presented by

UEFI Firmware – Securing SMM

UEFI Spring Plugfest – May 18-22, 2015

Presented by Dick Wilkins, Ph.D.

Principal Technology Liaison

Agenda

- Introduction
- SMM Attack Vectors
- Mitigation Strategies
- NX Protection in SMM
- Closing Remarks

HACKING BIOS CHIPS ISN'T JUST THE NSA'S DOMAIN ANYMORE

Vulnerability Note VU#631788

Multiple BIOS implementations permit unsafe SMM function calls to memory locations outside of SMRAM

Researchers Show Hacking Your Computer's BIOS is Child's Play

Noobs can pwn world's most popular BIOSes in two minutes

SMM is Under Attack

Attacking SMM Memory via Intel® CPU Cache Poisoning

Vulnerability Note VU#976132

Some UEFI systems do not properly secure the EFI S3 Resume Boot Path boot script

How Many Million BIOSes Would you Like to Infect?

Introduction

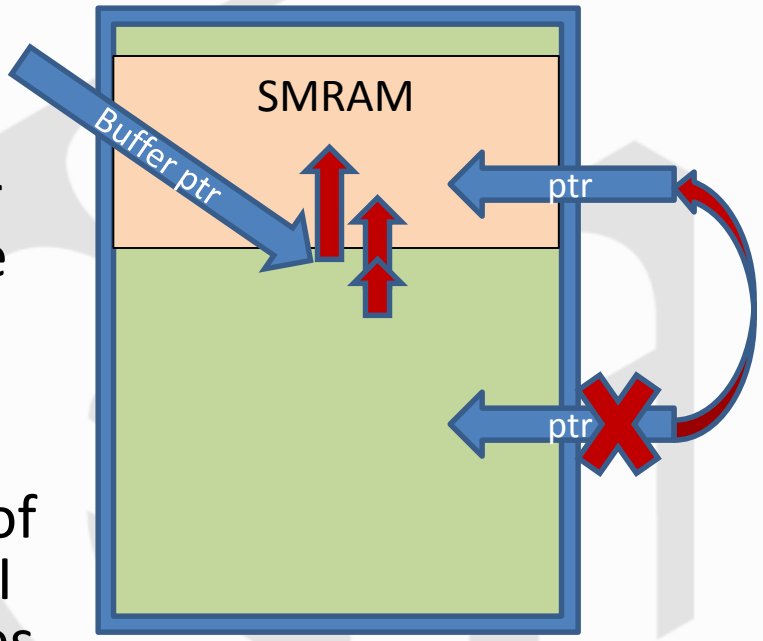
Why are we talking about this?

- We believe these vulnerabilities are widespread
- Some implementers may not be aware of their vulnerabilities
- We want to share our “best practices” in an effort to raise the level of security across the industry

SMM Attack Vectors

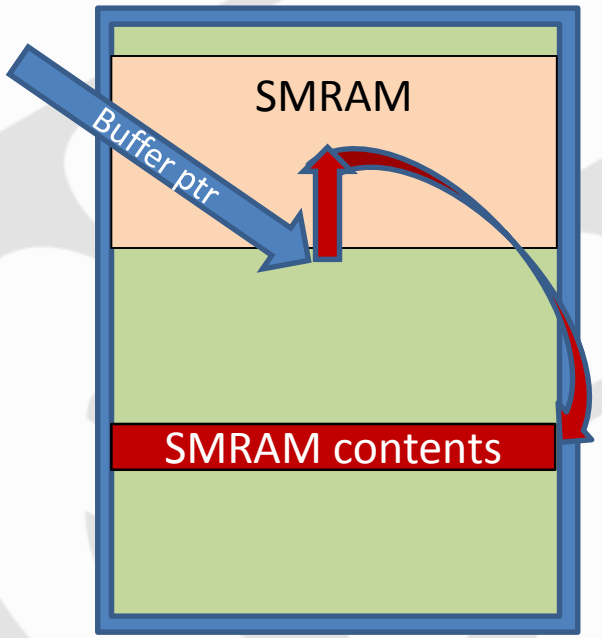
SMM Attack Vectors

- Corruption of SMRAM Contents
  - Pass in a buffer ptr just below SMRAM and request a write into the buffer
  - Pass in a buffer ptr and buffer size, then quickly increase the size to extend into SMRAM. If BIOS reads size twice, you might win the race
  - Modify a ptr located outside of SMRAM that is used in an SMI handler to perform data writes



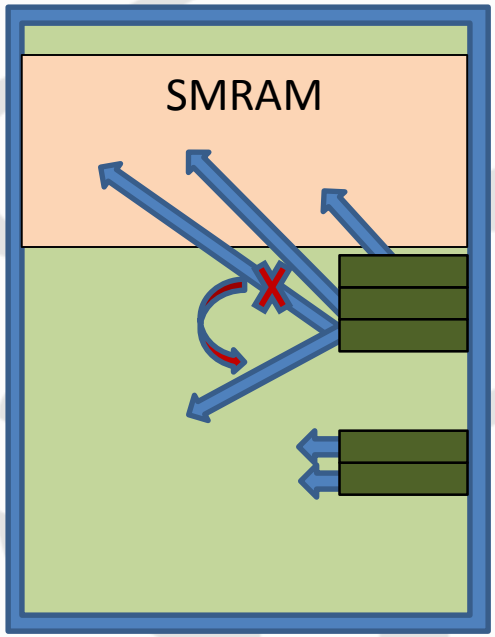
SMM Attack Vectors

- Disclosure of SMRAM Contents
  - Set a data ptr just below SMRAM with a large size, then trick SMI handlers into copying SMRAM contents into unprotected memory



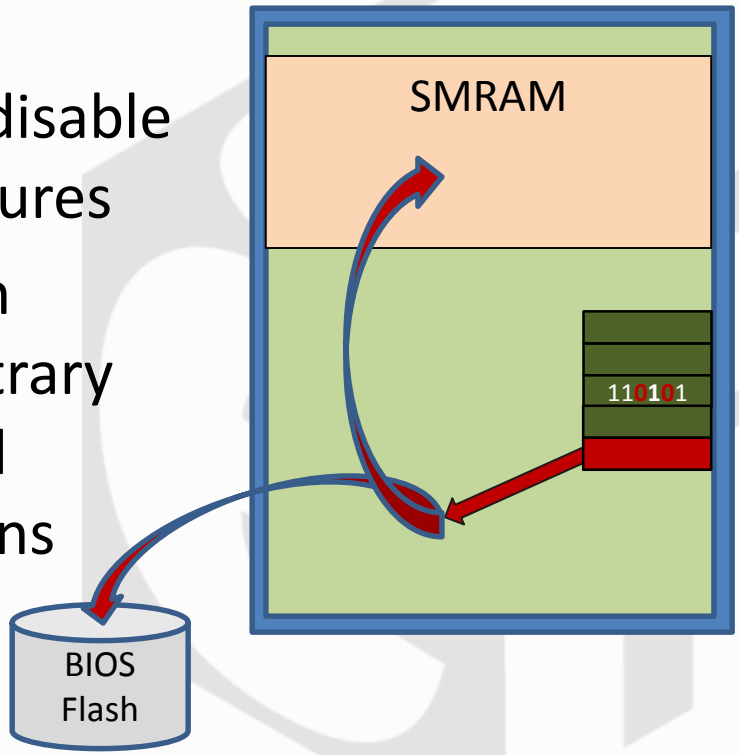
SMM Attack Vectors

- Arbitrary code execution in SMM
  - Modify a function ptr stored outside of SMRAM and used in SMI handlers to point to arbitrary code
  - Utilize an interface that remains available after it is no longer needed



SMM Attack Vectors

- S3 Boot Script Corruption
  - Modify settings stored outside of SMRAM to disable hardware security features
  - Modify to add dispatch OpCodes that call arbitrary code prior to Flash and SMRAM lock protections



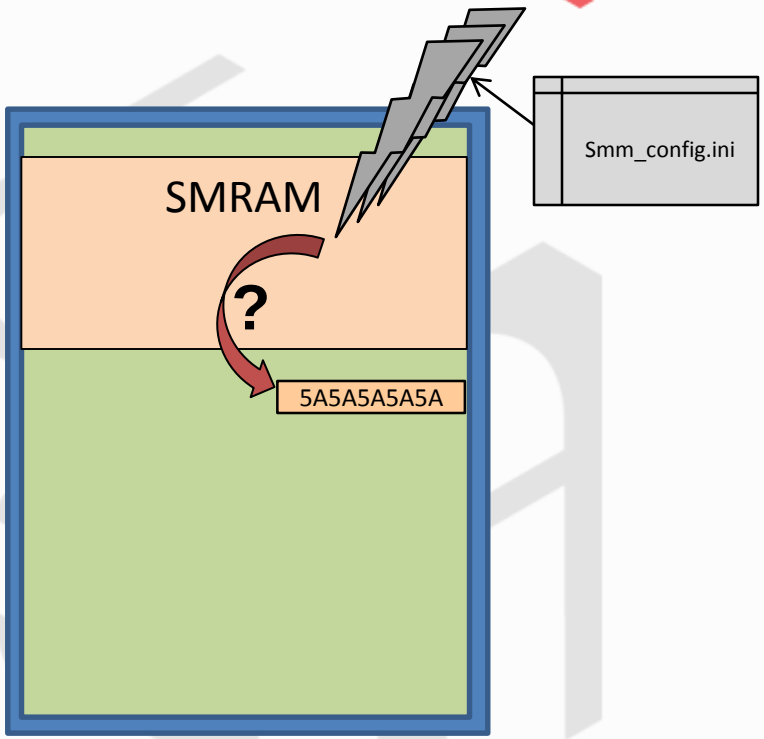
Mitigation Strategies

Mitigation Strategies

- Intel CHIPSEC Tool
  - Verifies SMRRs are configured properly
  - Verifies all lock bits are set (SMM, SPI Flash controller, etc.)
  - Verifies BIOS write protections are set
  - Verifies S3 Boot Script is not in unprotected memory
  - Supports additional silicon-specific tests
  - Extensible through new script files

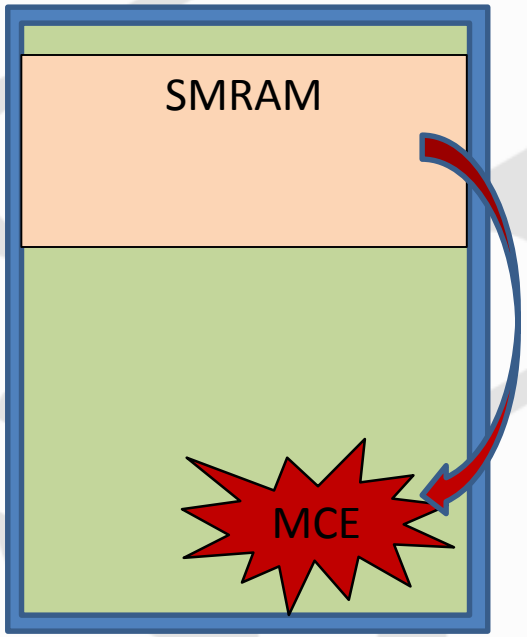
Mitigation Strategies

- New CHIPSEC SMI handler pointer validation module (smm_ptr.py)
  - Tests SMI handlers for possible pointer validation vulnerabilities
  - Writes a test pattern into a specified buffer
  - Triggers interrupts with the buffer pointer and tests if the contents were modified
  - If modified, the handler should be reviewed to ensure it performs proper range checking on input pointers



Mitigation Strategies

- Intel Haswell SMM Code Access Check bit
  - Execution of code outside SMRAM while inside SMM causes a Machine Check Exception
  - BUT, this feature is only available on Haswell and newer Intel processors!



Mitigation Strategies

- So, what can be done to provide this protection on systems that do not support the SMM Code Access Check hardware feature?

…Stay tuned…

Mitigation Strategies

- Product Development Security Strategy
  - SMM Code Access Check bit can be used to reveal unsecure drivers during development
  - This feature SHOULD be enabled early in the development cycle
  - This feature MUST be enabled on all production-level systems

Mitigation Strategies

- QA Security Strategy
  - CHIPSEC Tool should be used as part of the standard platform validation tests throughout the development life cycle
  - Exercising all typical user scenarios with SMM Code Access Check feature enabled can reveal any misbehaving code that was missed during development

Mitigation Strategies

So, does this mean we can rely on tools and hardware protections to ensure that SMM is secure?

Mitigation Strategies

NO!

- Tools do not typically support all silicon
- Tools cannot typically test all use cases
- OEM/ODMs may enable features that were disabled during QA, potentially enabling additional vulnerabilities
- OEM/ODMs may override core functions and implement proprietary SMI handlers
- Certain hardware features are only available on certain systems

Mitigation Strategies

This means we must also review every line of code in every SMI handler?!

Mitigation Strategies

- SMI Handler Code Review Checklist
  - Must validate all external data against buffer overruns and integer overflows
  - Must not expose APIs and features not needed at runtime
  - Must properly handle errors (do not use ASSERTs because these are removed in Release builds)
  - Must never call code outside of SMRAM
  - Must never call code through function pointers stored outside of SMRAM
  - Must not allow external pointers (e.g., BARs) to modify or disclose SMRAM contents
  - For structures that contain variable length or object counts, must verify that external input will not cause fetches that exceed the length of the structure

… as a starting list …

Mitigation Strategies

- Organizing the Code Review
  - Create the list of SMI handlers to review
  - Create a list of common unsecure coding patterns (e.g., gBS, gRT, gST usage)
  - When a new pattern is discovered, add it to the list of patterns
  - Use a centralized SMRAM Range Check function to help validate external input
  - Pay special attention to SMI handlers that operate using external input

Mitigation Strategies

- What Phoenix is Doing
  - Performed code reviews of all SMI handlers
  - Reviewed disclosures and guidelines, and verified our implementations
  - Back ported security fixes to previous codebases
  - Working with customers to educate them on important security fixes
  - Using security test tools through development and all platform validation phases
  - Developed an ongoing review process and new security strategies…

NX Protection in SMM

NX Protection in SMM

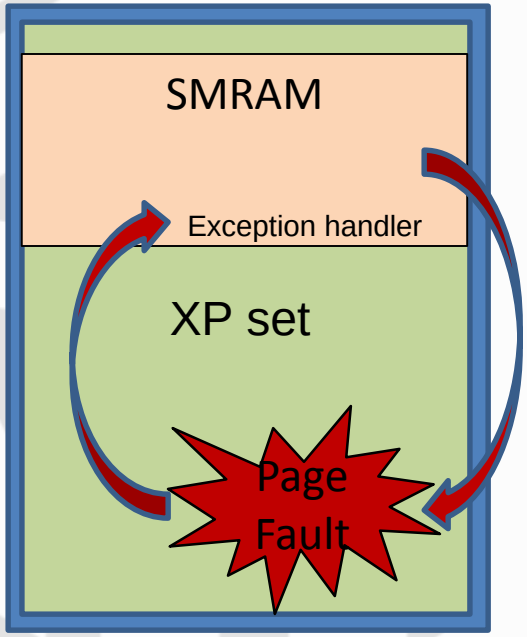
- In 2012 Phoenix worked with Microsoft to create and submit a proposal for NX (No-eXecute) protections in UEFI
- Modern x86 CPUs provide support for NX as a bit that can be set in PAE and IA32E page tables - called XD (eXecute Disable) in Intel Volume 3
- Setting the execute disable bit in a page table entry (PTE) causes the processor to page fault when fetching code from the associated page
- We experimented and found that these same protections could be used to prevent execution of code outside of SMRAM while in SMM
- We have offered this as a solution for systems that do not support the SMM Code Access Check hardware feature

NX Protection in SMM

- Changes to Support NX
  - SMI entry code is modified to set the NXE (No eXecute Enable) bit in IA32_EFER; on X64 targets, this is in addition to setting LME (Long Mode Enable) bit in the same register
    - On X64 targets, paging is already enabled to support LME, and we use existing page tables that are already in SMRAM; no additional work is required to allocate page tables or enable paging
    - On IA32 targets, page tables need to be allocated in SMRAM and code must also be added to select those page tables and enable paging
  - We added a function to read the attributes for current PTEs to the Phoenix page table management library originally developed for NX support
  - We added an SMM driver that uses our page table library to read current attributes and set the XD bit for every page that is outside of SMRAM
    - The map of SMRAM used by the centralized range checking library is used to identify the SMRAM regions

NX Protection in SMM

- How it works
  - On every SMI, the same page tables are selected, paging and NX support is enabled
  - The original state is already saved in SMM save state to be restored on exit
  - The page tables have been configured with the XD bit in every PTE that does not overlap with SMRAM
  - The CPU throws a page fault on any attempt to fetch code that is located in a page outside of SMRAM



NX Protection in SMM

- The results –
  - When we enabled NX protections, vulnerable drivers were exposed
  - We hooked the exception handler to quickly identify the unsafe drivers
  - Drivers were fixed and made available to our customers

Closing Remarks

Closing Remarks

- Everyone that provides UEFI code for systems that support SMM needs to follow similar steps to validate their implementations
  - Available tools must be run throughout development and during all platform validation phases
  - To ensure the strongest SMM security, all available hardware protections must be enabled in shipping systems
  - All SMI handlers must be analyzed for vulnerabilities using both tools and code review
  - All other protections must be enabled to prevent unauthorized writes to flash and SMM (e.g., signed capsule updates, rollback protection, etc.)

Closing Remarks

- While there are many UEFI BIOS attack vectors, securing SMM must be a high priority
- SMM code developers should adopt the following best practices
  - Anyone writing SMM code must be thinking “how could someone exploit this?”
  - All existing SMM code must be reviewed for exploitable vulnerabilities
  - New SMM code must be peer-reviewed for exploitable vulnerabilities
  - SMM code review must be an integral and ongoing part of the development process

Additional Reading and References

- CanSecWest 2015
  - A new class of vulnerability in SMI Handlers of BIOS/UEFI Firmware
  - Attacks on UEFI Security
  - How many million BIOSes would you like to infect?
- BlackHat 2014
  - Extreme Privilege Escalation on Windows 8/UEFI Systems

Thanks for attending the UEFI Spring Plugfest 2015

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

presented by