Secure Coding for UEFI Firmware

Presented by UEFI Forum

Tuesday, March 12, 2019

Secure Coding

Welcome & Introductions

Moderator: Brian Richardson
Firmware Ecosystem Development
Member Company: Intel Corporation
@intel_brian

Panelist: Trevor Western
Member Company: Insyde Software

Panelist: Eric Johnson
Member Company: American Megatrends Inc.

Panelist: Dick Wilkins
Member Company: Phoenix Technologies

Dick Wilkins

Phoenix Technologies

What is the threat model for firmware?

Security should be a priority

- As Apps and OSes become more secure, firmware is a bigger target
- If platform firmware is compromised, that system cannot be secure

Assume a hostile environment

- Check every external input
- Never rely on “security by obscurity”
- Minimize your attack surface (disable unneeded features)

Debug and security protection

- There are various compiler tools and build options for more secure firmware
  - Many have been added to the TianoCore EDK II environment
  - Enable these options during development
  - Examples: ASLR, NX, /GS

But…

- Remove debug interfaces (backdoors) in shipping code, hardware and software
- Be very careful of remote management interfaces (be careful of BMCs)
- ASSERTS in your code
  - ASSERTs are for catching bugs that should never happen
  - ASSERTs are not for catching possible errors or validating inputs

SMM is particularly dangerous

- Insecure SMM code can run amok at Ring 0/1
  - It is a good place to focus your security code reviews
  - SMM code must never call out of SMRAM
  - SMM code must copy input parameters and validate and use the copy, to prevent time-of-check-time-of-use (TOCTOU) vulnerabilities

Protection settings

- Flash memory protections should be properly set as early as possible
- Make sure this happens on S3 resume as well as boot
- Lock authenticated EFI variable regions early
- Set variables read-only if possible
- Make sure your code falls back to reasonable defaults if variables are compromised (prevent Denial of Service)

Trevor Western

Insyde Software

How do we compensate for "C" language insecurities?

The Insecurity of ‘C’

- ‘C’ is the most popular low-level systems programming language in the world
- ‘C’ is a very powerful and very dangerous programming language

The Insecurity of 'C'

- C has no mechanism to test that a memory pointer is valid – does the pointer really point to an actual memory type as intended?
- C permits code to access memory beyond the memory allocated and assigned to a function. For example, code can modify a function's return address in memory. Highly insecure!
- Code can be manipulated like data. Passing function addresses into routines. Easy to execute arbitrary code
- 'C' can be very complex. For example, a declaration of a 'pointer to an array of functions that return a pointer to an array of functions' is legal
- Syntax is subtle and prone to mistakes. Comparison and assignment operators are 1 character different and visually hard to distinguish

The Insecurity of ‘C’

- Naturally Programmers are making lots of security-related mistakes in C and UEFI
  - Microsoft at the recent BlueHat conference revealed: “70% of all vulnerabilities were memory safety issues.” “Terms like buffer overflow, race condition, page fault, null pointer, stack exhaustion, heap exhaustion or corruption, use after free, or double free --all describe memory safety vulnerabilities.”

Making ‘C’ Less Insecure

- ‘C’ compilers are getting better:
  - Turn on all warning options
  - Enable stack overflow checks / heap checking. Now available in EDKII
- Ban the use of unsafe C library functions
  - Use the StrN*S functions like StrnlenS(). Available in open source libs, such as EDKII
  - Ban the use of complex functions with variable arguments, like print() or InstallMultipleProtocolInstances()

Making 'C' Less Insecure

- Ban use of #pragmas and casts that tell the compiler to ignore the warnings or errors
- Assume that all arithmetic used to calculate memory allocations is wrong.
  - Any code used to determine array offsets or memory allocation should be removed, especially if it is using signed integers.
- Run SCA tools
  - Tools are better than ever and able to handle complexity
  - Klocwork & Coverity are two of the most widely used
    - MS VS2017 now has a usable SCA feature (too many FPs on VS2015)

Other Languages

- Every Programming Language Has Weaknesses:
  - *“24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them”* shows that most security issues can be seen in several programming languages
  - RUST – ‘C’-like, but focusses on memory-safety and concurrency. Works well in low-resource devices. See https://www.rust-lang.org/
- Security comes from following a security practice like SDL, not the coding language

Eric Johnson

American Megatrends, Inc.

How do we validate specific kinds of insecurities?

Firmware is hard to validate

- Code comes from many sources
- Firmware must be stable before you can test
- Configuration changes affect validity of tests

Code validation techniques

- Static Code Analysis Techniques
  - Code Review
  - Static Code Analyzer
- Dynamic Code Analysis Techniques
  - Integration Testing
  - Unit Testing
  - Symbolic execution

When to add new unit tests

- Fix a vulnerability
- Code that crosses trust boundary
- Developing new code
- Refactoring / bug fixing old code

Unit testing SMI handlers

- Test each structure / pointer controlled by adversary
- Test conditional branches controlled by adversary
- Goal is 100% code coverage
  - Symbolic Execution can help achieve this goal

However…

- Full code coverage is impossible on complex projects
  - Prioritize privileged code
  - Use a combination of validation techniques. i.e. fuzz testing, code review

Open Source Code Validation Tools

- Symbolic Execution:
  - angr
  - CRETE (already used on TianoCore)
  - KLEE
  - And more. See Wikipedia
- Unit Testing Frameworks
  - Host-based Firmware Analyzer (available Q2)
  - MicroPython Test Framework for UEFI

Secure Coding Panel Discussion

Questions?

Thank you!

Join the UEFI Forum and become part of the solution:

- www.uefi.org/membership

Contact the UEFI Forum:

- admin@uefi.org

Contact the USRT:

- For more information go to: www.uefi.org/security
- Email a firmware security issue or vulnerability to: security@uefi.org

More Resources

- [Intel] "*A Tour Beyond BIOS – Security Design Guide in EDK II*", September 2016
- [Howard] "***24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them***", Michael Howard, David LeBlanc, John Viega, McGraw-Hill, 2009, ISBN: 978- 0071626750
- [Apple] "***Secure Coding Guide***", September 2016
- [Intel] "*Using Host-based Firmware Analysis to Improve Platform Resiliency*", March 2019