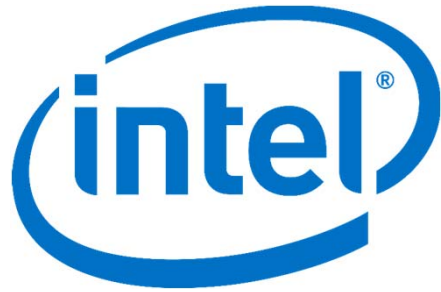


presented by



Hardening Firmware Components with Host-based Analysis

Spring 2019 UEFI Plugfest

April 8-12, 2019

Presented by Brian Richardson (Intel Corporation)

Agenda



- Today's Reality for Firmware Developers
- Examine Tools for Driver & Application Developers
- Host-based Firmware Analyzer
- Call to Action

Today's Reality for Firmware Developers



- Platform firmware is an essential component in software root-of-trust.
- Platform firmware is a low-level component, so potential security risks may not be apparent to users or developers.
- Most firmware validation is done via integration testing, which emphasizes functionality and stability.
- Integration testing is not ideal for detecting potential vulnerabilities in new firmware drivers.

Common Tools for Firmware Security Testing



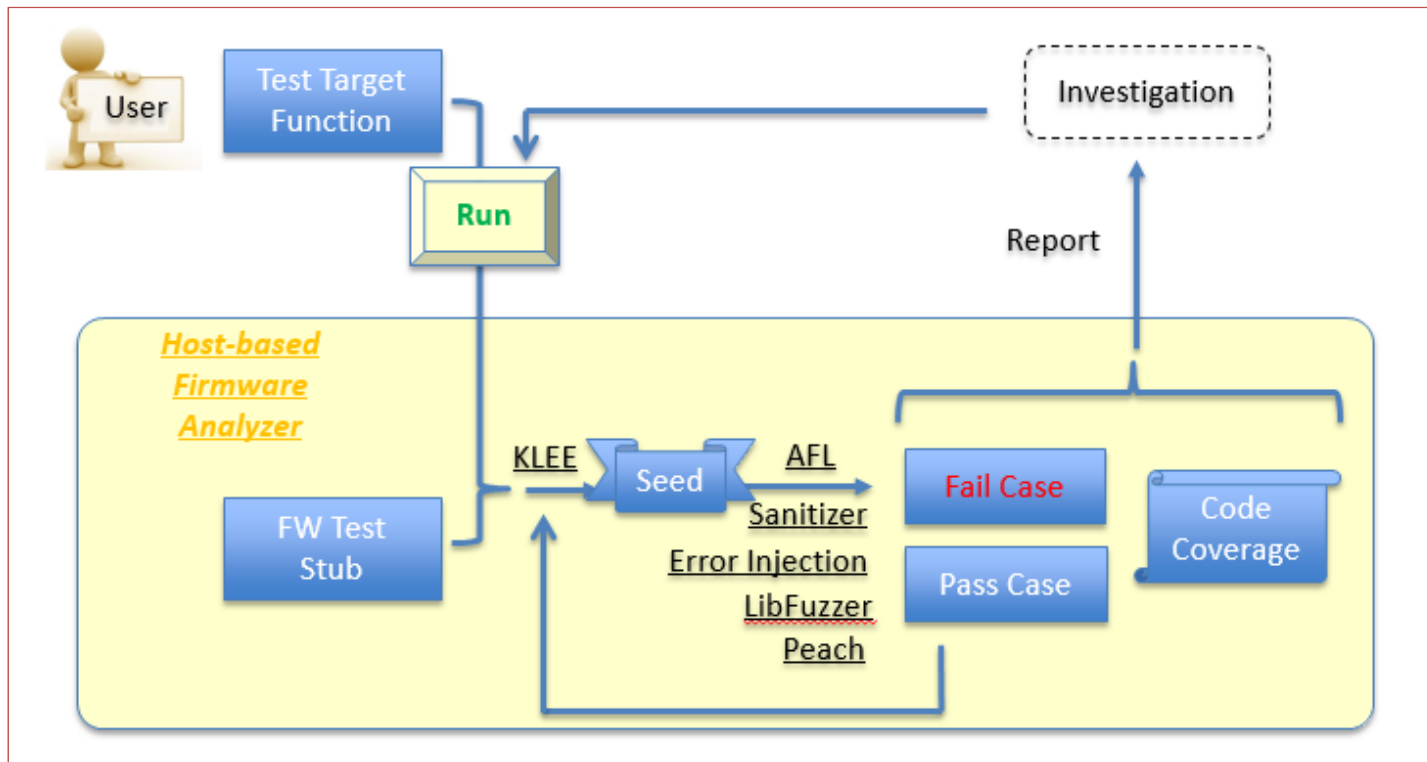
- [CHIPSEC](#): open source framework for analyzing the security of platform firmware and hardware configuration at runtime, based on the Unified Extensible Firmware Interface ([UEFI](#)) specification.
- [Code Coverage](#): Tools like the [Intel® Intelligent Test System](#) measure the amount of firmware code executed during test runs (high percentage is better).
- [Symbolic Execution and Virtual Platforms](#): Intel's [Excite](#) project uses a combination of symbolic execution, fuzzing, and concrete testing to find vulnerabilities in firmware running on Wind River* [Simics](#)* virtual platforms.
- *Great tools... but they're based on integration testing, so issues are more expensive to detect and mitigate. Can we improve testing before integration?*

Examine Tools for Driver & Application Developers



- Fuzzing: Test application programming interfaces (APIs) by subjecting them to random, invalid, unexpected, or untrusted (potentially malicious) inputs.
- Address Sanitizing: Detect memory corruption issues such as heap buffer overflow, stack buffer overflow, and global buffer overflow.
- Code Coverage: Identify code paths not executed during validation so test scope can be increased to avoid corner cases.
- ***The Challenge**: How can firmware developers use OS-based test tools on isolated firmware components?*

Host-based Firmware Analyzer (HBFA)



Testing UEFI firmware drivers in the developer's environment

Host-based Firmware Analyzer

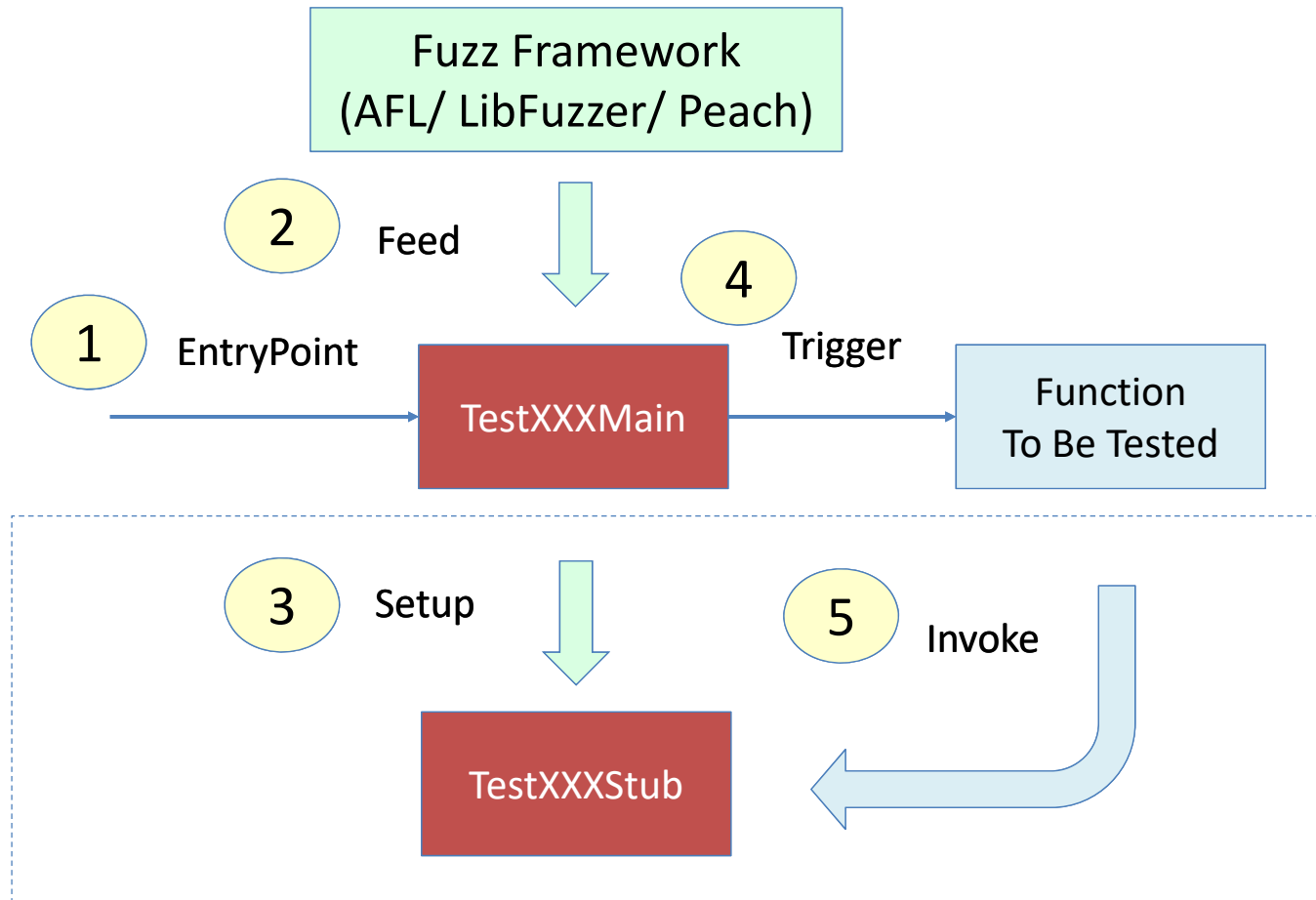
OS based environment utilizing best-in-class test tools



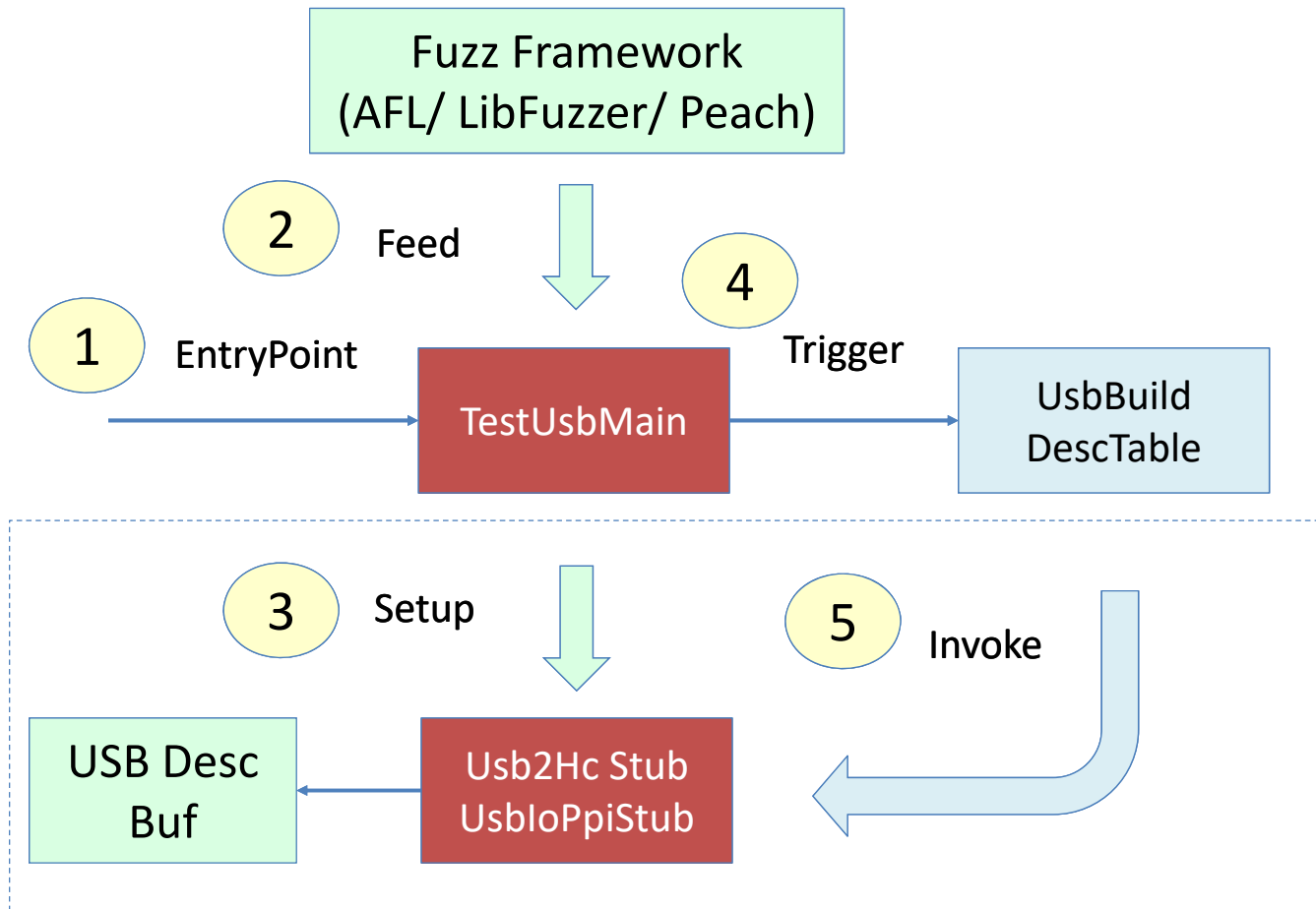
- GUI and command-line interfaces
- Fuzzing testing
 - AFL, libFuzzer, Peach
- Symbolic execution (KLEE/STP)
- Address Sanitizer & Code Coverage
- Automated unit test execution (Cunit)
- Instrumentation methods for fault injection and trace
- Database of unit test cases

The screenshot displays the 'Host-based Firmware Analyzer' interface. At the top, there are buttons for 'Import', 'Method', 'Seed', 'Exec', 'Report', and 'CodeCoverage'. Below these, the 'Test Case' list includes 'TestFmpAuthenticationLibR', 'TestPartition', 'TestPeiUsb', 'TestTpm2CommandLib', 'TestUdf' (highlighted), 'TestUsb', and 'TestVariableSmm'. The 'Test Method' section has checkboxes for 'KLEE(STP)', 'AFL' (checked), 'libFuzzer', 'Sanitizer', and 'Peach'. The 'Seeds' list contains several binary files under the path '/home/tiano/SEEDS/UDF/'. At the bottom, a 'TestUdf Test Report Summary' box shows: 'Total Seeds: 279', 'Crashes: 37', 'Hangs: 7', and 'Execution Time: 0 days,18 hrs,53 min,31 sec'.

Host-based Firmware Analyzer - Case Design



Host-based Firmware Analyzer – USB Test



Example - AFL



```
american fuzzy lop 2.52b (TestBmpSupportLib)

process timing |-----| overall results
  run time : 0 days, 0 hrs, 2 min, 1 sec | cycles done : 22
  last new path : 0 days, 0 hrs, 1 min, 3 sec | total paths : 42
  last uniq crash : none seen yet | uniq crashes : 0
  last uniq hang : none seen yet | uniq hangs : 0
-----|-----|
cycle progress |-----| map coverage
  now processing : 25 (59.52%) | map density : 0.04% / 0.26%
  paths timed out : 0 (0.00%) | count coverage : 1.32 bits/tuple
-----|-----|
stage progress |-----| findings in depth
  now trying : splice 7 | favored paths : 25 (59.52%)
  stage execs : 15/16 (93.75%) | new edges on : 27 (64.29%)
  total execs : 1.11M | total crashes : 0 (0 unique)
  exec speed : 9082/sec | total tmouts : 0 (0 unique)
-----|-----|
fuzzing strategy yields |-----| path geometry
  bit flips : 20/64.1k, 4/64.1k, 0/64.0k | levels : 3
  byte flips : 0/8018, 0/2781, 1/2707 | pending : 0
  arithmetics : 5/157k, 0/102k, 0/59.9k | pend fav : 0
  known ints : 0/12.1k, 0/53.3k, 0/92.3k | own finds : 32
  dictionary : 0/0, 0/0, 0/24.8k | imported : n/a
  havoc : 2/185k, 0/215k | stability : 100.00%
  trim : 13.77%/3610, 64.04%
-----|-----|
                                                                    [cpu000:117%]
```

Example – Peach + Sanitizer



```
tiano@tiano-Vostro-3902: ~/TEST/edk2
=====
==9294==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x602000000032
at pc 0x00000052c278 bp 0x7fffffffcca0 sp 0x7fffffffcc98
READ of size 2 at 0x602000000032 thread T0
#0 0x52c277 in UsbBuildDescTable /home/tiano/TEST/edk2/UefiHostTestPkg/TestC
ase/MdeModulePkg/Bus/Usb/UsbBusDxe/UsbDesc.c:830:64
#1 0x5297ef in main /home/tiano/TEST/edk2/UefiHostTestPkg/TestCase/MdeModule
Pkg/Bus/Usb/UsbBusDxe/TestUsb.c:94:3
#2 0x7ffff6ee582f in __libc_start_main /build/glibc-Cl5G7W/glibc-2.23/csu/..
/csu/libc-start.c:291
#3 0x41a728 in _start (/home/tiano/TEST/edk2/Build/UefiHostTestPkg/DEBUG_CLA
NG8/X64/TestUsb+0x41a728)

0x602000000032 is located 1 bytes to the right of 1-byte region [0x602000000030,
0x602000000031)
allocated by thread T0 here:
#0 0x4e945f in __interceptor_malloc /home/tiano/Downloads/llvm/projects/comp
iler-rt/lib/asan/asan_malloc_linux.cc:146
#1 0x52d1e4 in AllocateZeroPool /home/tiano/TEST/edk2/UefiHostTestPkg/Librar
y/MemoryAllocationLibHost/MemoryAllocationLibHost.c:37:12
#2 0x52bbe1 in UsbGetOneConfig /home/tiano/TEST/edk2/UefiHostTestPkg/TestCas
e/MdeModulePkg/Bus/Usb/UsbBusDxe/UsbDesc.c:744:9
#3 0x52bfc9 in UsbBuildDescTable /home/tiano/TEST/edk2/UefiHostTestPkg/TestC
ase/MdeModulePkg/Bus/Usb/UsbBusDxe/UsbDesc.c:814:14
```