SAMSUNG

Technical White Paper

Galaxy Book4: Security Below the OS

Table of Contents

Introduction

This document provides an overview of the **Samsung Galaxy Book4 Secured by Knox** laptop PC, focusing on the unique advantages that differentiate it from other options in the PC market from a security standpoint. The contents of this white paper are designed for C-level executives and IT security professionals looking to evaluate the Galaxy Book4 Secured by Knox as a comprehensive enterprise laptop PC solution to deploy to their workforce.

Please note that this white paper specifically describes the Samsung Galaxy Book4 Secured By Knox model. The security features described here don't necessarily reflect the security features available on other Samsung Galaxy Book laptop models.

Protecting below the OS

Laptops can be particularly challenging for IT admins to protect since the BIOS — the software that initializes the computer's hardware components and starts the operating system (OS) — is often the target of attackers. The BIOS needs to be particularly secure from threats to its **data** and **code**, as these are vulnerable to hackers. The following vulnerabilities are present if the BIOS isn't properly protected:

Data compromise

A data compromise occurs when an attacker gains access to protected information like passwords, certificates, or encryption keys. During a normal boot sequence, the BIOS relies on the **Trusted Platform Module (TPM)** to encrypt sensitive data and fend off hacking attempts on the hardware. Windows features like **BitLocker** then uses the TPM to ensure that protected data is only available when trusted firmware has been loaded. If the BIOS is compromised, it may provide false information to the TPM and trick it into loading a malicious OS, thus giving an attacker backdoor access to protected enterprise data.

Code compromises

The integrity of the OS depends on the integrity of the BIOS, since it is ultimately trusted for loading and executing the code that starts the OS. If the BIOS is compromised, then the self-check code Windows uses to detect threats may also be compromised. Without properly functioning update-validation systems, the BIOS (which normally validates update packages) can update itself with malicious code, granting backdoor access to hackers and preventing itself from being removed. This could lead to the entire device being compromised unless an IT Admin intervenes.

Impacts of BIOS compromise

If a BIOS is compromised, the following may happen to either its data or code:

- **Tampered UEFI data** — The BIOS maintains most of its boot configuration data in UEFI variables stored in onboard flash memory. If a malicious root or admin process modifies one of these variables, then security-critical features such as Secure Boot can be disabled, thus letting the device potentially boot untrusted or malicious software.

- **Corrupted firmware** — BIOS code is also stored in flash memory, which may be accessible to compromised OS kernels or physical attackers looking to gain privilege or persistency. In the last several years, there have been multiple instances of malware adding malicious code to the BIOS which then caused malicious code to be run on the host OS.

- **Exposed enterprise network** — Physical attacks can disable Secure Boot settings or circumvent the TPM. This could result in the loading of unapproved OS images, and exfiltration of sensitive data. If an attacker does manage to boot a malicious OS image, they may abuse VPN policies to gain access to a corporate network.

Although these problems are persistent, they aren't new to IT security professionals. Various technologies have been developed to help mitigate these problems. For example, **UEFI** introduced standards to the Secure Boot process to ensure that code only loads after its been verified. Other technologies like **Intel Boot Guard** extend the Secure Boot process by ensuring code verification begins with the CPU hardware itself. Though these technologies help to mitigate some of the BIOS compromise issues mentioned above, they also present several limitations. For example:

- While the UEFI Secure Boot process validates firmware code, it doesn't validate UEFI configuration data. This can lead to users and software processes making modifications to corrupt the BIOS settings. Once an attacker has access to BIOS settings, they could remove Secure Boot features altogether to allow the loading of malicious firmware code.

- UEFI is limited in the response options it provides when it detects security issues. This can be a problem in situations where the device can't silently handle security issues by itself. For example, if a compromise is detected at boot time, the user or admin may need to trigger incident response processes, which might require unified and flexible response mechanisms out-of-scope from UEFI.

- The existing Secure Boot process is focused solely on the *detection* and *prevention* of unsigned or revoked firmware being loaded. During a normal boot sequence, the BIOS is loaded in several stages, with each stage verifying the authenticity of the next. If a tampered boot stage is detected anywhere in the process, the BIOS would simply stop booting, and likely require administrator intervention – costing valuable time and resources – to be repaired.

With all these limitations in place, IT admins and executives need a more secure and robust solution to protect their enterprise from the growing number of threats below the OS. A solution that guarantees data and code protection before the OS ever boots up. This is why Samsung built the Galaxy Book4 Secured by Knox; our most secure and tamper-resistant laptop to date.

Introducing Samsung Galaxy Book4 Secured by Knox

The Samsung Galaxy Book4 Secured by Knox was designed to provide best-in-class hardware-backed security to meet the needs of enterprise users. It pairs industry-wide standards with powerful OEM features to provide a high level of firmware protection and robust firmware recovery capabilities.

Every Galaxy Book4 Secured by Knox comes equipped with our **Secure Embedded Processor (SecEP)**, a standalone processor that validates firmware *before* the CPU ever powers up. This processor, along with its memory (RAM) and dedicated storage unit, are all isolated from the CPU in order to protect against compromises to the OS, hypervisor, and BIOS.

While the BIOS takes advantage of industry-standard security features like Intel Boot Guard and UEFI Secure Boot, the SecEP brings additional robust security features to help further protect the firmware. For example:

- Intel Boot Guard ensures that only an OEM-approved BIOS may run on the CPU. The SecEP takes security one step further by ensuring that only approved versions of the BIOS may modify any boot configuration data, and that a backup copy of the BIOS firmware is always available even if the original is corrupted.
- UEFI ensures that all firmware modules are verified by hash and validated against a binary revocation list. This means that only privileged processes have the ability to modify boot configuration data. The SecEP further enhances this process by making the revocation lists and boot configuration data tamper-resilient. It works with the BIOS to ensure that critical boot parameters only get modified by an authenticated user or admin through the BIOS setup utility. If UEFI detects an attempt to load suspicious firmware, the SecEP's **Tamper Alert** feature can be configured to require either user or admin confirmation before booting.

By introducing the SecEP, admins gain more awareness and control of common security compromises. Critical features like auto-recovery, data protection, and tamper alert are deeply integrated with the BIOS, but also completely isolated from the OS. This helps to ensure that an attack on the BIOS gets handled *before* firmware ever gets loaded.

Throughout the rest of this document, we'll describe the various hardware-backed security features and enhancements introduced in the Samsung Galaxy Book4 Secured by Knox. Join us, as we showcase our best-in-class, most secure and protected laptop PC yet.

Hardware-backed trust

In typical laptop PCs, the motherboard normally provides power directly to the CPU during the boot sequence. Once powered up, the CPU would then access its dedicated SPI flash storage to load and run the BIOS.

With the Samsung Galaxy Book4 Secured by Knox, the boot sequence begins with the SecEP. When the motherboard first powers up, it initializes the SecEP. The SecEP is responsible for providing power to the CPU. Before the SecEP powers on the CPU, it first verifies whether or not the BIOS is corrupted. If the BIOS is safe and corruption-free, the SecEP then powers on the CPU, and the CPU accesses SPI flash storage (to load the BIOS).

These layers of protection are put in place to prevent the device from loading and executing corrupt firmware and compromising the system. **Figure 2** further describes how the SecEP protects the CPU during the boot sequence.

Throughout this chapter, we'll explain the various security features used by the SecEP to provide a robust, hardware-backed system of trust to help protect the Samsung Galaxy Book4 Secured by Knox from threats and compromises.

SecEP FW self-validation

With Galaxy Book4 Secured by Knox, the SecEP prevents the CPU from accessing the SPI Flash Storage until permission is granted. However, before that permission is even granted the SecEP goes through a series of self-validation checks to ensure that its own firmware hasn't been compromised.

Figure 1 describes how the SecEP performs this FW self-validation:

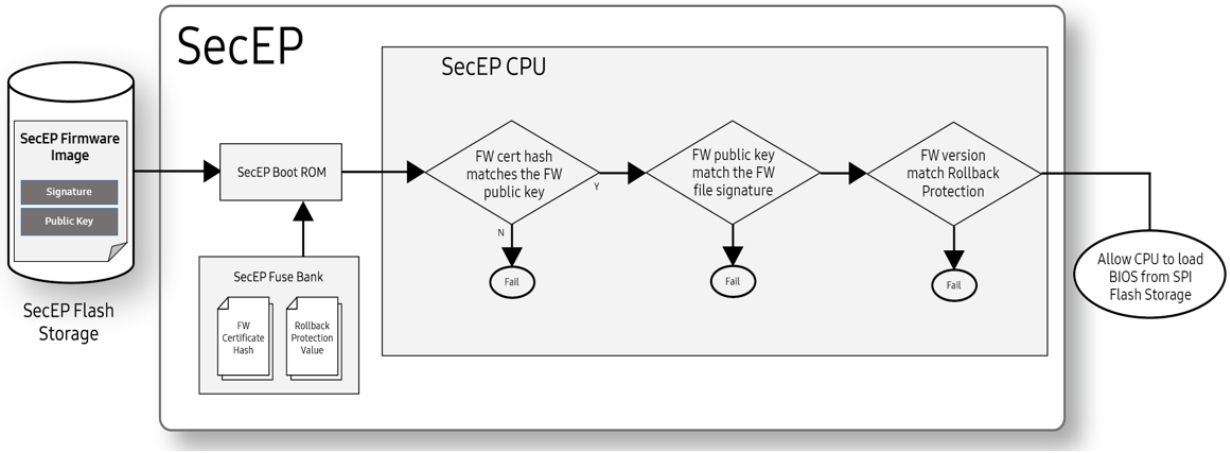

Figure 1: SecEP FW self-validation

- The SecEP FW self-validation first begins with a firmware hash check. This happens when the **SecEP Boot ROM** retrieves the **FW Certificate Hash** (SHA384) – generated using an EC SecP384r1 algorithm – from the **SecEP Fuse Bank**. The Boot ROM then uses this hash to validate the **Public Key** stored in the SecEP flash storage.
- Once the FW Cert Public Key is verified, the Boot ROM uses the **Public Key** to verify the integrity of the **SecEP Firmware Image** stored in flash storage.
- After the **SecEP Firmware Image** gets verified, the SecEP Boot ROM then verifies whether or not the firmware version number is valid. It does this by comparing the firmware version against the **Rollback Protection Value** stored in the fuse bank. Whenever the SecEP firmware gets updated, the fuses corresponding to the *previous* version get set to indicate that it is no longer valid. Once a Rollback Protection fuse is set, it can't be unset.
- If the SecEP firmware version number is valid, the Boot ROM loads and executes the SecEP firmware.

SecEP BIOS validation and CPU initialization

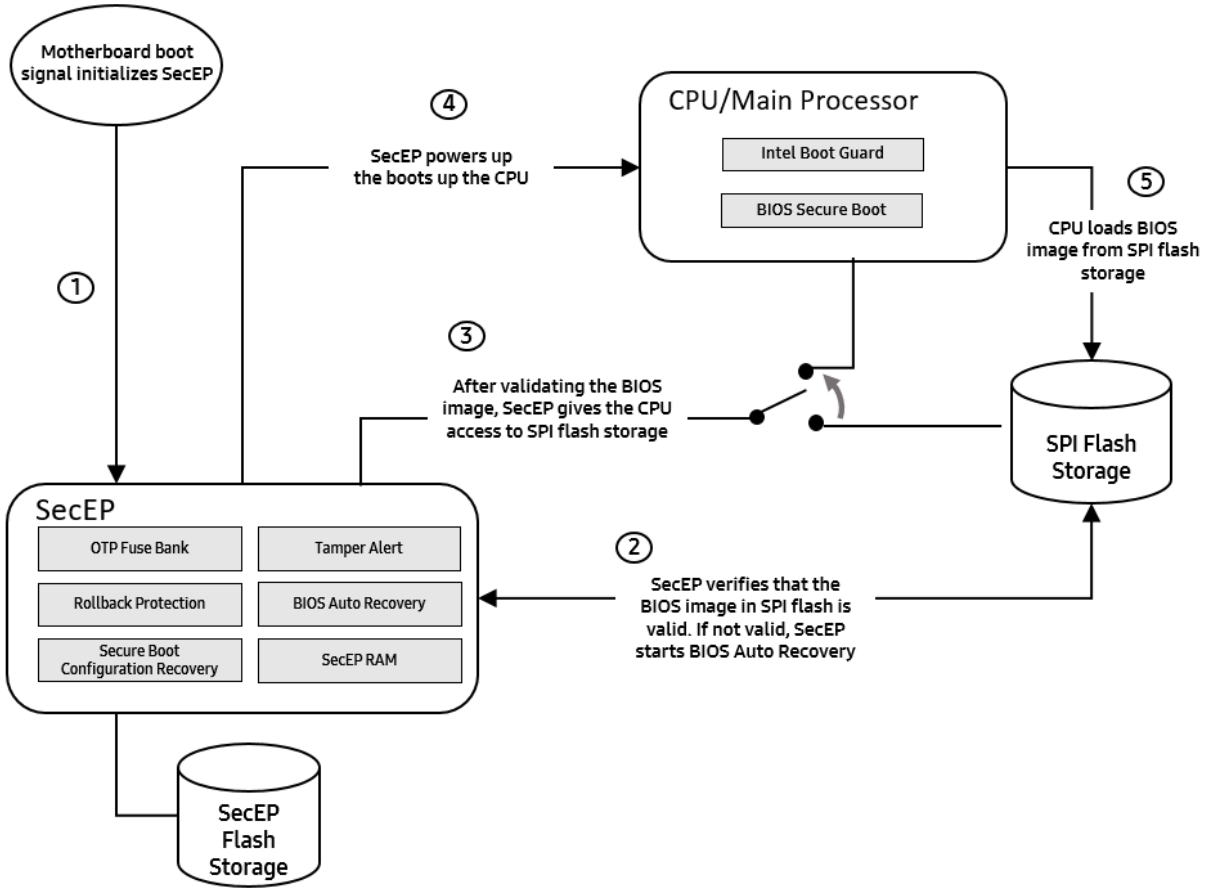

Figure 2: SecEP protection during boot up

Once the SecEP validates and runs its own firmware, it then proceeds to verify the main BIOS. In order to verify the BIOS, the following sequence of events occurs:

- First the SecEP reads the BIOS RSA2048 OEM public key from SPI flash storage as illustrated in **figure 2, step 2**. This key is checked against a SHA384 hash stored in the SecEP's fuse bank. If the key is valid, the SecEP then checks the BIOS version number against the Rollback Protection Value stored in the SecEP fuse bank.
- Once both the BIOS public key and version number are validated, the SecEP gives the CPU access to SPI flash storage in order to load the BIOS, as illustrated in **figure 2, step 3**.
- The SecEP then powers on the CPU. Once CPU has power, it can load the BIOS from SPI Flash Storage, as illustrated in **figure 2, steps 4, and 5**.

If the SecEP detects any incorrectly-signed firmware in place of the BIOS, the SecEP will not grant the CPU access to the firmware, and the CPU will not boot.

After the CPU is powered on, **Intel Boot Guard** independently validates and runs the BIOS Bootloader, which begins the standard UEFI boot process. As part of UEFI, the BIOS Bootloader contains hardcoded **SHA256** hashes for every firmware module that loads. Since these hashes are hardcoded, they are validated by the same signature checks used to validate the BIOS Bootloader. When loading any firmware module, UEFI verifies that the hash of each loaded module is included in the hardcoded table.

BIOS Auto Recovery

During a normal boot flow, the SecEP depends on the CPU's flash storage to load the BIOS firmware. This firmware, while cryptographically verified before being loaded, is still susceptible to attacks from the CPU. If the firmware in flash storage gets corrupted, SecEP will prevent it from loading. While this could mitigate the threat, the device might still be unable to boot up (thus rendering it unusable) without IT admin intervention.

BIOS Auto Recovery addresses this problem by including a back-up copy of the BIOS firmware in the SecEP's flash storage. This storage is physically inaccessible from the CPU and other hardware running on the device. If the SecEP fails to verify the BIOS firmware stored on the CPU's flash storage, it overwrites that firmware with its own locally-stored copy. Once the BIOS firmware has been restored, the SecEP reboots the device so that the recovered firmware can once again be verified and loaded.

Secure Communications with the BIOS

Once the BIOS has been initialized, the CPU and the SecEP communicate with one another through a dedicated line called the **System Management BUS (SMBUS).** This line helps ensure that only messages using an authentication key are passed back and forth between these two components.

Since the SMBUS is connected to the CPU, the OS also sends messages to the SecEP, opening up the possibility for a compromised OS to impersonate the BIOS and execute privileged operations with the SecEP. To help protect against this type of compromise, the SecEP and BIOS negotiate a per-boot HMAC-SHA256 authentication key using ECDH-P256 during the Pre-EFI Initialization Phase.

This shared key gets maintained by the BIOS as it progresses through its boot stages and enters System Management Mode. Once in this mode, the shared key is hardware-isolated from the OS in a special area of main system RAM dedicated to system management data.

The early key negotiation, combined with the boot-time-guarantees provided by the SecEP, ensure that only the SecEP and the BIOS have access to this authentication key. This enables the BIOS to use the SecEP to secure its data, and the SecEP to restrict data access to only the BIOS.

Secured Data

The SecEP stores all of its data on a dedicated flash storage, which only it has access to. This isolated storage, along with the authenticated communication, enables the SecEP to protect its own data against compromises in the CPU's OS.

The SecEP further protects its data by associating it with a HMAC-SHA256 message authentication key. Each HMAC-SHA256 key is derived using a data identifier and a Master Storage Key unique to each device.

The isolated storage enables the SecEP to protect its data from other software running on the device, and the additional encryption enables the SecEP to protect its data from hardware tampering. This lets the SecEP provide stronger security for the BIOS than the BIOS is capable of providing for itself.

Enhancements to UEFI

The SecEP ensures that the CPU only executes a trusted BIOS, and the BIOS relies on the SecEP to protect security-critical data — this is the foundation that Samsung Galaxy Book4 Secured by Knox builds upon to provide best-in-class laptop PC security. In this chapter, we describe how this foundation is used improve on the standard UEFI boot process.

Secure Boot Configuration Recovery

The security of the UEFI boot process depends on the integrity of both the firmware and its configuration data. Secure boot configuration data is particularly security-sensitive, since it forms the basis to make decisions such as:

- Whether Secure Boot is enabled.
- Which keys to use for Secure Boot.
- Which OS images have been revoked.

Therefore, protection of Secure Boot configuration data is critical to establish the root of trust. In addition, these Secure Boot settings need to be recovered from a "known-good" backup if corruption is detected.

The section on Hardware-backed Trust details how UEFI code is protected and auto-recovered from a backup if corruption is detected. However, the same techniques for code cannot be applied for configuration data, since configuration data can be modified after the BIOS signatures are generated. Thus, it is not possible to generate a signature for configuration data during compilation, *and* validate it using the same UEFI code validation flow.

Samsung's **Secure Boot Configuration Recovery** addresses this shortcoming.

As part of the UEFI standard, security-critical boot configuration data is written to flash as **UEFI Variables**. UEFI Variables are stored as key-value pairs, where each key consists of a **Globally-Unique Identifier (GUID)** and a variable name. Boot configuration data is typically modified by an administrator through the BIOS menu. These "known-good" values are backed up in SecEP flash storage.

An attacker can attempt to subvert Secure Boot by modifying or corrupting on-disk variables with inexpensive flash writing tools. For example, they can disable Secure Boot entirely, or change the public keys used to verify signatures. The Secure Boot Configuration Recovery feature detects such out-of-band writes and also recovers these variables to the previously backed up values.

Detecting and Recovering from Unauthorized Offline Writes to Secure Boot Configuration

First, integrity information for the Secure Boot UEFI variables is stored on the SecEP's flash storage, protecting and isolating it from software and hardware-based tampering. This information is cryptographically-protected using the **Master Storage** key kept within the SecEP. This key authenticates the variable's content, name, and GUID.

During boot, the SecEP provides integrity information to the BIOS. The BIOS uses this information to validate the Secure Boot UEFI variables. If an integrity corruption is detected, the BIOS automatically discards any corrupted values, and reverts all protected variables back to their "known-good" values. Once the SecEP completes the recovery, the user is notified through a BIOS notification before the boot process is continued.

Tamper Alert

The **Tamper Alert** feature alerts administrators of certain security-critical events that indicate tampering with security-critical data or code in any layer of the software stack. Specifically, this feature deals with tampering that the system cannot automatically recover from. Such events typically require administrator action to restore the system to a good state. Tamper Alerts are presented to the administrator in a protected pre-boot environment. This feature can also require administrators to acknowledge alerts by entering their BIOS password before allowing the system to boot up.

Tamper tracking is handled through the SecEP's **Tamper Flag**, an indicator used to determine if there were any new policy violations on the device. The Tamper Flag is stored on the SecEP's flash storage, isolating and protecting it from any compromises on the OS.

Authorized software, including the BIOS and SMM (System Management Mode), can request the SecEP to turn on the Tamper Flag when they detect tampering, such as during the following scenarios:

- When unauthorized writes to protected, read-only SMM regions are detected. This is typically indicative of an exploit attempt within SMM.
- When security-critical OS components, such as the event logging service are detected to have been uninstalled. This is typically indicative of malware.
- When invalid policies for SMI call protection are detected. This is typically indicative of attempts to tamper with the system's security configuration.
- When the Windows Platform Binary Table is disabled. This is typically indicative of malware.

Tamper Alert can be configured to require either **admin** or **user** confirmation whenever a violation occurs, before the BIOS starts the OS. During system start up, the BIOS Bootloader reads the Tamper Flag state from the SecEP. If this flag indicates that corruption is detected, the following occurs:

- If admin confirmation is required, then the admin BIOS password must be entered to proceed with boot, or
- If user confirmation is required, then the user can acknowledge the alert by choosing to continue with the boot process.

In both cases, the device will continue to display a message indicating that a corruption was detected during system boot, until the Tamper Flag is cleared.

By providing a consistent method for different components to report a corruption, Tamper Alert ensures that IT admins and device users are made aware of compromises when they happen, thus providing enterprises an opportunity to trigger a response more quickly and effectively.

Protecting the SMM

The UEFI/BIOS installs code that remains persistent through the system's runtime in a mode called **System Management Mode (SMM)** to maintain control of the system even after system boot. SMM code handles a variety of low-level, system-wide tasks such as power management, read/write of persistent BIOS variables and configuration, and also contains any OEM-specific code.

SMM is the highest privilege mode of the processor. Like the BIOS/UEFI, SMM code runs below the OS. SMM has full access to physical memory, SMM-specific memory called SMRAM, MSR special-purpose registers, the SPI flash region to read and write BIOS variables, and I/O operations. Furthermore, the SMM is designed to be invisible to lower privilege layers such as the OS kernel or hypervisor.

A compromise of the SMM has extremely high impact on security since the SMM is the highest privileged mode and invisible to lower privileged layers. For example, an attacker that compromises the SMM can write to the SPI flash to persist stealthy malware code, disable BIOS variables such as Secure Boot, and change system configuration. Moreover, the attacker can completely conceal such activities from any OS or hypervisor detection techniques like anti-virus and bypass mitigations.

How SMM Exploitation Works

Attackers typically escalate privileges to the SMM by exploiting vulnerabilities in the SMM code. The OS calls SMM code through **System Management Interrupts (SMI)**, and passes parameters to SMI handlers using a shared memory area called the SMM Communication Buffer.

If the attacker finds a vulnerability such as memory corruption in any of the SMI handlers, they can craft an input in the SMM Communication Buffer to exploit vulnerable code in the SMI handler. An attacker typically uses well-known exploitation techniques like **Return-Oriented Programming (ROP)** to escalate to SMM privileges, disable SPI write protections, and persist malware.

Samsung Galaxy Book4 Secured by Knox has several features dedicated to detecting, hardening, and mitigating the effect of SMM exploits. These features are described below.

SMI Guard: Protecting SMI Calls

The SMI Guard is a security routine that intercepts SMI calls and analyzes the content of the shared buffer to detect malicious inputs or known exploit patterns based on rules, as shown in **Figure 3**.

Once an attack is detected, remedial actions include blocking the SMM call from going through, and alerting an admin.

This feature enables IT admins to define the following rules to limit calls to the SMM at runtime:

- **Attack surface reduction** – This type of rule blocks SMIs that are not required at run-time, including blocking SMI setup function calls and security-critical BIOS variable updates that legitimately happen only during system initialization at boot-time.
- **Exploit detection –** This type of rule prevents attacks on specific SMI handler calls by detecting exploit patterns and comparing SMI call argument values against known-bad values. If a known-bad argument value is detected, SMI Guard can block such known-bad calls to the vulnerable SMI handler until a firmware patch is deployed.

The Exploit detection feature is used for rapid incident response against newly discovered SMM vulnerabilities until the enterprise deploys a fully patched BIOS version. Enterprises may intentionally delay BIOS updates for stability and compliance purposes. In such scenarios, the SMI Guard minimizes the period of time a device is vulnerable by detecting and blocking exploit attempts on the firmware.

The typical rapid incident response flow is as follows (**Figure 4**).

1. A new SMM vulnerability is reported (Step 1).

2. If the vulnerability affects Samsung Galaxy Book4 Secured by Knox and can be blocked by SMI Guard, Samsung rapidly releases SMI Guard rules to block exploitation on vulnerable firmware versions (Step 2).

3. SMI Guard rules are deployed along with a notification (Step 3). The Samsung Galaxy Book4 Secured by Knox is now protected from attempts to exploit the vulnerability.

4. Finally, when an updated BIOS firmware version that fixes the vulnerability is deployed, the SMI Guard rule is automatically disabled.

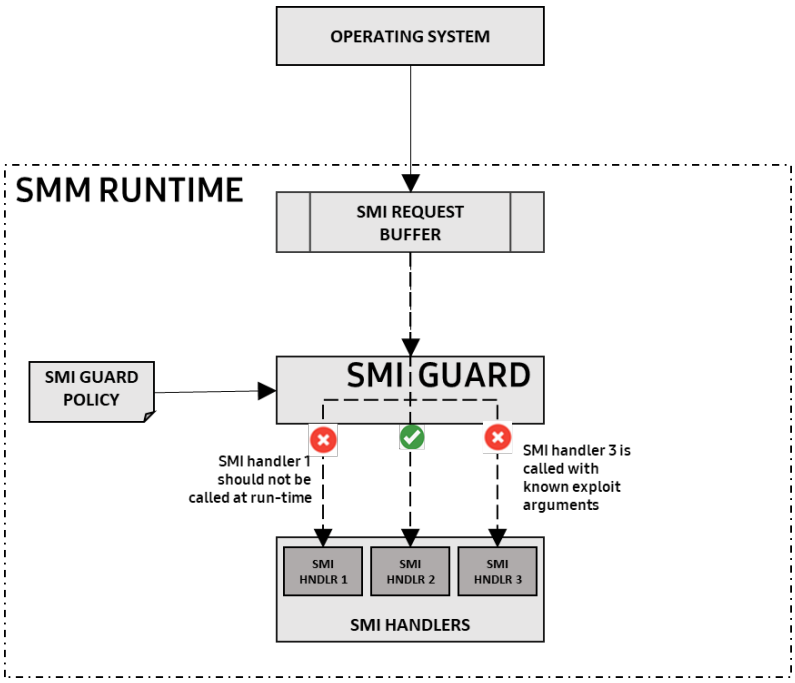

Figure 3: SMI Guard blocks SMI calls that should not be called at run-time and SMI calls with known-bad arguments indicative of exploits

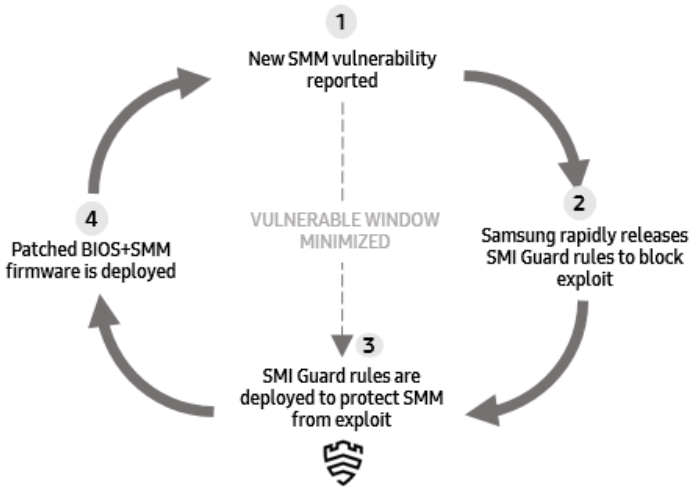

Figure 4: SMI Guard enables rapid incident response and minimizes the window of vulnerability

Advanced SMM Protection

This feature adds a layer of exploit mitigation to protect SMM even in the face of previously unknown vulnerabilities and zero-day attacks.

While chip firmware manufacturers and BIOS vendors take special efforts to ensure that SMM code has no vulnerabilities, given the significant volume of SMM code, attackers are sometimes able to discover previously unknown vulnerabilities (also called "zero days" vulnerabilities). Advanced SMM Protection aims to block the attacker from taking advantage of such these previously unknown vulnerabilities.

Leveraging Existing SMM Exploit Mitigations

Samsung BIOSes have basic exploit mitigations, such as stack cookies, compiled into them. In addition, Intel introduced two security features aimed at making the SMM runtime environment more resilient to attackers, namely **Intel Runtime BIOS Resilience** and **Intel System Resources Defense**, both of which are enabled on Samsung PCs.

The goal of these features is to make it more difficult for an attacker with access to a zero-day to perform anything useful, such as leaking SMM memory or modifying the privileged processor state.

Intel Runtime BIOS Resilience protects page tables by locking them so they cannot be modified later, and enables strong defense mechanisms such as read-only code sections and non-executable data sections.

Intel System Resources Defense further locks-down the SMM runtime environment by running SMI handlers at ring 3 (lower privilege) instead of ring 0 (higher privilege). By forcing all SMI handlers to run at a lower privilege, any attempts to access hardware such as MMIO or MSR's by an SMI handler will trigger a fault and transfer control to a trusted ring-0 component within SMM called the Policy-shim.

The Policy-shim has the power to allow/disallow any resource access, allowing OEM's like Samsung to tightly control which system resources the SMI handlers can access. This makes it much more difficult for a potential attacker to force one of our SMI handlers to perform a malicious action and further compromise the platform.

Introducing Novel Mitigations

While the above mitigations significantly reduce the attack surface and impact of an SMM exploit, they all focus on protecting code but not data. Security-critical data structures in SMM are still vulnerable to unauthorized write attempts that can be leveraged by an attacker. For example, one such data structure describes non-SMM memory regions, which could be modified by an exploit to disclose or modify SMM memory.

Advanced SMM Protection provides additional exploit mitigation by marking security-critical SMM data as read-only, making it impossible for an attacker to overwrite or modify (**Figure 5**).

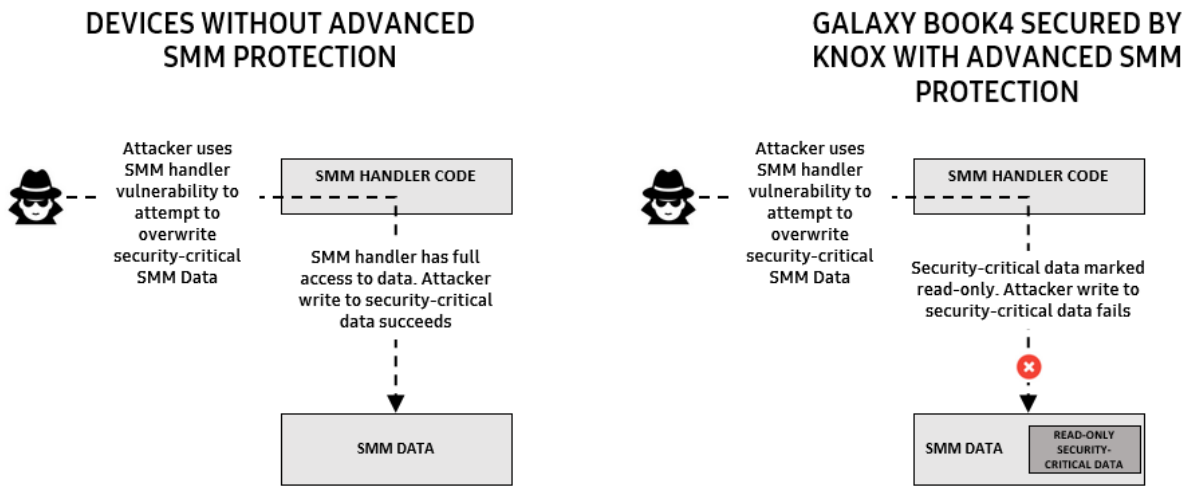

Figure 5: Advanced SMM Protection mitigates zero-day SMM vulnerabilities by blocking writes to security-critical data

Extending Below-the-OS Security into the OS

Having established a secure Below-the-OS platform, it is critical to extend security into the OS. Several security-sensitive operations, such as logging, occur either in the OS kernel or in processes. For robust security, Below-the-OS platform components need to integrate with, and protect these security-critical OS components.

Samsung PCs have special Below-the-OS features to protect security-critical OS components from tamper, and to integrate Below-the-OS security events into Windows events logs.

Samsung Security Service Tamper Protection using Below-the-OS

Malware and ransomware often attempt to kill, disable and uninstall security components such as logging services to evade detection. Therefore, it is important to protect such services from tamper. However, protection is especially challenging since Windows malware often executes with elevated system privileges, allowing malware to easily kill and uninstall security-related services.

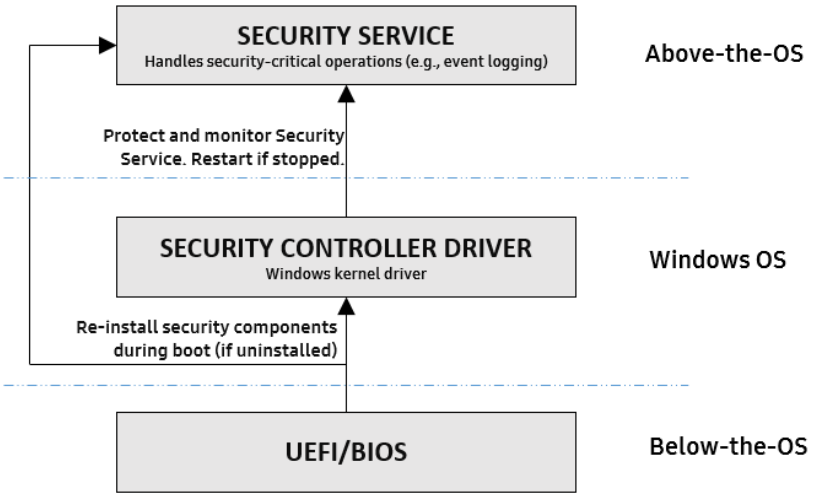

Figure 6: Samsung Security Service Tamper Protection from Below-the-OS

Samsung's Security Service handles critical operations such as logging. It is therefore important that the Samsung Security Service remains protected from tamper.

This protection begins at boot-time and extends through run-time. During boot-time, as shown in **Figure 6**, the UEFI/BIOS, securely re-installs the Samsung's Security Controller Driver and the Samsung Security Service if either of these are uninstalled or not present. It does this by using a standard Windows mechanism called WPBT during boot. This process verifies that the security driver and service binaries are signed by Samsung before starting them.

The Security Controller Driver and Security Service binaries are embedded into the UEFI image, and are thus immune from filesystem tampering. At run-time, the Samsung Security Controller Driver monitors and protects the Security Service by using layered defense mechanisms. If the Security Service process is somehow stopped, the Security Controller Driver is notified through callbacks and instantly restarts the service. The Security Service binary is itself compiled with protection against run-time attacks such as code or DLL injection using various defense-in-depth techniques.

Event Integration with Windows Event Viewer

Another important aspect of extending Below-the-OS security into the OS is to ensure that Below-the-OS security events are appended to existing OS logs in a secure manner. Samsung's Below-the-OS and in-OS security components generate boot-time and runtime security events that are stored in logs in persistent and secure SPI flash memory.

The Security Service's **Event Logging Service** appends these events to Windows Event Logs, which can be viewed using the Windows Event Viewer. To ensure that event logging cannot be bypassed, the Event Logging Service is started on every boot and is protected from tamper or shut-down at run-time (as described in the Samsung Security Service Tamper Protection section).

The Event Logging Service logs several types of security events and information that can be viewed using the Windows Event Viewer under a dedicated "Samsung Security" application event log section. The following events are available:

- **Below-the-OS security events** such as Tamper Alerts, unauthorized writes to BIOS variables and protected SMM areas, BIOS and Secure Boot key recovery attempts, and suspicious SMI calls.

- **In-OS security events** such as uninstalls, unloads of security-critical components such as the Security Service, and writes to security-critical registry keys.
- **Security state of the system**, such as the security software version information, enable/disable status of security-relevant BIOS settings, and variables such as Secure Boot, Advanced SMM Protection, SMI Guard, and TPM activation status.

Event severity levels are one of **Information** (normal but security-relevant events), **Warning** (notice of abnormal events that have been automatically resolved), and **Error** (unrecoverable events that need administrator action).

User Notifications

When certain security-critical events happen at runtime, in addition to being logged, the user is notified of the event through a Toast Notification in Windows. The user has the option to obtain more information or dismiss the event (see **Figure 7**).

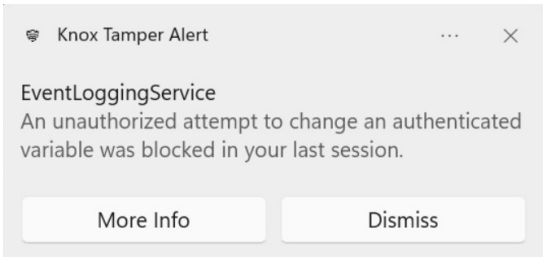

Figure 7: Notification of a Tamper Alert

Glossary

BIOS — A collection of firmware components that includes the below the OS environment and the BIOS setup utility. The BIOS setup utility is used to configure hardware and BIOS features.

BIOS Auto-Recovery — A Samsung Galaxy Book4 Secured by Knox feature through which the Secure Embedded Processor will automatically recover a working BIOS if the original has been corrupted.

Secure Boot Configuration Recovery — A Samsung Galaxy Book4 Secured by Knox feature through which UEFI variables related to BIOS Secure Boot are protected and recovered if corrupted.

Intel Boot Guard — A feature available on Intel processors to validate the initial boot block of the BIOS by signature and version number. The initial boot block verifies and loads subsequent BIOS components.

One Time Programmable (OTP) Fuse Bank — A collection of hardware fuses that can only be set once. This is used to store the firmware rollback protection value and firmware certificate hash.

SPI Flash Storage device — A flash storage device used for storing the BIOS firmware and any data required by the BIOS.

Secure Embedded Processor (SecEP) — A Samsung Galaxy Book4 Secured by Knox processing unit running alongside the main CPU. The Secure Embedded Processor provides security features for the BIOS.

System Management Mode — A privileged CPU mode used for power management, UEFI variable management, and firmware management.

Tamper Alert — A Samsung Galaxy Book4 Secured by Knox feature through which booting an OS requires user or admin confirmation if any boot policy violation has been detected.

Trusted Platform Module (TPM) — A dedicated microcontroller designed for cryptographic operations.

UEFI Standard — Either the UEFI Specification that defines the software interface between an OS and firmware or the reference implementation for the UEFI Specification.

UEFI variables — Persistent data stored on SPI Flash Storage. The format and interface for UEFI Variables are included as part of the UEFI Specification. This data is used by the BIOS and by UEFI components running in the OS.