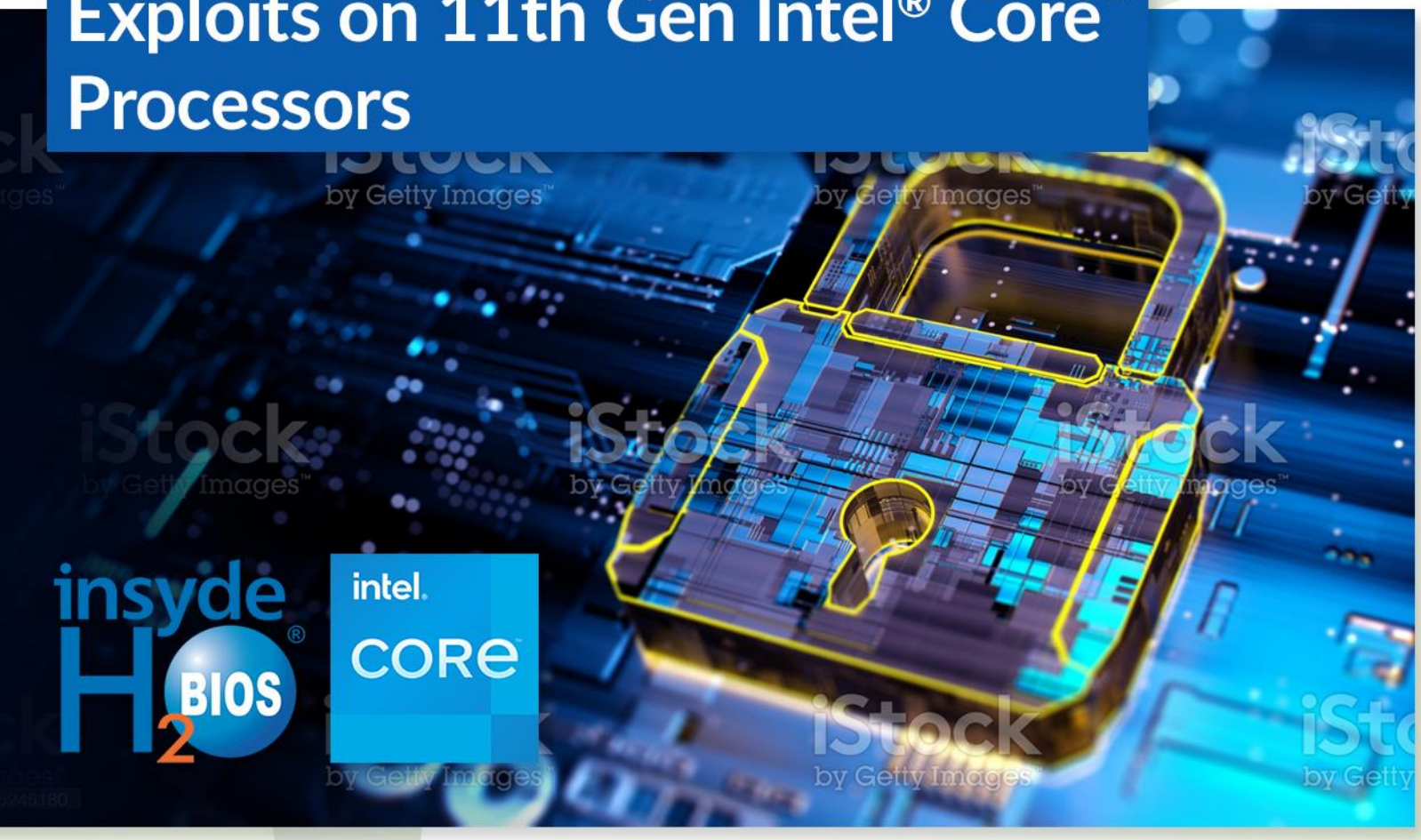
Defense in Depth Against Exploits on 11th Gen Intel® Core™ Processors



Published by: The Office of Security & Trust, Insyde Software
Tim Lewis, CTO

Rev 05.25.2021

intel. partner
Gold
IoT Solutions

Executive Summary

Learn how to leverage the latest security features available from Intel embedded platforms and InsydeH2O UEFI BIOS, while also following the latest stringent industry security standards, to design and deliver more resilient and secure embedded products to market.

Introduction

Firmware is the new frontier for platform security. It is safe to say that if your firmware is compromised, then your platform is compromised. The firmware sets up and maintains the platform's hardware security capabilities and hands off control to the operating system securely. The firmware is integral to the platform and some or all of it is stored in non-volatile storage (like flash) directly attached to the platform. This makes it difficult to find malware embedded in firmware, difficult to mitigate its effects (because most firmware is essential to the boot process) and difficult to remediate.

In this paper, we look at a few ways you can use Insyde Software's UEFI firmware and the Intel® System Resource Defense[1] feature of the Intel Runtime BIOS Protection found in the 11th Gen Intel® Core™ Processor (codename Tiger Lake) when the firmware has been compromised. Either an attacker's unauthorized code is executing in the firmware or authorized code in the firmware is executing badly under the attacker's control. How can the user or IT administrator limit the damage and respond?

First, we'll look at two types of firmware attacks that are a serious threat to computing platforms today.

Second, we'll look at how the defense-in-depth strategy deals with these attacks.

Third, we'll show how 11th Gen Intel Core Processors and Insyde's InsydeH2O firmware are uniquely equipped to help end-users and IT administrators thwart these attacks.

Insyde Software's flagship BIOS, InsydeH2O®, provides production ready firmware that is reliable, secure and highly customizable to allow system makers to fully leverage the 11th Generation Intel Core Processor's increased efficiency, advanced I/O and security features such as Thunderbolt 4, Intel System Resource Defense, Intel Total Memory Encryption and much more.

Types of Firmware Attacks

The runtime security threats to firmware posed by malware fall generally into two categories: injection attacks or confused deputy attacks.

With confused deputy attacks, the attacker uses a weakness in the firmware that already exists in the platform and tricks it into doing something it wasn't designed to do. For example, corrupting or leaking privileged data or calling malware-provided functions. We have seen this several times where firmware code in SMRAM attempts to call a UEFI service function (boot or runtime) outside of SMRAM. Whereas code in SMRAM is protected, it loses the benefits of that protection when it calls outside of SMRAM.

With injection attacks, the attacker injects the malware into the flash device. By injecting it into the flash device, the malware becomes a part of every boot. This makes it hard to detect, since most anti-virus solutions don't effectively scan the firmware. This also makes it hard to eradicate, since fixing the solution requires modification

[1] For more details on this and other features of Intel's Hardware Shield security technologies, see https://cdw-prod.adobecqms.net/content/dam/cdw/on-domain-cdw/brands/intel/white-paper-hardware-shield-2021.pdf

of the flash device. Injection attacks usually rely on misconfigured hardware (such as the LoJack attacks[2] or TrickBoot[3]) or physical access to the platform (such as supply chain attacks). Once the code is in the firmware, it can weaken defenses, leak information or perform privileged tasks.

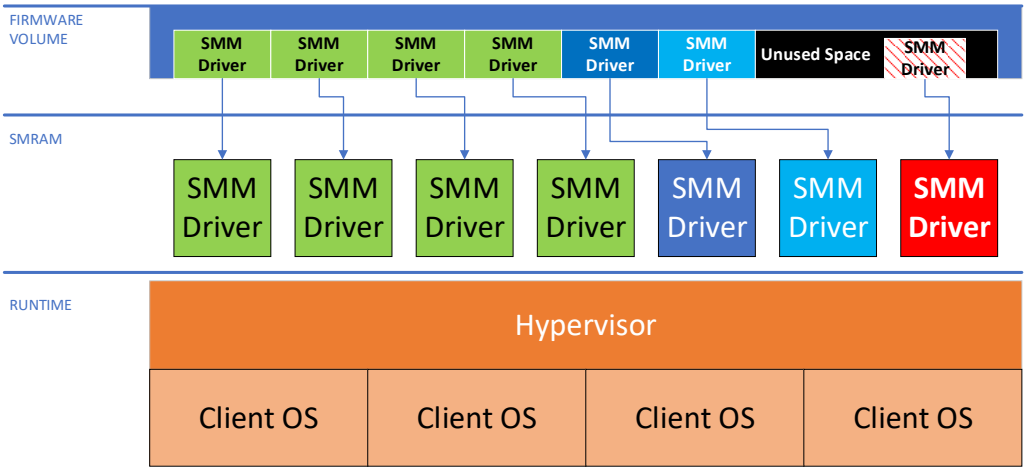


Figure 1 Injection Attack

Defense in Depth

One of the key tenets of security is defense in depth, which asks the question "Assume we have been hacked, now what?" No matter how good the security protections are, at some point, someone will find a weakness and circumvent the protections. Therefore, we need to establish layers of protections so that even if one is bypassed, another will stop or limit the damage.

For example, Intel chipsets have contained protections detecting and limiting runtime code's ability to write to the flash device. For another example, InsydeH2O uses BootGuard on 11th Gen Intel Core Processors to measure the firmware using a signature and a key fused into the chipset.

Similarly, the best practices for firmware have long described how firmware can measure the flash contents to detect tampering. InsydeH2O uses chain-of-trust measurements and TPMs (Trusted Platform Module) for exactly this purpose.

But let's assume the worst, even if there weren't any "mistakes" an injection attack succeeds: somehow the chipset protections have been bypassed (say, through a physically present attacker with a SPI programmer) and the firmware didn't measure correctly (because the code was in a flash region that wasn't supposed to be used).

What then? We need a backup plan.

While injection attacks rely on adding new code to the firmware undetected, confused deputy attacks rely on tricking the existing firmware into leaking information or handing control to the malware with escalated privileges. In some ways, this is easier for the attacker, since it doesn't require modification of the firmware

[2] For an overview of the LoJack-related attacks, see https://www.bleepingcomputer.com/news/security/apt28-hackers-caught-hijacking-legitimate-lojack-software/. For technical details, see https://www.blackhat.com/docs/us-14/materials/us-14-Kamluk-Computrace-Backdoor-Revisited-WP.pdf

[3] For an overview of the TrickBoot-related attacks, see https://eclypsium.com/2020/12/03/trickbot-now-offers-trickboot-persist-brick-profit/

image. In other ways, this is harder for the attacker, since it required familiarity with a specific firmware implementation. Attacks may be limited to a particular platform, vendor or firmware vendor.

Intel Processors have often provided additional security capabilities for operating systems that protect memory from unauthorized access by drivers and applications. Firmware has traditionally not used these capabilities because all firmware was traditionally trusted and certain firmware needs complete access to the hardware resources. InsydeH2O uses page tables to detect NULL pointer accesses and accesses to unallocated heap memory.

But, again, let's assume the worst, even if there no issues detected during development and testing, a firmware module is tricked. Perhaps runtime code is tricked into calling malware.

What then? We need a backup plan.

Any backup plan would assume that weak or malicious code is already in the BIOS. Assuming that, the backup plan needs a way to (a) detect attacks by this code, (b) limit the damage these attacks can do and (c) enable the user or IT administrator to respond to the attacks.

This new model is a shift for the industry. Up until recently, firmware is consider trusted in the same way that other chips on the motherboard are trusted. But now, firmware is viewed by hypervisors, OS vendors and OS application writers with a hint of suspicion. Yes, it is trusted: it is all measured by a hardware root-of-trust solution like BootGuard, the firmware's own chain-of-trust measurement and the TPM. If implemented correctly, we know who authorized that code to be executed.

In order implement the backup plan, defense in depth requires that we separate code into two classes: the normal firmware code that implements the majority of the firmware's functionality and the kernel firmware code that watches the normal code and responds in case of a security breach. The kernel uses both software and hardware to enable security protections, detect security incidents and then report and respond to them.

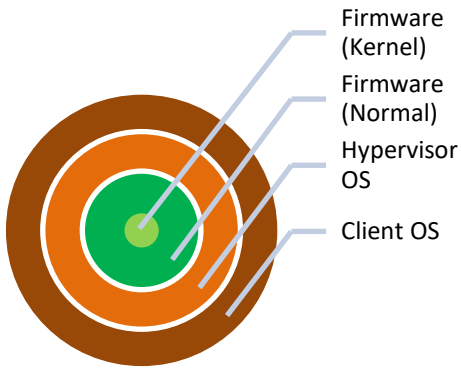


Figure 2 Defense in Depth for Firmware

Firmware Security Resources

We've mentioned a few of these, but the following diagram shows how, at each stage in the firmware's lifetime, InsydeH2O uses these technologies to reduce the likelihood of a successful attack and, when there is an attack, reduce its consequences.

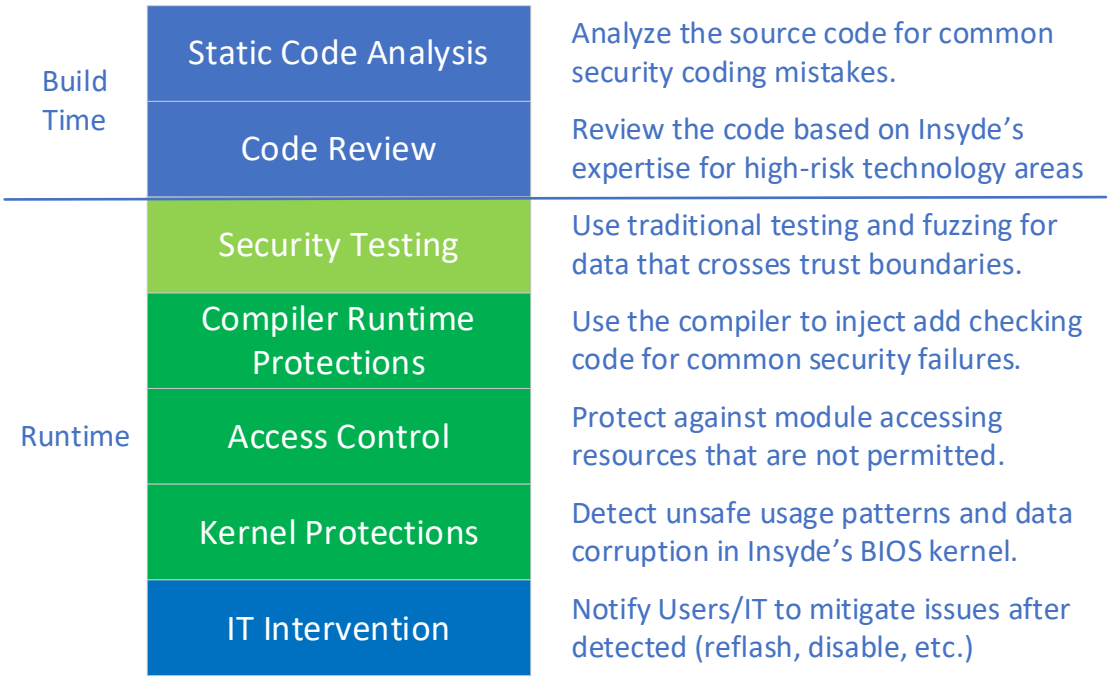


Figure 3 How Do You Secure Firmware?

During the software development process, Insyde uses **static code analysis tools** to check for code patterns that may indicate security weaknesses. Insyde has also instrumented key libraries and APIs in its firmware kernel to give these tools hints as to the expected behavior. This, along with a custom issue database used during analysis, reduces the chances of false positives and raises the productivity of engineers who have to sort through the voluminous technical reports generated by these tools.

Even with good tools, some security weaknesses can only be found through **code review**. This review process includes 15 general security review items and then 70 specific security review items spread across 7 high risk technology areas. These review areas represent the areas where, historically, security weaknesses have been discovered and trust boundary analysis has indicated that attacks are likely.

Once InsydeH2O has been built and reviewed, Insyde tests its code. But secure development requires that Insyde not just verify the functionality, but also its security through **security testing**. Security testing tries to attack the firmware using the corner cases, unexpected sequences of events, strange conditions and weird inputs and verifies that the firmware (hopefully) rebuffed them.

After this point, the platforms are moving out of the labs and off the manufacturing line into the hands of real users and into the sights of malware authors. InsydeH2O has uses **compiler runtime protections** against stack corruption, integer overflows or uninitialized variables.

Compiler protections only help when you have source code. But some of the code is binary-only. The Intel Processors provide **access controls** that generate exceptions or faults upon detecting invalid memory access-writes to code pages, execution of code in data pages, overflows in the stack and heaps and usage of NULL pointer. The 11th Gen Intel Core Processor enhances these capabilities with its Intel® Runtime BIOS Resilience and Intel® System Resources Defense technologies, which we will examine more later.

There are certain conditions that should never occur in InsydeH2O's normal operation. The **kernel protections** can detect these conditions and generate a security event. These protections enhance modules with source code

(like enhanced compiler macros) and modules without source code (like Task Priority Level (TPL) and memory functions).

Finally, InsydeH2O provides a security event framework that reports and responds to security events allowing **IT intervention**. When security events occur, in addition to status codes or POST codes, InsydeH2O reports the event information, allows the system to be taken off-line and reflashed or even contacts the IT administrator through the BMC (Baseboard Management Controller) or other management networks.

11th Gen Intel Core Processors and Intel® System Resource Defense (ISRD)

One of the most sensitive portions of the firmware runtime environment are the drivers that execute in System Management Mode (SMM). SMM code is highly privileged and typically has full access to the platform's resources. Not only does this include hardware and CPU registers, but it also includes all of the memory outside of System Management memory (SMRAM) controlled by the OS kernel, the OS applications, the firmware and (if present) the hypervisor. If SMM code gets injected into the firmware, or the firmware is tricked somehow, the SMM code could reveal sensitive information (like security encryption keys, passwords or keystrokes) or tamper with the OS or hypervisor operation (altering critical decisions related to security (for example). Runtime SMM firmware code is critical to the security posture of the entire system because SMM code could theoretically do so much damage.

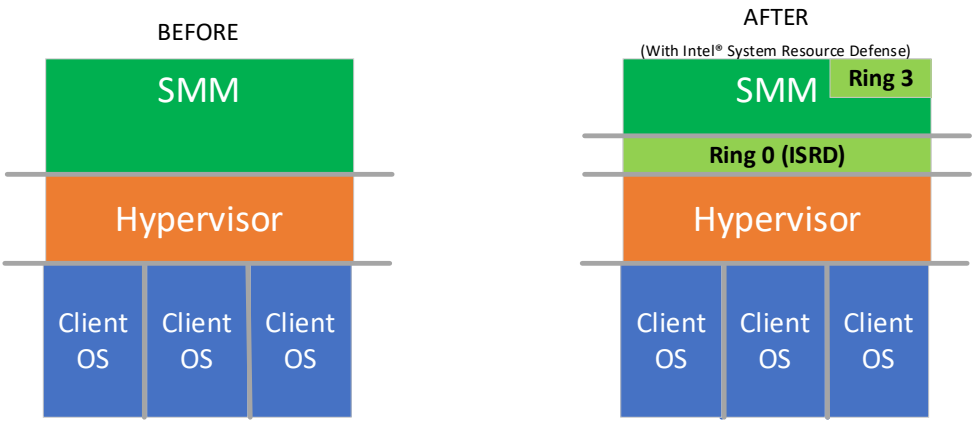


Figure 4 Runtime Firmware Security Boundaries

In the traditional security model, SMM tries to be protected from the hypervisor or client OS. The hypervisor OS tries to make sure that the client OS can't tamper with the hypervisor's code or data. And the client OSes are looking suspiciously at each other.

In the revised security model, all of those things are still true. But now the client OS is worried about what the hypervisor sees and can do, and both the hypervisor and the client OS are worried about what SMM can see and do. Not because the SMM code is "bad" but because it might have been hijacked or exploited and they wouldn't know.

ISRD adopts the firmware "kernel" concept and applies it to SMM. A very small portion of the SMM code, positioned at the System Management Interrupt (SMI) entry point, runs at ring 0 (supervisor mode). Then, before handing control to the SMI handling described in the UEFI specification, it transitions to ring 3 (user mode). This transition, along with new chipset registers and hardware MSRs, enables protections that prevent access to CPU MSRs, critical I/O locations, page tables and all of the runtime memory used by the operating system.

This ring 0 SMM entry point code controls:

Paging – All of the code and data pages in SMRAM is mapped appropriately for code (RO/NX) or data (RO/NX or RW/NX) Th page table itself is marked as RO/NX. MMIO is mapped as RW/NX. All non-SMRAM pages are marked NX, firmware communication buffers marked Reserved or ACPI NVS are marked RW/NX and all OS memory is not present. This will generate an exception if any access by the ring 3 (normal) SMM code violates these restrictions.

MSRs – Access to MSRs is controlled using a bitmap originally found in Intel's Virtualization Technology MSR Bitmap. When MSRs are accessed from ring 3, this bitmap determines whether the MSR access is allowed or generates an exception to ring 0. In ring 0, there is a further determination made as to whether the MSR access should be allowed and, if not, a hardware violation is reported.

I/O – Access to I/O is controlled by the I/O bitmap found in the TSS for the ring 3 code. Attempts to access a prohibited I/O location generates an exception.

Save State – The CPU save state, which is a memory buffer filled with each CPU core's register contents when entering SMM, is also protected by paging. Access to the save state prohibited by ISRD so that the SMM code cannot change the CPU register contents of a thread. SMIs can be invoked synchronously (a software SMI) or asynchronously (due to a hardware event), so hacked SMM code could change CPU registers contents or read CPU register contents at a critical moment. ISRD limits the possible damage by generating an exception when the ring 3 code attempts to change the save state contents. [4]

Other Registers – The OS context in all floating point, SSE, AVX and related registers is scrubbed before control is given to the ring 3 code. This prevents hacked ring 3 code from reading anything about the OS or its applications from inside of SMM. All BIOS-accessible register backs are scrubbed and saved before entering ring 3 and then restored before returning to the OS.

The information about the security configuration of the SMM code allows the hypervisor or operating system to decide how they want to handle user data and security. Along with ISRD, the 11th Gen Intel Core Processor generates the Intel® System Security Report (ISSR) which can be verified by a TXT-launched MLE and used by the operating system. SINIT attested code in the Platform Properties Assessment Module produces a combined hash of the SMM entry point code, the SMM entry point information table and the ring 0 exception handler. By validating this, the OS can understand whether specific ISRD capabilities have been enabled.

InsydeH2O and Insyde's Boot Firmware Threat Defense™

What should the firmware do when there is a security event detected? Security events might happen for various reasons:

1. Compiler-inserted runtime check failure or unhandled C++ exceptions.
2. Kernel-detected heap corruption, TPL inversion or stack overflow.
3. Developer-inserted assertions to catch never-should-occur behavior.
4. OEM-inserted platform-specific events.

[4] ISRD makes the EFI_MM_CPU_PROTOCOL/EFI_SMM_CPU_PROTOCOL unusable for most SMM code. ISRD still allows access to the IO_MISC and RAX registers. These are allowed because typical software SMI processing refers to the RAX register contents on the CPU that generated the software SMI, which is detected by examining the IO_MISC save state location. ISRD does allow this feature to be turned off for compatibility with existing SMM implementations.

5. CPU page fault or general-purpose exceptions. Most of the protections provided by ISRD fall into this category.

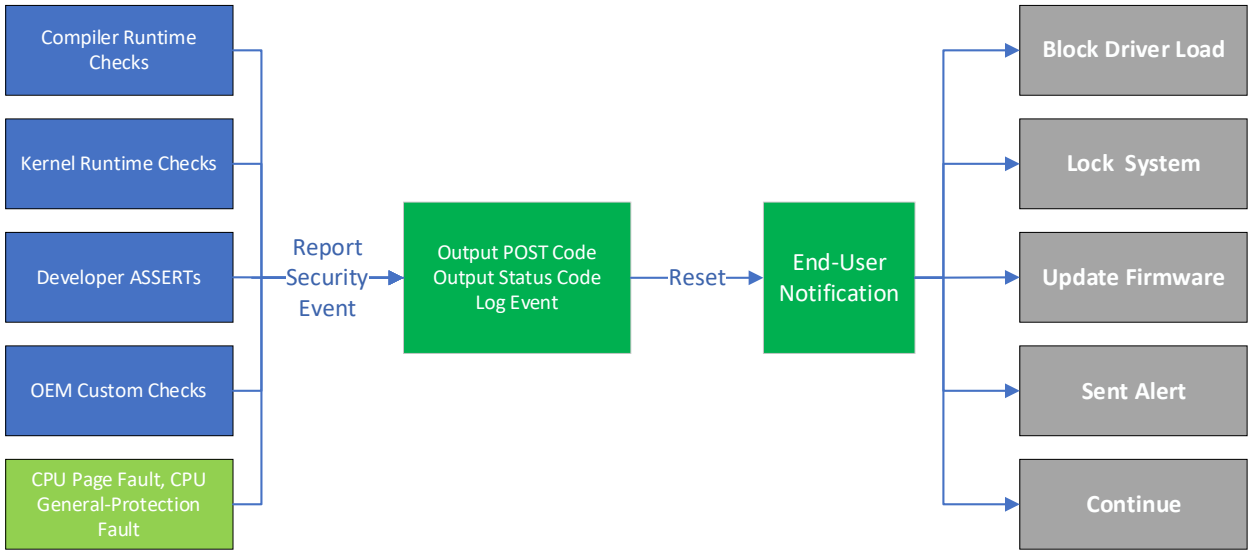


Figure 5 InsydeH2O Boot Firmware Threat Defense

While an operating system offers a wealth of options to deal with components that exhibit risky behavior, the options in the firmware environment are limited. Usually, the security event might output a status code and reset the system or, if you are lucky, log the event. This does protect the platform, because the offending code is no longer running, but it may actually hinder recovery. When a platform resets, the user doesn't know why there was a reset. It is likely that the same vulnerability is present, but the user doesn't know that it was a vulnerability or how to fix it.

However, in InsydeH2O each of these is treated as a security event that flows into its infrastructure to inform the end-user or IT department that a security event occurred and give options as to how the security event should be handled, either automatically or with user intervention.

In InsydeH2O, when the security event is detected, there is a unique POST code and status code output and detailed data about the event is logged. For example, for ISRD page table faults, the type of fault is recorded, along with the address of the code that caused the fault and the address it was trying to access. By using the code address, it is possible to determine the name and line of code that caused the exception.

Then, InsydeH2O will either shut down or reset the system to start again with a clean system. During early power-on, the logged events cause the system to enter remediation, where the event is sent to the BMC, to an enterprise management application or to the end-user to determine the type of mitigation to apply. Mitigations include actions like:

- Block the driver that caused the security event. Since the security event handler was able to determine which module generated the security event, this module can be blocked on the next bot using UEFI's security architecture protocol.
- Lock the system. This takes the system off-line until an administrator can unlock it, preventing it from performing any actions that might affect the rest of a corporate network. This might require remote activation or someone who knows the administrator password.

- Update the firmware. The firmware can check specified websites or servers for signed updates to the firmware that have a higher version than the currently running firmware. The updated firmware might have a fix for the security issue.
- Sent an alert. The firmware may have a connection to a BMC or other remote management service and can send an alert containing the log information to tell the IT administrator that the system has a security issue.
- Continue. Of course, the end user could ignore the event for the moment and reach the OS environment where other mitigation tools may be available.

These actions might be selected by the end-user or applied automatically based on policy.

These actions make the end-user aware that there is a security situation and gives them user control over the next steps. When Insyde's InsydeH2O Security Event architecture is combined with the 11th Gen Intel Core Processor's IRSD feature, the platform is protected, and the user is empowered to respond.

Conclusion

Firmware threats are growing as malware authors find new targets among those platforms vendors who are not aware or insufficiently prepared. Injection attacks are a signal to the industry that the enemy they are defending against may already be inside. Confused deputy attacks are worth watching because they take advantage of an unintended side effects of a product's otherwise useful features.

Intel System Resource Defense on 11th Gen Intel Core Processors and InsydeH2O's security event architecture give you defense-in-depth against runtime SMM attacks. Attackers are frustrated when, after they successfully inject or successfully confuse, these technologies protect against damage and actively aid in recovery.

About Insyde Software

Insyde Software (www.insyde.com) is a leading worldwide provider of UEFI firmware, systems management solutions and custom engineering services for companies in the mobile, server, desktop and IoT (Internet-of-Things) computing industries. The company is publicly held (GTSM: 6231) and headquartered in Taipei, Taiwan with U.S. headquarters in Westborough, MA. The company's customers include the world's leading computing, communications and storage device designers and manufacturers. Insyde Software is a Gold Member of the Intel® Partner Alliance.

Disclaimer

Insyde Software provides this document without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

This document could contain technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in future revisions of this document. Insyde Software is under no obligation to notify any person of the changes.

The following trademarks are used in this document:

Insyde and InsydeH2O are registered trademarks and Boot Firmware Thread Defense is a trademark of Insyde Software. Intel and Intel Core are registered trademarks or trademarks of Intel Corporation. All other trademarks or trade names are property of their respective holders.