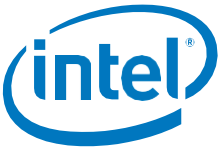


WHITE PAPER

Firmware Tools

Using Host-based Firmware Analysis to Improve Platform Resiliency

February 2019

Computer platform firmware is a critical element in root-of-trust. Firmware developers need a robust tool set to analyze and test firmware components, enable detection of security issues prior to platform integration, and reduce validation costs. Intel applies best practices for software development to deliver an industry-leading framework for automating the testing of firmware components prior to integration.

Intel has developed a new firmware tool, Host-based Firmware Analyzer (HBFA), for the TianoCore open source community. HBFA allows open source developers to run advanced testing tools such as fuzz testing, symbolic execution, and address sanitizers in a system environment.

Authors

Brian Richardson

Chris Wu

Jiewen Yao

Vincent J. Zimmer

Contents

Intended Audience

This white paper is intended for firmware engineers, platform designers, and system developers.

Introduction

Because firmware is a low-level part of a computer platform, it has potential security risks that may not be apparent to developers. Platform firmware executes after reset, initializes hardware, and performs a handoff to the operating system (OS). This makes firmware an essential component in software root-of-trust.

> *"Firmware has become more popular in the world of computer security research. Attacks operating at the firmware level can be difficult to discover and have the potential to persist across platform recovery."*
> *– 'Breaking Firmware for Fun and Profit… and Security', Brian Richardson, Intel Developer Zone*

The most common methods of firmware validation are traditional unit and integration testing, with emphasis on functionality and stability. Advanced features require more complex approaches that rely on a combination of open source projects, proprietary drivers, and platform-specific customizations. These methods are more efficient thanks to automation and improved test coverage, but make it difficult to isolate errors in specific components.

Current State of Firmware Security Testing

Intel makes significant investments in firmware security and improved overall product quality, using a Secure Development Lifecycle (SDL) process. Intel publishes security design guidelines, and develops platform features such as Intel® Device Protection Technology with Boot Guard (Boot Guard), to help address potential security issues.

Intel also develops open source tools, including the following:

- CHIPSEC: CHIPSEC is a framework for analyzing the security of platform firmware and hardware configuration at runtime, introduced in 2014. Threat models are based on the Unified Extensible Firmware Interface (UEFI) specification. Tests are based on published security research and best practices for platform configuration. CHIPSEC includes a security test suite, tools for accessing low-level interfaces, and forensic capabilities.
- Code Coverage: The Intel® Intelligent Test System is a test automation framework that uses code coverage to measure the amount of firmware code executed during test runs. A program with a high percentage of test coverage has more code executed during testing, which lowers the chance of shipping with undetected issues.
- Symbolic Execution and Virtual Platforms: Intel's Excite project, in development since 2015, uses a combination of symbolic execution, fuzzing, and concrete testing to find vulnerabilities in sensitive code. It uses the Wind River* Simics* virtual platform to replay tests while checking for security issues and measuring coverage.

While these elements have provided improvements in overall firmware security, they are all designed to operate on fully integrated platform firmware. Most firmware security tests happen after system integration, when bugs are more expensive to mitigate than those found during design or implementation (see Figure 1 in this article). Developers need a robust toolset for analyzing firmware components prior to unit testing or integration testing.

> *"Finding vulnerabilities in code is part of the constant security game between attackers and defenders. An attacker only needs to find one opening to be successful, while a defender needs to search for and plug all or at least most of the holes in a system. Thus, a defender needs more effective tools than the attacker to come out ahead."*
> *– 'Finding BIOS Vulnerabilities with Symbolic Execution and Virtual Platforms', Jakob Engblom, Intel Developer Zone*

Introducing the Host-based Firmware Analyzer

Intel has developed HBFA as a contribution to the TianoCore open source firmware community. HBFA enables advanced testing of UEFI drivers and UEFI Platform Initialization (PI) drivers in the developer's OS environment. See Figure 1 for a sample screenshot of the HBFA user interface.

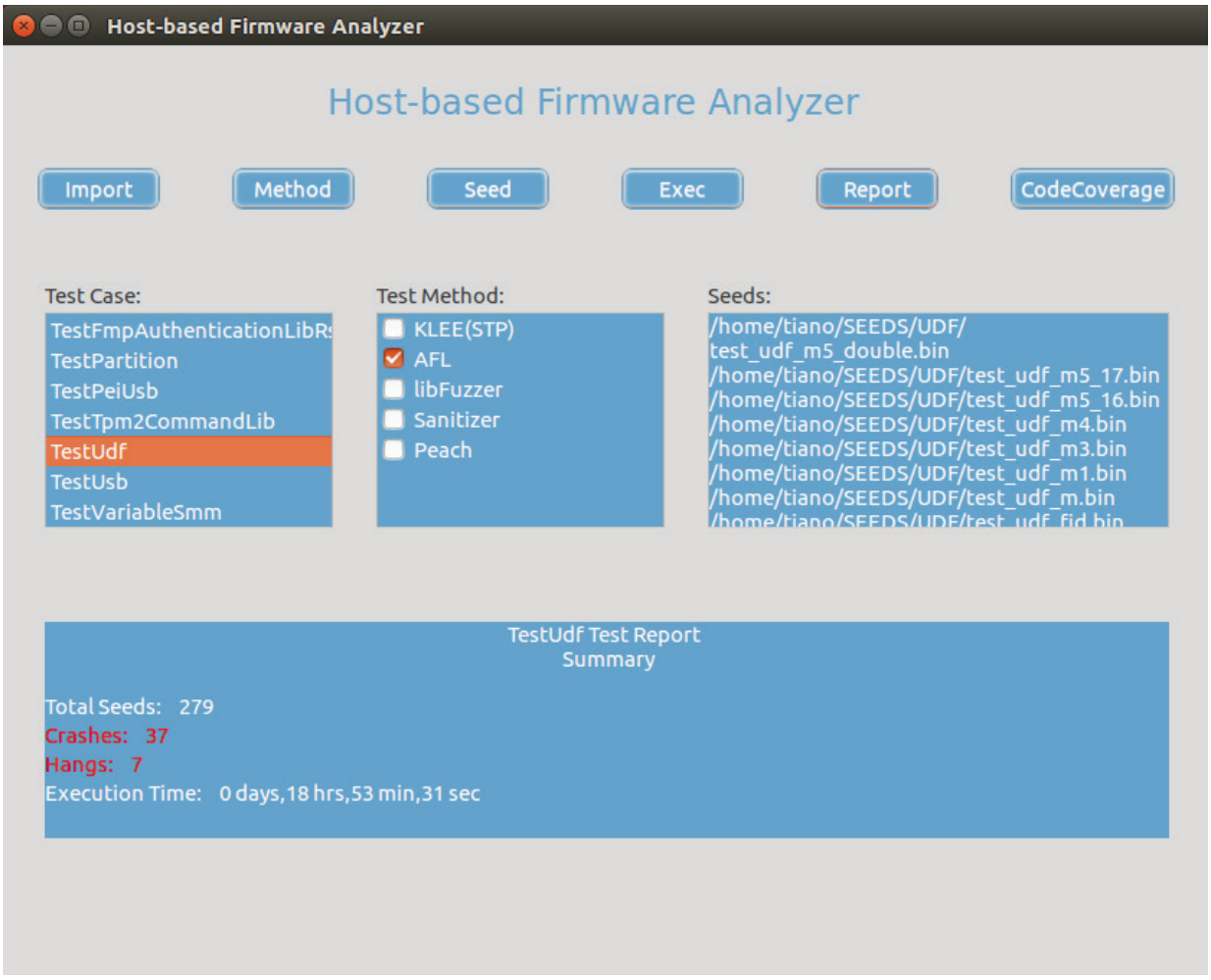
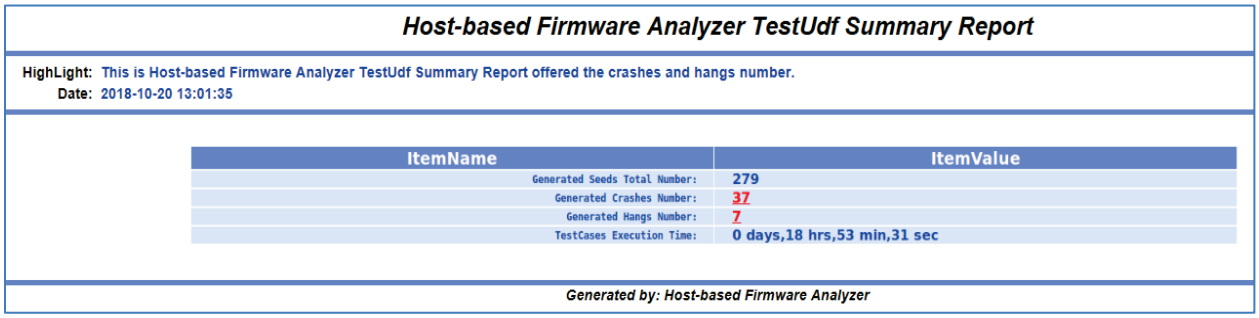


Figure 1: HBFA User Interface

The HBFA test suite uses existing open source tools to offer a variety of features:

- GUI and command-line interfaces
- Execute common fuzzing frameworks (AFL, libFuzzer, Peach)
- Supports symbolic execution (KLEE/STP)
- Incorporates Address Sanitizer
- Automated unit test execution via CUnit
- Generate code coverage reports (GCOV/LCOV)
- Instrumentation methods for fault injection and trace
- Database of unit test cases
- Test reports with extended debug information



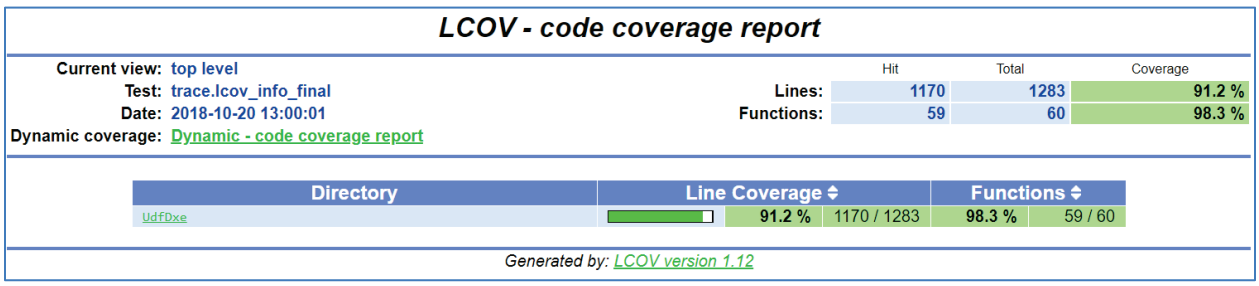
Host-based Firmware Analyzer TestUdf Summary Report

HighLight: This is Host-based Firmware Analyzer TestUdf Summary Report offered the crashes and hangs number.
Date: 2018-10-20 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 2: HBFA TestUdf Summary Report



LCOV - code coverage report

Current view: top level
Test: trace.lcov_info_final
Date: 2018-10-20 13:00:01
Dynamic coverage: Dynamic - code coverage report

	Hit	Total	Coverage
Lines:	1170	1283	91.2 %
Functions:	59	60	98.3 %

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

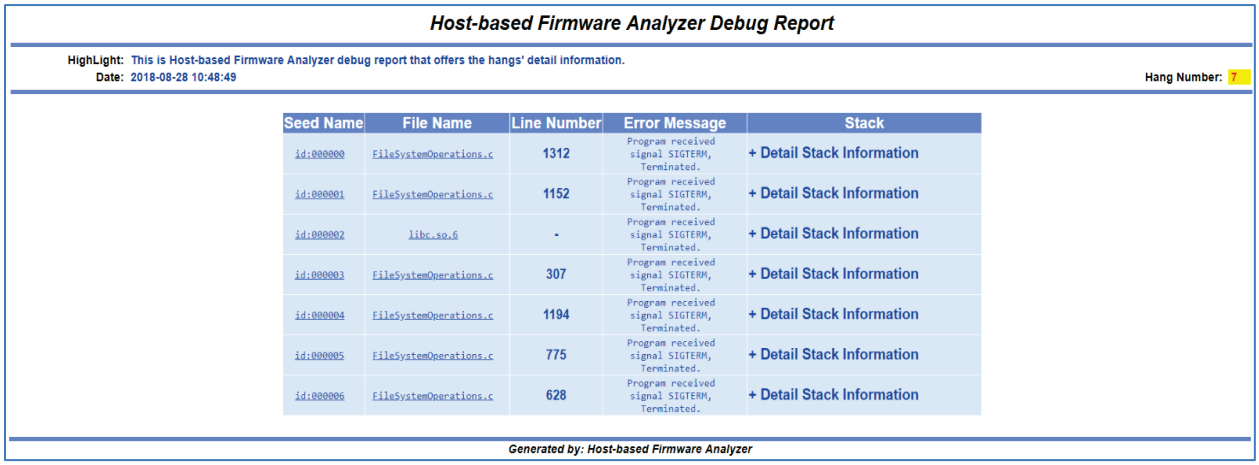
Generated by: LCOV version 1.12

Figure 3: LCOV Code Coverage Report

Firmware developers can use HBFA for security unit testing early in the development phase. HBFA helps to enable firmware testing under an OS environment, based on stub functions from the TianoCore EDK II project. This allows early detection of security issue prior to integration, reducing validation costs. HBFA also provides test summaries, debug information, and code coverage reports that facilitate quick issue resolution.

```
3741 :   if (RecordingFlags == LongAdsSequence) {
3683 :     return GetLongAdLsn (Volume, (UDF_LONG_ALLOCATION_DESCRIPTOR *)Ad, Lsn);
  58 :   } else if (RecordingFlags == ShortAdsSequence) {
  58 :     PartitionDesc = GetPdFromLongAd (Volume, ParentIcb);
  58 :     if (PartitionDesc == NULL) {
   0 :       return EFI_UNSUPPORTED;
     :     }
     :
  58 :     *Lsn = GetShortAdLsn (
     :               Volume,
     :               PartitionDesc,
     :               (UDF_SHORT_ALLOCATION_DESCRIPTOR *)Ad
     :               );
  58 :     return EFI_SUCCESS;
     :   }
     :
     :   //
     :   // Code should never reach here.
     :   //
   0 :   ASSERT (FALSE);
   0 :   return EFI_UNSUPPORTED;
     : }
```

Figure 4: Code Coverage Analysis



Host-based Firmware Analyzer Debug Report

HighLight: This is Host-based Firmware Analyzer debug report that offers the hangs' detail information.
Date: 2018-08-28 10:48:49
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 5: HBFA Debug Report

HBFA focuses on pure software logic, and tests module APIs independently, so test cases can be executed at the module level. This method reduces the effort required to root-cause issues when integrated with numerous components from open source projects, firmware vendors, or third-party device manufacturers.

The HBFA test case database allows developers and validation engineers to share a common set of examples, add new cases, and share test cases with partners. This improves security test efficiency across the industry, by reusing and refining tests across projects.

Incorporating Best-in-Class Software Practices

Intel's work with Excite and code coverage demonstrates the ability to incorporate high-level software testing into firmware development. These tests increase development efficiency and validate using best practices. OS developers have a variety of tools that can detect commonly exploitable issues in drivers, libraries, and programs prior to deployment:

- 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 for firmware developers is designing test cases that utilize these methods, building an environment that can isolate firmware components in an OS environment, and executing these test cases prior to platform integration.

Validation Improvements

Intel HBFA developers reviewed a set of known firmware issues, and sorted them into eight categories:

- External input
- Race condition
- Hardware input
- Secret handling
- Register lock
- Configuration
- Replay/rollback
- Cryptograph(Key)

These categories are used to develop test guidelines for detecting potential vulnerabilities. Additional categories can be added based on new research. Intel also offer guidelines for secure code design, code review, test strategies for EDK II, and test tool development. Test automation is easier when code follows "design for testing" guidelines.

Summary

Because firmware is a low-level part of a computer platform, it has potential security risks that may not be apparent to developers. Intel has developed the Host-based Firmware Analyzer (HBFA) as a new testing tool for the TianoCore open source firmware community, using software development best practices. HBFA enables advanced testing of UEFI drivers and UEFI Platform Initialization (PI) drivers in the developer's OS environment.

HBFA focuses on pure software logic, and tests module APIs independently, so test cases can be executed at the module level. The HBFA allows developers to run advanced tools such as fuzz testing, symbolic execution, and address sanitizers in a system environment.

Intel also offers guidelines for secure code design, code review, test strategies for EDK II, and test tool development. Test automation is easier when code follows "design for testing" guidelines.

Intel plans to release the HBFA as an open source tool in Q2 of 2019. For more information, please see the HBFA section in the TianoCore community wiki.

Notices & Disclaimers