signal SIGABRT, Aborted.	+ Detail Stack Information
id:000021	FilesystemOperations.c	2533	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information
id:000022	FilesystemOperations.c	2533	Program received signal SIGSEGV, Segmentation Fault.	+ Detail Stack Information

Figure 17. Debug Report for Crashes

Click the "ItemValue" of "Generated Hangs Number", then you can get the DEBUG report with hangs' detail information like below:

Host-based Firmware Analyzer Debug Report

HighLight: This is Host-based Firmware Analyzer debug report that offers the hangs' detail information.
Date: 2018-10-20 12:57:51

Hang Number: 7

Seed Name	File Name	Line Number	Error Message	Stack
id:000000	FilesystemOperations.c	1312	Program received signal SIGTERM, Terminated.	+ Detail Stack Information
id:000001	FilesystemOperations.c	1152	Program received signal SIGTERM, Terminated.	+ Detail Stack Information
id:000002	libc.so.6	-	Program received signal SIGTERM, Terminated.	+ Detail Stack Information
id:000003	FilesystemOperations.c	307	Program received signal SIGTERM, Terminated.	+ Detail Stack Information
id:000004	FilesystemOperations.c	1194	Program received signal SIGTERM, Terminated.	+ Detail Stack Information
id:000005	FilesystemOperations.c	775	Program received signal SIGTERM, Terminated.	+ Detail Stack Information
id:000006	FilesystemOperations.c	628	Program received signal SIGTERM, Terminated.	+ Detail Stack Information

Generated by: Host-based Firmware Analyzer

Figure 18. Debug Report for Hangs

Step 3:

Run following command to get the code coverage report:

```
python /home/tiano/HBFA/UefiHostTestTools/Report/GenCodeCoverage.py -e
/home/tiano/edk2/Build/UefiHostFuzzTestCasePkg/DEBUG_GCC5/IA32/TestUdf -d
/dev/shm/TestUdf/queue -r /home/tiano/AFL/TestUdf
```

After running the script, you will get the code coverage report in folder
 /home/tiano/AFL/TestUdf/CodeCoverageReport like below:

LCOV - code coverage report

Current view: top level		Hit	Total	Coverage
Test: trace.lcov_info_final		Lines: 1170	1283	91.2 %
Date: 2018-10-20 13:00:01		Functions: 59	60	98.3 %

Directory	Line Coverage %	Functions %
UdfDxe	91.2 % 1170 / 1283	98.3 % 59 / 60

Generated by: LCOV version 1.12

Figure 19. Code Coverage Report 1

Click directory name "UdfDxe" in html report, then you can get the detail code coverage info like below:

LCOV - code coverage report

Current view: top level - UdfDxe		Hit	Total	Coverage
Test: trace.lcov_info_final		Lines: 1170	1283	91.2 %
Date: 2018-10-20 13:00:01		Functions: 59	60	98.3 %

Filename	Line Coverage %	Functions %
File.c	96.2 % 256 / 266	100.0 % 11 / 11
File.c	95.1 % 77 / 81	100.0 % 4 / 4
FileSystemOperations.c	92.5 % 637 / 689	100.0 % 33 / 33
TestFile.c	75.0 % 18 / 24	100.0 % 2 / 2
TestUdf.c	81.6 % 182 / 223	90.0 % 9 / 10

Generated by: LCOV version 1.12

Figure 20. Code Coverage Report 2

3.3.1.2 Run Test with libFuzzer

This chapter explains how to run test with libFuzzer. EDK II and Host-based Firmware Analyzer environment (Refer to chapter 2.2 and 3.1) should be set up ready before using this script.

General Steps:

Step 1: Run Test Case

Execute script RunLibFuzzer.py under HBFA/UefiHostTestTools package to build test module and run test with libFuzzer.

Command:

```
python <SCRIPTPATH> -a <TARGETARCH> -b <BUILDTARGET> -m <MODULEFILE> -i
<INPUTSEED>
```

Usage: python RunLibFuzzer.py [options][argument]

Copyright (c) 2019, Intel Corporation. All rights reserved.

Options:

--version show program's version number and exit

```

-h, --help            show this help message and exit
-a TARGETARCH, --arch=TARGETARCH
                        ARCHS is one of list: IA32, X64, ARM or AARCH64, which
                        overrides target.txt's TARGET_ARCH definition.
-b BUILDTARGET, --buildtarget=BUILDTARGET
                        Using the TARGET to build the platform, overriding
                        target.txt's TARGET definition.
-m MODULEFILE, --module=MODULEFILE
                        Build the module specified by the INF file name
                        argument.
-i INPUTSEED, --input=INPUTSEED
                        Test input seed path.
-o OUTPUT, --output=OUTPUT
                        Test output path for LibFuzzer.

```

Note: Currently, we only support "-a X64" for libFuzzer, if you use other arch, module will fail to be built.

Step 2: Generate Summary Report

Execute script ReportMain.py under HBFA/UefiHostTestTools/Report package to generate the summary report for AFL test.

Command:

```
python <SCRIPTPATH> -e <MODULEBIN> -i <LIBFUZZERRESULTPATH> -r
<REPORTPATH>
```

Step 3: Generate Code Coverage Report

Execute script GenCodeCoverage.py under HBFA/UefiHostTestTools/Report package to generate the code coverage report.

Command:

```
python <SCRIPTPATH> -e <MODULEBIN> -d <INPUTSEED> -r <REPORTPATH>
```

Example on Windows:

For libFuzzer test, we use TestPartition.inf as an example:

Step 1:

Run following command to kick off the test:

```
python C:\Users\tiano\HBFA\UefiHostTestTools\RunLibFuzzer.py -m
UefiHostFuzzTestCasePkg\TestCase\MdeModulePkg\Universal\Disk\PartitionDxe
\TestPartition.inf -a X64 -b DEBUG -i
C:\Users\tiano\HBFA\UefiHostFuzzTestCasePkg\Seed\UDF\Raw\Partition -o
C:\Users\tiano\Failures
```

Step 2:

After libFuzzer stop running itself of running for a period of time and stop it by 'Ctrl C' in terminal, run following command to get the summary report

```
python C:\Users\tiano\HBFA\UefiHostTestTools\Report\ReportMain.py -r
C:\Users\tiano\TestPartition -i C:\Users\tiano\Failures -e
C:\Users\tiano\Build\UefiHostFuzzTestCasePkg\DEBUG_CLANGWIN\X64\TestParti
tion.exe
```


Note: CLANGWIN binary need to be built before getting report.

Step 3:

Run following command to get the code coverage report:

```
python C:\Users\tiano\HBFA\UefiHostTestTools\Report\GenCodeCoverage.py -e
C:\Users\tiano\Build\UefiHostFuzzTestCasePkg\DEBUG_VS2015x86\X64\TestPart
ition.exe -d
C:\Users\tiano\HBFA\UefiHostFuzzTestCasePkg\Seed\UDF\Raw\Partition -r
C:\Users\tiano\TestPartition
```

Example on Linux:

For libFuzzer test, we use TestPartition.inf as an example:

Step 1:

Run following command to kick off the test:

```
python /home/tiano/HBFA/UefiHostTestTools/RunLibFuzzer.py -m
UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Universal/Disk/PartitionDxe
/TestPartition.inf -i
/home/tiano/HBFA/UefiHostFuzzTestCasePkg/Seed/UDF/Raw/Partition -a X64 -b
DEBUG
```

After running the script, you will get libFuzzer running status screen like below:

```
INFO: Seed: 364996928
INFO: Loaded 1 modules (91 inline 8-bit counters): 91 [0x7a8330, 0x7a838b),
INFO: Loaded 1 PC tables (91 PCs): 91 [0x581bf8,0x5821a8),
INFO: 36 files found in /home/tiano/EAST/UefiHostTestCasePkg/Seed/UDF/Raw/
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 1048576 byt
es
INFO: seed corpus: files: 36 min: 4b max: 1048576b total: 23140600b rss: 39Mb

=====
==4155==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 144 byte(s) in 1 object(s) allocated from:
#0 0x51ec5f (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x51ec5f)
#1 0x5610f4 (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x5610f4)
#2 0x5621bd (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x5621bd)
#3 0x55ef3d (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x55ef3d)
#4 0x430628 (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x430628)
#5 0x433894 (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x433894)
#6 0x435e13 (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x435e13)
#7 0x436732 (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x436732)
#8 0x42916b (/home/tiano/edk2/Build/UefiHostTestCasePkg/DEBUG_LIBFUZZER/X64/TestParti
tion+0x42916b)
```

Figure 21. libFuzzer Output

Step 2:

After libFuzzer stop running itself of running for a period of time and stop it by 'Ctrl C' in terminal, run following command to get the summary report:

```
python /home/tiano/HBFA/UefiHostTestTools/Report/ReportMain.py -r
/home/tiano/libFuzzer/TestPartition -i /dev/shm/Failures -e
/home/tiano/edk2/Build/UefiHostFuzzTestCasePkg/DEBUG_CLANG8/X64/TestParti
tion
```

Note: CLANG8 binary need to be built before getting report.

Step 3:

Run following command to get the code coverage report:

```
python /home/tiano/HBFA/UefiHostTestTools/Report/GenCodeCoverage.py -e
/home/tiano/edk2/Build/UefiHostFuzzTestCasePkg/DEBUG_GCC5/X64/TestPartiti
on -d /home/tiano/HBFA/UefiHostFuzzTestCasePkg/Seed/UDF/Raw/Partition -r
/home/tiano/libFuzzer/TestPartition
```

3.3.1.3 Run Test with Peach + Sanitizer

This chapter explains how to run test with Peach + Sanitizer. EDK II and Host-based Firmware Analyzer environment (Refer to chapter 2.2 and 3.1) should be set up ready before using this script.

General Steps:

Step 1: Run Test Case

Execute script RunPeachSanitizer.py under HBFA/UefiHostTestTools package to build test module and run test with Peach + Sanitizer.

Command:

```
python <SCRIPTPATH> -a <TARGETARCH> -b <BUILDTARGET> -m <MODULEFILE> -o
<OUTPUT> --enablesanitizer
```

Usage: python RunPeach.py [options][argument]

Copyright (c) 2019, Intel Corporation. All rights reserved.

Options:

```
--version      show program's version number and exit
-h, --help     show this help message and exit
-a TARGETARCH, --arch=TARGETARCH
                ARCHS is one of list: IA32, X64, ARM or AARCH64, which
                overrides target.txt's TARGET_ARCH definition.
-b BUILDTARGET, --buildtarget=BUILDTARGET
                Using the TARGET to build the platform, overriding
                target.txt's TARGET definition.
-m MODULEFILE, --module=MODULEFILE
                Build the module specified by the INF file name
                argument.
-o OUTPUT, --output=OUTPUT
                Test output path for PEACH.
--enablesanitizer
                Using --enablesanitizer to enable sanitizer for peach
                test.
```

Note: Currently, we only support "-a X64" for Peach + Sanitizer, if you use other arch, module will fail to be built.

Step 2: Generate Summary Report

Execute script ReportMain.py under HBFA/UefiHostTestTools/Report package to generate the summary report for AFL test.

Command:

```
python <SCRIPTPATH> -e <MODULEBIN> -i <PEACHSANITIZERRESULTPATH> -r <REPORTPATH>
```

Step 3: Generate Code Coverage Report

Execute script GenCodeCoverage.py under HBFA/UefiHostTestTools/Report package to generate the code coverage report.

Command:

```
python <SCRIPTPATH> -e <MODULEBIN> -r <REPORTPATH>
```

Example on Windows:

For Peach + Sanitizer test, we use TestUsb.inf as an example:

Step 1:

Run following command to kick off the test:

```
python C:\Users\tiano\HBFA\UefiHostTestTools\RunPeachSanitizer.py -m UefiHostFuzzTestCasePkg\TestCase\MdeModulePkg\Bus\Usb\UsbBusDxe\TestUsb.inf -a X64 -b DEBUG --enablesanitizer -o C:\Users\tiano\Failures
```

Step 2:

After libFuzzer stop running itself of running for a period of time and stop it by 'Ctrl C' in terminal, run following command to get the summary report:

```
python C:\Users\tiano\HBFA\UefiHostTestTools\Report\ReportMain.py -r C:\Users\tiano\Peach\TestUsb -i C:\Users\tiano\Failures -e C:\Users\tiano\Build\UefiHostFuzzTestCasePkg\DEBUG_CLANGWIN\X64\TestUsb.exe
```

Example on Linux:

For Peach + Sanitizer test, we use TestUsb.inf as an example:

Step 1:

Run following command to kick off the test:

```
python /home/tiano/HBFA/UefiHostTestTools/RunPeachSanitizer.py -m UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Bus/Usb/UsbBusDxe/TestUsb.inf -a X64 -b DEBUG --enablesanitizer -o /dev/shm/Failures
```

After running the script, you will get Peach+Sanitizer running status screen like below:

```

[*] Fuzzing: FULL_USB_CONFIG_DESCRIPTOR.DataElement_3
[*] Mutator: DataElementSwapNearNodesMutator

[634,-,-] Performing iteration
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
==2969==LeakSanitizer has encountered a fatal error.
==2969==HINT: For debugging, try setting environment variable LSAN_OPTIONS=verbosity=1:log
_threads=1
==2969==HINT: LeakSanitizer does not work under ptrace (strace, gdb, etc)
[Inferior 1 (process 2969) exited with code 01]
[*] Fuzzing: FULL_USB_CONFIG_DESCRIPTOR.DataElement_1.TotalLength
[*] Mutator: NumericalEdgeCaseMutator
[*] Fuzzing: FULL_USB_CONFIG_DESCRIPTOR.DataElement_3.Length
[*] Mutator: DataElementSwapNearNodesMutator
[*] Fuzzing: FULL_USB_CONFIG_DESCRIPTOR.DataElement_3
[*] Mutator: DataElementRemoveMutator

[635,-,-] Performing iteration
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
==2976==LeakSanitizer has encountered a fatal error.
==2976==HINT: For debugging, try setting environment variable LSAN_OPTIONS=verbosity=1:log
_threads=1
==2976==HINT: LeakSanitizer does not work under ptrace (strace, gdb, etc)
[Inferior 1 (process 2976) exited with code 01]
[*] Fuzzing: FULL_USB_CONFIG_DESCRIPTOR.DataElement_3.NumDescriptors
[*] Mutator: NumericalEdgeCaseMutator
[*] Fuzzing: FULL_USB_CONFIG_DESCRIPTOR.DataElement_2.Length
[*] Mutator: NumericalEdgeCaseMutator

```

Figure 22. Peach+Sanitizer Running Status

Step 2:

After libFuzzer stop running itself of running for a period of time and stop it by 'Ctrl C' in terminal, run following command to get the summary report:

```
python /home/tiano/HBFA/UefiHostTestTools/Report/ReportMain.py -r
/home/tiano/Peach/TestUsb -i /dev/shm/Failures -e
/home/tiano/edk2/Build/UefiHostFuzzTestCasePkg/DEBUG_CLANG8/X64/TestUsb
```

Step 3:

Run following command to get the code coverage report:

```
python /home/tiano/HBFA/UefiHostTestTools/Report/GenCodeCoverage.py -e
/home/tiano/edk2/Build/UefiHostFuzzTestCasePkg/DEBUG_CLANG8/X64/TestUsb -r
/home/tiano/Peach/TestUsb
```

3.3.1.4 Generate Test Seed by KLEE

This chapter explains how to generate test seed by KLEE. EDK II and Host-based Firmware Analyzer environment (Refer to chapter 2.2 and 3.1) should be set up ready before using this script.

General Steps:

Step 1: Run Test Case

Execute script RunKLEE.py under HBFA/UefiHostTestTools package to build test module and run test with KLEE.

Command:

```
python <SCRIPTPATH> -a <TARGETARCH> -b <BUILDTARGET> -m <MODULEFILE> -o
<OUTPUT>
```

```
Usage: python RunKLEE.py [options][argument]
```

Copyright (c) 2019, Intel Corporation. All rights reserved.

Options:

```
--version          show program's version number and exit
-h, --help         show this help message and exit
-a TARGETARCH, --arch=TARGETARCH
                   ARCHS is one of list: IA32, X64, ARM or AARCH64, which
                   overrides target.txt's TARGET_ARCH definition.
-b BUILDTARGET, --buildtarget=BUILDTARGET
                   Using the TARGET to build the platform, overriding
                   target.txt's TARGET definition.
-m MODULEFILE, --module=MODULEFILE
                   Build the module specified by the INF file name
                   argument.
-o OUTPUT, --output=OUTPUT
                   Test output path for Klee.
```

Step 2: Transfer Seed

Transfer “.ktest” files generated by KLEE to “.seed” files. Remove KLEE header in “.ktest” file and generate pure “.seed” file.

Command:

```
python TransferKtestToSeed.py <KtestFolder>
```

```
Usage: python TransferKtestToSeed.py [Argument]
```

Remove header of ktest format file, and save the new binary file as .seed file.

Argument:

```
<KtestFile>          the path of .ktest file.
<KtestFile1> <KtestFile2> ... the paths of .ktest files.
<KtestFolder>        the path of folder contains .ktest file.
<KtestFolder1> <KtestFolder2> ... the paths of folders contain .ktest file.
```

Step 3: Generate Code Coverage Report

Refer to step 3 of chapter 3.3.1.1.

Example:

For KLEE test, we use TestPartition.inf as an example:

Step 1:

Run following command to kick off the test:

```
python /home/tiano/HBFA/UefiHostTestTools/RunKLEE.py -a IA32 -b DEBUG -m
UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Universal/Disk/PartitionDxe
/TestPartition.inf -o /home/tiano/KLEE/seed
```

After running the script, you will get KLEE running status screen like below:

```
KLEE: output directory is "/home/tiano/KLEE/seed"
KLEE: Using STP solver backend
KLEE: WARNING ONCE: Alignment of memory from call "malloc" is not modelled. Using alignmen
t of 8.
```

Figure 23. KLEE Running Status

Check generated seeds after KLEE test is over, all the seeds are named end with ".kest" in folder `/home/tiano/KLEE/seed` like below:

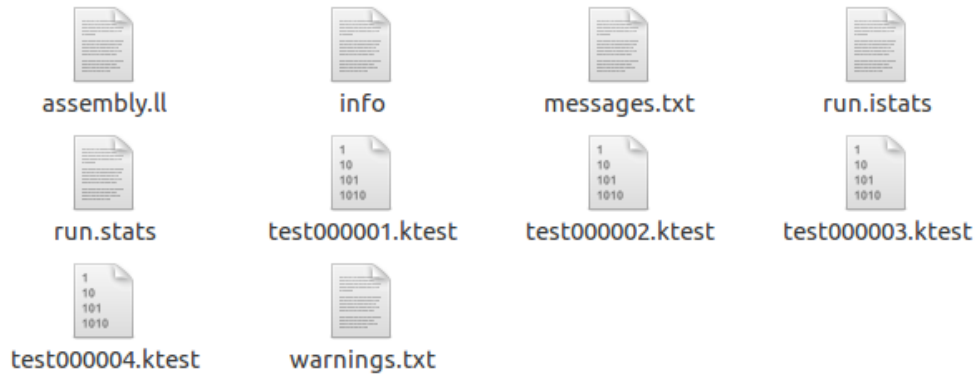


Figure 24. Generated ".ktest" Files

Step 2:

Run following command to transfer seed files:

```
python /home/tiano/HBFA/UefiHostTestTools/Script/TransferKtestToSeed.py  
/home/tiano/KLEE/seed
```

After running the script, you will find seeds files named end with ".seed" in folder `/home/tiano/KLEE/seed/Buffer0000001` like below:

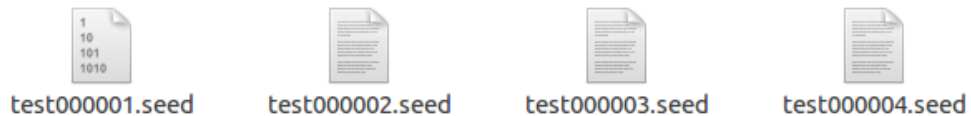


Figure 25. Generated ".seed" Files

Step 3:

Run following command to build test module by GCC5:

```
build -p UefiHostFuzzTestCasePkg/UefiHostFuzzTestCasePkg.dsc -m  
UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Universal/Disk/PartitionDxe  
/TestPartition.inf -a IA32 -t GCC5 --conf  
/home/tiano/HBFA/UefiHostTestPkg/Conf/Other
```

Run following command to get the code coverage report:

```
python /home/tiano/HBFA/UefiHostTestTools/Report/GenCodeCoverage.py -e  
/home/tiano/edk2/Build/UefiHostFuzzTestCasePkg/DEBUG_GCC5/IA32/TestPartit  
ion -d /home/tiano/KLEE/seed -r /home/tiano/KLEE/TestPartition
```

After running the script, you will get the code coverage report in folder `/home/tiano/KLEE/TestPartition/CodeCoverageReport` like below:

LCOV - code coverage report					
Current view: top level		Hit		Total	Coverage
Test: coverage.info		Lines:	97	373	26.0 %
Date: 2018-10-21 13:48:50		Functions:	8	15	53.3 %
Directory	Line Coverage			Functions	
/home/tiano/EAST/UefiHostTestCasePkg/TestStub/DiskStubLib	<div><div></div></div>	27.6 %	53 / 192	42.9 %	3 / 7
PartitionDxe	<div><div></div></div>	24.3 %	44 / 181	62.5 %	5 / 8

Generated by: LCOV version 1.12

Figure 26. KLEE Code Coverage Report

3.3.1.5 Run Test with ErrorInjection

This chapter explains how to run test with ErrorInjection. In this chapter, an example is given to show that ErrorInjection improves code coverage to 100%. EDK II and Host-based Firmware Analyzer environment (Refer to chapter 2.2 and 3.1) should be set up ready before using this script.

General Steps:

Step 1: Build Test Case with UefiInstrumentTestPkg

Command:

```
build -p UefiHostFuzzTestCasePkg\UefiHostFuzzTestCasePkg.dsc -m
<MODULEFILE> -a IA32 -t XXX -D TEST_WITH_INSTRUMENT
```

Step 2: Create Error Injection Profile

Create 'test.ini' file like below example:

```
[AllocateZeroPool]
  CallErrorCount = 1
  ReturnValue = 0
```

AllocateZeroPool is a function name.

CallErrorCount means which call returns error. CallErrorCount = 0 means disable.

ReturnValue means when error happens, which value is returned.

Step3: Run Test Case and Generate Code Coverage Report

Execute script GenCodeCoverage.py under HBFA/UefiHostTestTools package to generate code coverage report.

Command:

```
python <SCRIPTPATH> -e <MODULEBIN> -d <SEEDPATH> -t <TESTINIPATH> -r
<REPORTPATH>
```

Usage: python GenCodeCoverage.py [options][argument]

Copyright (c) 2018, Intel Corporation. All rights reserved.

Options:

```
--version      show program's version number and exit
-h, --help     show this help message and exit
```

```
-e MODULEBIN, --execbinary=MODULEBIN
    Test module binary file name.
-d SEEDPATH, --dir=SEEDPATH
    Test output seed directory path.
-t TESTINIPATH, --testini=TESTINIPATH
    Test ini files path for ErrorInjection, only for
    ErrorInjection.
-r REPORTPATH, --report=REPORTPATH
    Generated code coverage report path.
```

Example:

For ErrorInjection test, we use TestPartition.inf as an example. For this example we will compare the code coverage data between without using ErrorInjection method and using ErrorInjection method.

- **Without using ErrorInjection**

Step 1:

Run following command to build TestPartition:

```
build -p UefiHostFuzzTestCasePkg/UefiHostFuzzTestCasePkg.dsc -m
UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Universal/Disk/Parti
tionDxe/TestPartition.inf -a IA32 -t GCC5 --conf
/home/tiano/HBFA/UefiHostTestPkg/Conf/Other
```

Step2:

Seeds in

/home/tiano/HBFA/UefiHostFuzzTestCasePkg/Seed/UDF/Raw/Partition
already have high coverage, only some error handling paths are not covered.

Run following command to the generate code coverage without running ErrorInjection:

```
python
/home/tiano/HBFA/UefiHostTestTools/Report/GenCodeCoverage.py -e
/home/tiano/edk2/Build/UefiHostFuzzTestCasePkg/DEBUG_GCC5/IA32/Tes
tPartition -d
/home/tiano/HBFA/UefiHostFuzzTestCasePkg/Seed/UDF/Raw/Partition -r
/home/tiano/AFL/TestPartition
```

After running the script, you will get code coverage report in folder
/home/tiano/AFL/TestPartition/CodeCoverageReport, you can get Udf.c
code coverage is 93.2% and find some error handling paths cannot be
covered.

LCOV - code coverage report

Current view: top level - PartitionDxe		Hit	Total	Coverage
Test: coverage.info		Lines: 164	181	90.6 %
Date: 2018-10-21 14:57:38		Functions: 8	8	100.0 %
Filename	Line Coverage	Functions		
TestPartition.c	78.8 % 26 / 33	100.0 % 3 / 3		
Udf.c	93.2 % 138 / 148	100.0 % 5 / 5		

Generated by: LCOV version 1.12

```

107 : //
108 : Status = DiskIo->ReadDisk (
109 :     DiskIo,
110 :     41 : BlockIo->Media->MediaId,
111 :     MultU64x32 (256, BlockSize),
112 :     sizeof (*AnchorPoint),
113 :     AnchorPoint
114 : );
115 : if (EFI_ERROR (Status)) {
116 :     return Status;
117 : }

```

Figure 27. Code Coverage without Running ErrorInjection

- **Using ErrorInjection**

Step 1:

Run following command to build TestPartition with UefiInstrumentTestPkg:

```

build -p UefiHostFuzzTestCasePkg/UefiHostFuzzTestCasePkg.dsc -m
UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Universal/Disk/Parti
tionDxe/TestPartition.inf -a IA32 -t GCC5 -D TEST_WITH_INSTRUMENT
--conf /home/tiano/HBFA/UefiHostTestPkg/Conf/Other

```

Step 2:

Run following command to create all ErrorInjection Profile for TestPartition:

```

python
/home/tiano/HBFA/UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Uni
versal/Disk/PartitionDxe/InstrumentHookLibTestPartition/CreateErro
rInjectionProfile.py -c 20

```

Step 3:

Run following command to generate the code coverage without running ErrorInjection:

```

python
/home/tiano/HBFA/UefiHostTestTools/Report/GenCodeCoverage.py -e
/home/tiano/edk2/Build/UefiHostFuzzTestCasePkg/DEBUG_GCC5/IA32/Tes
tPartition -d
/home/tiano/HBFA/UefiHostFuzzTestCasePkg/Seed/UDF/Raw/Partition -r
/home/tiano/AFL/TestPartition -t
/home/tiano/HBFA/UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Uni
versal/Disk/PartitionDxe/InstrumentHookLibTestPartition/Case

```

After running the script, you will get code coverage report in folder `/home/tiano/AFL/TestPartition/CodeCoverageReport`, you can get Udf.c code coverage is 100.0% and find the error handling paths are covered.

LCOV - code coverage report

Current view: top level - PartitionDxe		Hit		Total	Coverage
Test: coverage.info		Lines:	188	199	94.5 %
Date: 2018-10-21 15:30:33		Functions:	9	9	100.0 %
Filename	Line Coverage ↕	Functions ↕			
TestPartition.c	<div><div></div></div> 78.4 % 40 / 51	100.0 %	4 / 4		
Udf.c	<div><div></div></div> 100.0 % 148 / 148	100.0 %	5 / 5		

Generated by: [LCOV version 1.12](#)

```

107 : //
108 :
109 : 1192 : Status = DiskIo->ReadDisk (
110 : : DiskIo,
111 : : 596 : BlockIo->Media->MediaId,
112 : : : MultU64x32 (256, BlockSize),
113 : : : sizeof (*AnchorPoint),
114 : : : AnchorPoint
115 : : );
116 :
117 : 596 : if (EFI_ERROR (Status)) {
118 : 50 : return Status;
119 : }

```

Figure 28. Code Coverage after Running ErrorInjection

Appendix A

List of All Examples

This chapter provides the list of test example in this User Guide doc.

AFL:

Case Name	Case Path
TestUdf	HBFA/UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Universal/Disk/UdfDxe

KLEE:

Case Name	Case Path
TestPartition	HBFA/UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Universal/Disk/PartitionDxe

Peach+Sanitizer:

Case Name	Case Path
TestUsb	HBFA/UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Bus/Usb/UsbBusDxe

libFuzzer:

Case Name	Case Path
TestPartition	HBFA/UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Universal/Disk/PartitionDxe

Error Injection:

Case Name	Case Path
TestPartition	HBFA/UefiHostFuzzTestCasePkg/TestCase/MdeModulePkg/Universal/Disk/PartitionDxe