UEFI Firmware Security Best Practices

Presented by: Dick Wilkins, PhD

Principal Technology Liaison

presented by

UEFI Plugfest – May 2014

Agenda

Introduction

Threats and Vulnerabilities

Mitigation Guidelines

Validation Guidelines

Next Steps

Questions

Introduction

- UEFI firmware is now being widely deployed and is becoming a target for hackers and security analysts
- Poor implementations affect the credibility of the UEFI “brand” and market perception of all implementations
- As with all software implementations, there are going to be faults (Ours are not perfect by any means)
- Phoenix would like to share some of our best practices in the interest of increasing communication and raising the quality and security of all our implementations

Threats and Vulnerabilities

Firmware is software, and is therefore vulnerable <u>to the same threats that typically target software</u>

- Maliciously crafted input
- Elevation of privilege
- Data tampering
- Unauthorized access to sensitive data
- Information disclosure
- Denial of Service
- Key Management
- Etc.

Threats and Vulnerabilities

Firmware-Specific Threats

- Maliciously crafted input – Buffer overflows to inject malicious code
- Elevation of privilege – SMM code injection
- Data tampering – Modifying UEFI variables (SecureBoot, Configuration, etc.)
- Unauthorized access to sensitive data – Disclosure of SMM contents
- Information disclosure – OpenSSL 1.0.1 through 1.0.1f (Heartbleed)
- Denial of Service – Corrupting UEFI variables to brick the system
- Key Management – Private Key Management for signed capsule updates

Threats and Vulnerabilities

We Are All At Risk!

Disclosures regarding UEFI BIOS security vulnerabilities look bad for the whole UEFI community!

So how to protect against Firmware attacks?

Mitigation Guidelines

Many organizations have provided disclosures and guidelines for developing more secure firmware!

Examples come from Intel, Microsoft, Mitre, NIST, Linux distros and others. Some are public and some are available only under NDA via direct communications with the involved companies.

Mitigation Guidelines

Key areas for concern

- BIOS Flash Regions
- UEFI Variables in Flash
- Capsule Updates
- SMRAM
- Secure Boot

Mitigation Guidelines

BIOS Flash Regions

- Lock System Firmware regions as early as possible
- Set SMM BIOS Write Protect
- Set BIOS Lock Enable and Implement SMI handler
- Lock Protected Range Registers for SPI Flash

Mitigation Guidelines

UEFI Variables in Flash

- Lock Authenticated Variable regions as early as possible
- Separate integral configuration and security-based variables from more widely modifiable variables
- Reduce permissions to only what is needed
  - Remove RT access for Firmware-usage-only variables
  - Set variables as Read-Only if they are not intended to be modified
- Verify non-authenticated variable values before usage
  - Fallback to defaults if corrupted

Mitigation Guidelines

OEMs and ODMs want to be able to modify variables after boot in manufacturing

- Adding security controls to these variables needs to be managed carefully to not break critical manufacturing infrastructure
- But this is not an excuse for leaving these variables unprotected

Consider adding extra variable integrity validity checks on critical values to prevent “bricking” of systems should a value be improperly changed

Mitigation Guidelines

Capsule Updates

- Enforce Signed Capsule Updates by default
- Enforce Rollback Protection by default
- Private Key Protection
  - HSM or Signing Authority used to create Signed Capsules

Mitigation Guidelines

SMRAM

- Enable Hardware Protections
  - Lock SMRAM as early as possible
  - Lock SMIs
- Never execute code outside SMRAM from within SMM
- Always validate data and address region limits when copying data to and from buffers

Mitigation Guidelines

Secure Boot

- Disable CSM
- Set image verification defaults to secure values:
  - DENY_EXECUTE_ON_SECURITY_VIOLATION
  - QUERY_USER_ON_SECURITY_VIOLATION
- Disallow fallback to legacy boot
- Store CSM Enable and all Secure Boot variables as Authenticated Variables in protected flash
- Require User-Presence to disable Secure Boot

Validation Guidelines

For complex systems, “Bug-Free” does not exist!

- **Bugs provide a means to compromise a system!**

Validation Guidelines

Challenges of developing "Bug-Free" Firmware

- There are thousands and thousands of lines of code
  - Manual review of code and code paths is impractical
- There are multiple settings that must all be configured properly
  - Test case matrixes for all use-cases can be overwhelming
- Even widely-accepted "safe" code can be found vulnerable
  - OpenSSL 1.0.1 through 1.0.1f (Heartbleed)
- Systems rarely use the most current and secure code base
  - Customers nearing production will not risk source code changes

Validation Guidelines

Many organizations have provided disclosures of known issues and guidelines for validating firmware security!

- i.e. Intel, NIST, Mitre, Microsoft

The UEFI security sub teams and Board are discussing the possibility of providing security validation suites to supplement the existing SCT tests. If you have thoughts on this, please let your representatives know

Validation Guidelines

Targeted Code Reviews

- Variable Separation
  - Firmware Usage Only
  - Read Only
  - Security Controls
- External Facing code and SMI Handlers
  - Validate input parameters
  - Proper bounds checking

Validation Guidelines

Validation Tools – real-time security testing

- Intel CHIPSEC utility
  - Manually test variable access in Win8/8.1
- UEFI Shell DMPSTORE
  - Review variable accessibilities
- UEFI Shell SetVar
  - Corrupt variables, reboot, and verify system behavior
  - Fill variable area, reboot , and verify system behavior
  - Erase all accessible variables, reboot , and verify system behavior

Validation Guidelines

Verify EFI variable-space-full handling

- History shows that this is a problematic area
- Thoroughly test on all platforms

Secure Boot Support – Take another look

- Review latest disclosures and findings
- Verify implementation and fix any vulnerabilities

Next Steps

What Phoenix is Doing

- Performing targeted code reviews
- Developing security test tools and integrating into our QA process
- Reviewing disclosures and guidelines, and verifying our implementations
- Back porting security fixes to previous codebases
- Working with customers to educate them on important security fixes
- Monitoring the EDK2 codebase for important security fixes
- Investigating emerging specifications, such as NIST SP800-155

Next Steps

Everyone that provides pre-OS code, and that includes firmware OptionROM code and EFI applications, needs to follow similar steps to validate their implementations

The UEFI forum working groups and the board are discussing steps that can be taken to insure the quality and security of implementations to minimize any negative effects on our customers and end users.

This will serve to reduce negative press and perceptions of UEFI and our products in the marketplace.

Additional Reading and References

Intel – Documents released to the UEFI USST (see the group docs page)

- UEFI Variable Usage Technical Advisory Overview – April 2014
- UEFI-USST-Variable-Usage-001.pdf - April 2014

Mitre

- BIOS Chronomancy: Fixing the Static Root of Trust for Measurement – 2013
- Defeating Signed BIOS Enforcement – 2014
- Setup For Failure: Defeating Secure Boot - 2014

Black Hat

- A Tale of One Software Bypass of Windows 8 Secure Boot – 2013

Invisible Things Lab

- Attacking Intel BIOS – July 30, 2009

Questions?

For more information on the Unified EFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by