Cyber-Resilient Platform Requirements

*Ronald Aigner, Paul England, Andrey Marochko, Dennis Mattoon, Rob Spiger, and Stefan Thom*

*Microsoft Corporation*

adjective: **resilient**

… able to withstand or recover quickly from difficult conditions

# Abstract

This specification describes processor and/or platform technologies that provide a foundation for device vendors to build cyber-resilient systems. The technologies are general purpose and can be implemented by any platform, processor, or SoC, but a priority is to define technologies that are suitable for IoT devices.

The mechanisms in this specification are well suited for constructing systems that meet the requirements of NIST SP-800-193 (DRAFT) *“Platform Firmware Resiliency Guidelines.”*

# Introduction

NIST SP-800-153 (DRAFT) identifies the following three principles for building resilient systems: [1]

**Protection:** Mechanisms for ensuring that Platform Firmware code and critical data remain in a state of integrity and are protected from corruption…

**Detection**: Mechanisms for detecting when Platform Firmware code and critical data have been corrupted.

**Recovery**: Mechanisms for restoring Platform Firmware code and critical data to a state of integrity in the event that any such firmware code or critical data are detected to have been corrupted, or when forced to recover through an authorized mechanism.

All Internet-connected devices should be designed to *protect* themselves to network-based attacks, and device vendors employ a wide range of hardware and software-based protection technologies to keep systems secure. Unfortunately, bugs and misconfigurations still lead to damaging exploits. A Cyber-Resilient Platform contains additional mechanisms that allow exploits and vulnerabilities to be *detected*, and for devices to be *recovered* if they are compromised or unresponsive.

Recovering a badly compromised computing device today usually involves manual steps. For example, new firmware or operating systems must be loaded using an external storage device or a second computer. The system must then be rejoined to network services using passwords, or other credentials, under conditions of physical security.

The IoT revolution will deliver orders of magnitude more computing devices. These devices will be built from the same imperfect software that we use today, but manual remediation will be less practical because the devices are too numerous, too inaccessible, and may not even have a suitable local user interface.

Technologies that support reliable and secure remote computer management and recovery are already available for more costly devices, for example, Service Processors (SPs) or Baseboard Management Controllers (BMCs) are employed to manage desktops and servers, and intelligent backplanes are used to manage blades in data centers. However, these technologies are not ideal for the Internet of Things because of their cost, power needs, or the lack of an out-of-band management channel.

The hardware capabilities described in this paper are a foundation for building resilient and secure device management that is appropriate for the smallest of Internet-connected devices (and, of course, larger devices as well). The capabilities are dependable even if the device’s firmware has been compromised by malware and is refusing to cooperate.

## Summary of the Resiliency Building-Blocks

The hardware capabilities described here allow device vendors to establish a small and well-protected *Root of Trust for Resiliency or RTRes (*pronounced “are-tee-rez”*)* for the device.[[1]](#footnote-2) The RTRes enjoys robust protection against malware – both at rest and at runtime. A Cyber-Resilient Platform also provides mechanisms that can be used to ensure that the RTRes is regularly scheduled or can be invoked by authorized controlling entities. The exact capabilities of the RTRes are determined by the device vendor, but secure recovery and update are expected to be core functions.

The specific resiliency features defined in this specification are:

**Stored Data Protection**

* A Write-Protection Latch for non-volatile memory (e.g. flash memory)
  + A Write-Protection Latch allows firmware to write-protect a storage range. Once the Protection Latch is engaged, a platform reset is required to re-enable write-access to the storage range
  + The Root of Trust for Resiliency can use this to protect itself (and possibly configuration data and other parts of the TCB)
  + (A Protection Latch is sometimes called a power-on protected area or sticky-bit-based protection)
* A Read-Protection Latch for non-volatile (e.g. flash) memory
  + To allow the RTRes to protect keys or other secrets

**A Secure Execution Environment for the Root of Trust for Resiliency**

* Devices *must* provide a safe execution environment early in boot, and *may* provide a protected environment when the OS or other platform firmware is running
  + To provide the Root of Trust for Resiliency with a safe place to run

**Attention Triggers**

Attention triggers allow authorized entities to trigger the Root of Trust for Resiliency to perform actions. Three variants are described in this specification:

**A Conventional Watchdog Timer**

* + To trigger execution of the Root of Trust for Resiliency if a device hangs

**A Latchable Watchdog Timer**

* + In contrast to a Conventional Watchdog Timer that can be disabled by malware, a Latchable Watchdog Timer cannot be disabled or deferred after it is set

**An Authenticated Watchdog Timer**

* + To allow an authorized cloud management service to reliably trigger execution of the Root of Trust for Resiliency if a device is misbehaving

A platform that meets the requirements in this specification is termed a *Cyber-Resilient Platform.* Depending on system design, the resiliency features may be implemented entirely in a SoC (System on Chip) or may be distributed across subsystems (e.g. storage controllers and custom logic.)

A Cyber Resilient Platform is designed to provide a secure and resilient *foundation* for an arbitrary Trusted Computing Base. The Trusted Computing Base may be a very simple application package – for example in a sensor-style IoT device - or may be a full-fledged hypervisor running multiple operating systems and applications. The Trusted Computing Base will typically use additional runtime hardware-based protection technologies such as processor privilege levels to protect itself if they are available. The features defined in this specification are designed *to supplement rather than replace* existing protection technologies, and provide remediation if all other protections fail - *i.e.* if the TCB itself is compromised.

The resiliency features can be utilized by standalone devices, but are most powerful when used in conjunction with a vendor or owner-operated cloud management service. Use of a centralized service allows devices to be managed at scale – for example, by providing a single point for device health to be assessed and remediated when needed. The resiliency features can ensure reliable management, even in the face of TCB compromise.

The resiliency features are designed to be both simple to implement in hardware, and simple for software to use. The simplicity increases the chance that systems built using these technologies will be resilient in the face of determined cyber-attack.

Vendors are encouraged to add additional security or resiliency features to improve assurance or meet specialized requirements (for example, dedicated security or management processors.)

# Audience

The Cyber-Resilient Platform Technologies defined in this specification can be implemented in:

* Microprocessors, including SoCs (system-on-chip) and MCUs (microcontrollers)
* Storage controllers (discrete and integrated), and
* Custom logic

Vendors of these systems, as well as other standards groups, are encouraged to incorporate the features defined in this specification.

# Definitions

**Attention Trigger**

A mechanism that lets a user, local firmware, or an Authorized Cloud Controller, invoke the Root of Trust for Resiliency so that management operations can be performed.

**Authenticated Watchdog Timer (AWDT)**

A Watchdog Timer that will initiate a Platform Reset after a specified period unless reset is deferred by cryptographic message from an authorized entity.

**Authorized Cloud Controller**

A network-accessible service that is authorized to manage a device. Authorized cloud controllers may be provided by the Device Vendor or the device owner.

**Boot Loader**

The code that is loaded from non-volatile storage and executed following power-up or a Platform Reset.

**Cyber Resilient Device or System**

A device that implements protection, detection, and recovery mechanisms.

**Cyber Resilient Platform**

A Processor, SoC, or MCU (and attendant logic) that meets the requirements of this specification.

**Cyber-Resilient Watchdog Timer**

A Watchdog Timer that cannot be indefinitely deferred by malware.

**Deferral Ticket**

A cryptographically protected and single use message from an Authorized Cloud Controller that restarts the timer of an Authenticated Watchdog Timer.

**Detection**

Mechanisms to identity compromised firmware or aberrant behavior. Detection, in the context of this specification, can be performed by local software, or by an Authorized Cloud Controller.

**Device Vendor**

The entity that incorporates a Cyber Resilient Platform into a Cyber Resilient Device and provides it to users.

**Firmware and Device Firmware**

The program code, including system software and application code, running on the device (but not including any firmware or microcode that is needed to implement the requirements of this specification).

**Latchable Watchdog Timer (LWDT)**

A Watchdog Timer that will unconditionally cause a Platform Reset after a configured delay.

**Microcontroller (MCU)**

A small CPU.

**Platform Reset**

Reset of a Cyber Resilient Platform, including contained autonomous bus-mastering devices, which meets the requirements of this specification.

**Platform Vendor**

The entity that provides the Cyber-Resilient Platform on which a Cyber-Resilient Device can be built.

**Protection**

Mechanisms that protect a device from interference; normally from internet threats and compromised local software.

**Protection Latch, Write-Protection Latch, Read-Protection Latch**

An access control mechanism that can write- or read-protect a region of non-volatile (flash) storage in such a way that access can only be regained with a power cycle or Platform Reset.

**Recovery**

Mechanisms to repair a device that has been compromised and is refusing to cooperate. Recovery may be use an image provided by the Authorized Cloud Controller or may use a local protected known-good image.

**Root of Trust for Resiliency (RTRes)**

Code that performs functions such as health checks and recovery. Part or all of the RTRes executes early in boot. Some Cyber Resilient Platforms provide protection that allows parts of the RTRes to run during normal device operation.

**System on a Chip (SoC)**

A CPU and attendant logic integrated into a single chip.

**Trusted Computing Base**

The operating system, library operating system, hypervisor, or other systems software, that provide the run-time environment for firmware that implements the main functions of the Cyber Resilient Device

**Watchdog Timer (WDT) and Conventional Watchdog Timer**

A mechanism that generates a Platform Reset if it is not periodically serviced.

# Root of Trust for Resiliency (RTRes)

The resiliency capabilities described in this specification allow a device vendor to construct, hardware-protect, and guarantee periodic execution of, a small and relatively simple *Root of Trust for Resiliency* (RTRes) for the device. The RTRes is responsible for assessing the health, and, if necessary, updating or repairing the remainder of the Trusted Computing Base (and possibly the RTRes itself). The Root of Trust for Resiliency is *not* designed to replace existing operating system or application protection mechanisms; instead the RTRes provides foundational security services to the TCB and can reliably service the TCB when all other defenses have failed.



*Figure 1: Example Cyber-Resilient System using the technology described in this specification.*

The RTRes is strictly a subset of the TCB, since device security depends upon it, but in this specification, it is more convenient to define the RTRes as being foundational to, but not necessarily part of, the Trusted Computing Base.

Figure 1 illustrates one possible software architecture for a Cyber Resilient System built on a Cyber Resilient Platform. In this case, the Root of Trust for Resiliency runs at boot-time and is separate from the remainder of the TCB.

An alternative RTRes packaging architecture is to integrate RTRes functions into the TCB, but take steps to mitigate additional vulnerabilities that arise from using a (potentially) much larger code base for the resiliency tasks. Two possible mitigations are:

* Always write-protect the entirety of device firmware and essential security state during normal operation. This ensures that a Platform Reset evicts any transient (RAM-resident) malware.
* Implement a boot-time safe-mode or RTRes-mode in the TCB. The RTRes-mode only loads and/or runs modules that are essential for resiliency functions, and only interacts with strongly authenticated network entities like the Authorized Management Controller. An RTRes mode mitigates bugs because bugs are much less hazardous if attackers cannot reach them.

Note that this is just a code packaging alternative: the RTRes functions still run at boot time, but the environment that they run in is the specially configured normal run-time environment of the device rather than a separate firmware package.

A second architectural variation is to incorporate some of the resiliency tasks into the *run-time function* of the TCB, rather than only performing them at boot time. This requires a Cyber Resilient Platform that includes support for run-time protection, and a non-maskable interrupt mechanism to ensure that these functions execute periodically.

Companion documents contain detailed descriptions of how the features in this specification can be used to build a cyber resilient device. In this section, a few essential elements are discussed.

**Write-Protection for the RTRes and other Firmware and State**

The Boot Loader is the program code that runs immediately after a platform reset or power-up.

Early in boot, the Boot Loader must use a Write-Protection Latch to protect itself from modification. Engaging write-protection early in boot, and particularly before complex inputs are processed, greatly decreases the chances that latent software defects can be exploited by malware. The Boot Loader (and later firmware) may also use Write-Protection Latches to protect other important system state, like backup recovery images, configuration data, or the Device Firmware itself.

During normal operation, and after the Write-Protection Latch is engaged, the Boot Loader will load, authenticate, and start the remainder of the Device Firmware.

During an update, the Boot Loader / RTRes must validate a candidate update image and then install it *before* the Write Protection Latch is engaged. The risk of persistent compromise (as opposed to transient/RAM compromise) will be reduced if the code performing the update only performs simple cryptographic validation checks on an image that was downloaded in a previous boot cycle.

**Read Protection for Keys**

Boot Loaders executing on platforms without supplemental security processors may use *Read*-Protection Latches to protect cryptographic keys: for example, for device authentication and data encryption.

Read-Protection Latches specifically allow DICE/RIoT-based systems to be built.

(Note: DICE – Device Identity Composition Engine – is functionality that enables secure and resilient/recoverable device identity and attestation schemes to be built. [1] RIoT (Robust, Resilient, Recoverable) IoT is a set of cryptographic techniques and protocols for device identity, attestation, data encryption (etc.) built on a DICE foundation. [2])

**Safe Execution of the RTRes**

A Cyber Resilient Platform must guarantee that a platform reset prepares a safe execution environment for early boot code. Specifically, malware in Device Firmware should not be able to configure the processor or devices to interfere with the proper execution of a simple Boot Loader program following a reset. (Note, that loaders may read data and parameters from storage that was writable by malware. In this case, it is the responsibility of the Loader to carefully validate such inputs.)

Cyber Resilient Platforms *may* also provide a safe execution environment for the RTRes (or part of the RTRes) to execute during normal device operation, as opposed to solely at reset. Many processors provide privilege levels that can be used to protect the RTRes, however if a protection domain contains additional complex software functions, then overall resiliency will likely be impaired. Platforms that provide a safe runtime environment for the RTRes must also implement a non-maskable interrupt timer that the can guarantee periodic execution of the RTRes.

**Attention Triggers**

An Attention Trigger is a mechanism that invokes the Root of Trust for Resiliency. Attention Triggers can be invoked by authorized cloud management controllers or local firmware or hardware.

In contrast to the Protection Latches and RTRes protected execution environments, which are universally valuable building blocks, Attention Triggers are more scenario dependent. This specification includes cyber-resilient variations of Watchdog Timers that can be used to build Attention Triggers for cloud-managed IoT devices. Devices that have other means to invoke the RTRes - e.g. a user pressing a reset button or power cycling the device - may not have need of the Attention Triggers defined in this specification.

Three Watchdog Time variants are described in this specification:

Cyber-Resilient platforms *must* provide a *Conventional Watchdog Timer* (WDT) that can reset the platform and invoke the RTRes if device firmware hangs. However, a Conventional Watchdog Timer can be disabled or indefinitely deferred by malware, so additional mechanisms are required to recover devices in the face of smart malware.

Cyber-Resilient platforms *must also* provide a *Latchable Watchdog Timer* (LWDT) that, once set, *cannot* be disabled or deferred by malware. Boot Loaders can use Latchable Watchdog Timers to ensure that the RTRes executes periodically. Note that conventional Watchdog Timers only fire when the device is hung. The Latchable Watchdog Timer fires during normal device execution, which may interfere with normal device function. This specification suggests several hardware and software techniques that minimize service interruption when the device is policy-compliant and operating normally. Alternatively, device vendors may use a run-time protected RTRes or an Authenticated Watchdog Timer.

An *Authenticated Watchdog Timer* (AWDT) is a third watchdog variant that, once configured, will reset the platform if an authorized management service stops issuing cryptographically protected *“deferral tickets.”* In contrast to a conventional Watchdog Timer, which can be indefinitely deferred by malware, AWDT-reset can only be deferred by the authorized Cloud Management Service.

Practically, if the Cloud Management Service determines that a device is displaying aberrant behavior or refuses to update itself, then the service can stop issuing deferral tickets. After the configured timeout, the AWDT will reset the device, and the RTRes can perform any necessary remediation.

Device vendors that incorporate Attention Triggers must trade off reliable cloud control with the chance that network faults result in unwanted device service interruption. To some extent this can be mitigated by the behavior of the RTRes – for example, a device that cannot contact the cloud service may continue to operate, may enter a safe mode (e.g. all lights flashing red, for a cloud-managed traffic light), or may wait for manual intervention (e.g. entry of new network credentials).

Device Vendors should also consider Watchdog timeout values that balance tolerance for network service interruptions with maximum latency for recovery. Note that it is expected that most management servicing will be coordinated by the Device Firmware itself, which will be performed immediately, because most devices will be operating correctly for most of the time. The Watchdog is designed to recover the fraction of devices that have been compromised and are refusing to update themselves.

# Normative Requirements

## Compliance Requirements

Cyber Resilient Platforms MUST implement

1. The Non-Volatile (e.g. flash) Storage Protection requirements of 5.2, AND
2. The RTRes protected execution requirements of section 5.3, AND
3. The Watchdog Timer requirements of section 5.4

Cyber-Resilient Platforms SHOULD implement the optional requirements of section 5.5.

## Non-Volatile (e.g. flash memory) Storage Protections

File systems implement access control for stored data, but these protections are only reliable if the operating system (or other system firmware) has not been subverted by malware. To mitigate such concerns, non-volatile memory storage controllers (for instance hard disk and flash-memory controllers) offer hardware-based protection capabilities that firmware can use to protect all or part of the device’s persistent state. One significant challenge in implementing hardware-based protection is that a hardware device cannot easily distinguish authorized access (for example, a properly working OS), from unauthorized access (for example, an OS with a rootkit.)

To address this challenge, a common assumption is that early-booting code is less likely to be compromised than the full running system, and the code and state that comprise early-booting code rarely or never needs to be updated by the full running system. A Write-*Protection Latch* is a storage controller subsystem mechanism that lets early boot code write-protect a storage range, in such a way that a platform reset (or power cycle) is required to re-enable write access. Write-Protection Latches compliant with this specification are provided in many SoCs and some storage controller standards.

Another common storage protection feature is irrevocable manufacture-time write-locking of memory regions. This may be appropriate for very simple boot-loaders, but use of this feature inhibits field-updates (and hence may ultimately be detrimental to resiliency) and is *not* required by this specification.

A third feature is a protection latch that disables *read access* to part of the storage device. This feature allows early boot-code to keep secrets, and *is* required by this specification.

### Storage Protection Requirements

**Protection Latch**

A *Protection Latch* is an access-control mechanism for a specific *action* on a specific *resource* that:

1. SHALL be inactive at Platform Reset (defined later in this specification),
2. SHALL present an interface that allows firmware to activate the protection
3. When protection is enabled, the *action* on the *resource* SHALL NOT be allowed
4. The protection SHALL ONLY be deactivated by a Platform Reset
5. The protection, when active, SHALL NOT be by-passable

**Write-Protection Latch**

A *Write-Protection Latch* is a Protection Latch in which the protected *resource* is a region of non-volatile storage, and the *action* is write-access to the storage region.

**Read-Protection Latch**

A *Read-Protection Latch* is a Protection Latch in which the protected *resource* is a region of non-volatile storage, and the *action* is read-access to a storage region.

**Protection Latch Requirements**

1. Devices SHALL implement at least one Write-Protection Latch that can be configured by the Boot Loader and can be used to protect the Boot Loader
   1. The size and start of the protected SHALL be configurable, either during initial manufacturing configuration, or, ideally, be programmatically set on each Processor Reset
   2. Storage devices SHOULD allow more than one write-protected region
2. Storage devices SHALL implement one Read-Protection Latch
   1. The size and start of the protected resource MAY be fixed, but SHOULD be configurable, either during initial manufacturing configuration, or ideally be programmatically set on each Processor Reset
   2. When resources are read-protected they SHALL also be write-protected
   3. Storage devices SHOULD allow more than one read-protected region

## Early Boot Protected Execution Environment and Platform Reset

The Storage Protection Requirements of section 5.2 can be used by the RTRes to protect its stored image and reliably perform servicing operations, but only if it is also protected while it is running. This specification demands that processors implement a *Platform Reset* mechanism that establishes a safe and known environment for the RTRes to be fetched from non-volatile storage and run - regardless of the prior execution state of the platform. Providing a safe execution environment will typically involve the processor resetting some internal registers and caches, but does not require erasing RAM contents. If the device incorporates independent programmable processors or devices that can affect memory or the processor complex (i.e. bus-mastering devices), then these devices must also be reset or quiesced until brought back online by the main processor.

Platform Reset (or its actions) will normally be performed during power-up, and this specification requires that Platform Reset can also be initiated both voluntarily by system firmware, and involuntarily when a Watchdog Timer expires.

Systems that incorporate attestation mechanisms (*e.g.* TPM, DICE/RIoT) must re-compute the attestation information upon Platform Reset.

Vendors are encouraged to allow RAM contents to survive platform reset. This allows the RTRes to work cooperatively with Device Firmware and Watchdog Timer timeouts. Specifically, Device Firmware can quiesce normal operation and voluntarily invoke the RTRes through a programmatic Platform Reset. If the RTRes determines that the device is healthy, it can re-configure the watchdog and jump back into the suspended firmware to continue operation without a lengthy boot process.

The specific requirements are:

1. Platform Reset SHALL transfer control to a Boot Loader program loaded from persistent storage that can be protected using the Storage Protection facilities of section 5.2.
2. Platform Reset SHALL reset sufficient processor internal state (registers, caches) that the Boot Loader can execute reliably
   1. I.e. Previous execution state MUST NOT affect execution of a Boot Loader program that does not access external data
   2. Note: The Boot Loader can (and typically will) read additional code and data (from storage and RAM) that it uses to modify its behavior
3. Platform Reset SHOULD NOT affect memory contents
4. Platform Reset SHALL either:
   1. Reliably reset/quiesce any bus-mastering devices, or
   2. Deny any attempted bus access by bus-mastering devices until access is specifically authorized by the Loader or later Device Firmware
5. Cyber-Resilient Platforms SHALL allow Platform Reset to be invoked programmatically or by a Watchdog Timer, and SHOULD be invokable with external logic (i.e. via a pin)
6. The cause of a Platform Reset (power-up, Watchdog Timer timeout, or programmatically initiated) SHALL be recorded for consumption by the Boot Loader
   1. Loader firmware MUST be able to distinguish Platform Reset from later firmware simply jumping into the start of the Loader
7. Platform Reset – however initiated – SHALL be the only mechanism for resetting the Latchable Watchdog Timer
8. Platform Reset – however initiated – SHALL be the only mechanism for resetting Storage Protection Latches
9. Systems that support attestation SHALL re-measure the Loader code

## Watchdog Timers

The Storage Protections defined in this specification allow system builders to protect the stored image of the RTRes, and this protected code can be used to recover or repair damaged systems at boot time.

However, if there are exploitable bugs in the remainder of device firmware, or if the firmware simply hangs, then the RTRes may not get to run in a timely fashion. The Watchdog Timer capabilities defined in this specification allow vendors to build systems that can periodically and reliably transfer control to the RTRes, even if the TCB becomes uncooperative or unresponsive.

Conventional Watchdog Timer reboots the system if the timer is not regularly serviced. This sort of Watchdog Timer will not reliably reset a compromised device because malware can regularly service the timer but still misbehave in other ways. Platforms compliant with this specification must implement a Watchdog Timer that can reliably invoke the RTRes, even when the TCB is non-cooperative or adversarial. These Watchdog Timer variants are called *Cyber-Resilient Watchdog Timers.*

One Cyber-Resilient Watchdog Timer variant is called a Latchable Watchdog Timer and is mandatory in this specification. Once set, a Latchable Watchdog Timer cannot be directly cancelled by Device Firmware: it will cause a Platform Reset after the configured delay *unless* a Platform Reset happens for other reasons – e.g. by Device Firmware programmatically resetting the device.

The Second Watchdog Timer variant, called an Authenticated Watchdog Timer, is optional and is described in Section 5.5.

Both the Latchable and Authenticated Watchdog Timer variants “fail safe” to invoking the RTRes if the Device Firmware hangs or is uncooperative. However, in the case of the Latchable Watchdog, the RTRes is invoked periodically even if the device is operating correctly. In the case of the Authenticated Watchdog, the RTRes might be invoked unnecessarily because of a network or service fault. The presence of these false alarms has two consequences:

First, one of the primary responsibilities of the RTRes in these systems is to make an authoritative determination of whether remediation is in fact required, rather than immediately taking corrective action. The RTRes is well-protected vendor code, so this can involve relatively sophisticated health assessments; perhaps involving both local firmware/state analysis as well as communication to, or validation of messages from a cloud service.

Second, Watchdog Interrupts are themselves disruptive to device availability if they occur when the device is performing its usual function. This specification suggests mechanisms and technologies that can mitigate service interruption. However, occasional (in the case of an Authenticated Watchdog) or periodic (in the case of the Non-Cancellable Watchdog) interruptions are unavoidable if this Attention Trigger technology is used. Device vendors that employ these Watchdog Timer variants should pick timeouts that balance worst-case recovery latency with the normal operational demands of the device.

This specification also requires conventional (unauthenticated) Watchdog Timers to trigger remediation when devices simply hang. Timeouts for a conventional Watchdog Timer can be much shorter than for Cyber-Resilient Watchdog timers, which will increase device availability in the face of simple hardware and software faults.

The following sections define the requirements for the Watchdog Timers.

### Watchdog Time

Modern processors and systems aggressively put subsystems into low-power states when the system is idle. Some or all processor/device clocks can be reset or stopped in these low-power states. Ideally, Resiliency Watchdog Timeouts should be based on wall-clock time during device execution. If this is not practical, then processors should “fail safe” to invoke Processor Reset whenever the timer value is reset or is indeterminate.

1. Watchdog Time SHOULD be provided by a Real-Time Clock that is not modifiable by device firmware
2. If requirement (1) is not met, then processors SHALL perform a Processor Reset if the Watchdog Time value is reset or suspect
3. The Watchdog clock accuracy should be better than 15%

### Latchable Watchdog Timer (LWDT) Requirements

The Latchable Watchdog Timer is a programmable timer that causes a Platform Reset when the timer expires. The timer can be set by the Boot Loader or later Device Firmware, but once set, cannot be cancelled apart from if a Platform Reset occurs for other reasons.

1. The LWDT performs no functions until configured by software
2. The LWDT SHALL allow firmware to set the timeout
3. When the timer expires, the Watchdog Timer SHALL cause a Platform Reset
4. The timer SHALL ONLY be cancelled through either:
   1. A Platform Reset
   2. A power cycle
5. After a Platform Reset, the Latchable Watchdog Timer SHALL be re-startable with the same or different timeout delay (i.e. requirement 2)

Although not a normative requirement, vendors should strive to minimize the service interruption resulting from a watchdog timeout. Service disruption for devices compliant with this specification is alleviated by the following normative requirements:

Requirement 4(b) in this section allows firmware to pre-emptively reset the device. This allows Device Firmware to schedule RTRes actions when they will be minimally disruptive.

Requirement 3 in section 5.3 is a recommendation that main-memory state survives Platform Reset. This allows software to coordinate with the RTRes to suspend and then resume operation from a RAM-resident image following Platform Reset, rather than performing a full device reboot.

### Conventional Watchdog Timer

Vendors SHALL incorporate a conventional watchdog timer that can invoke the RTRes if Device Firmware hangs. The Conventional Watchdog Timer mechanism is vulnerable to sophisticated malware attack (e.g. malware can keep restarting the timer, or possibly even cancel it), but since conventional Watchdog Timers do not interfere with normal operation, the timeout value can be short allowing fast recovery of devices in most circumstances.

The requirements are:

1. The Conventional Watchdog Timer performs no functions until configured by firmware
2. The Conventional Watchdog Timer SHALL allow firmware to set the timeout delay before Processor Reset is triggered
3. If the Conventional Watchdog Timer is not restarted before the timeout, the WDT SHALL cause a Platform Reset

## Optional Security and Resiliency Enhancements

Platforms compliant with this specification must meet the requirements of Section 5. The required functionality allows secure, reliable, and recoverable devices to be built, but with certain compromises. Notable issues are:

1. The Root of Trust for Resiliency can only run at boot time. Even with the hardware and software techniques described in this specification, device vendors will need to trade off service interruption with worst-case recovery latency
2. The key protection afforded by Read-Protection Latches falls short of that of a dedicated security processor
3. The use of the Latchable Watchdog Timer implies occasional Platform Resets – even when the device is healthy and compliant

In this section, optional Resilient Platform enhancements that mitigate these issues are defined. Vendors are encouraged to include these or other solutions to secure and reliable device management and recovery.

### Protected Runtime Environment for the RTRes

Devices MAY provide a protected runtime environment for the RTRes (or part of the RTRes). If the context switch latency for entering and exiting the protected environment is less than a Platform Reset, then a Protected Runtime environment will reduce service interruption during health and management checks.

If implemented:

1. The RAM or NV-storage backing the execution environment SHALL NOT be readable or writable by any hardware thread that is not currently running in the protected environment
2. The RAM or NV-storage backing the execution environment SHALL NOT be readable or writable by peripheral devices
3. The protected environment SHALL enforce a well-defined call/entry sequence to a predetermined start address. Prior processor state MUST NOT be able to affect execution of code in the RTRes *except* through explicit action of the RTRes – for instance, by the RTRes reading a parameter from a register or memory location
4. Interrupts SHALL be masked or deferred on environment-entry
5. The platform SHALL provide a non-maskable interrupt facility that the protected environment can use to guarantee periodic execution
6. Code running in the protected environment SHALL be able to trigger a Platform Reset

All but the smallest of processors already provide privilege levels e.g. user/supervisor, normal/hypervisor, or trusted/untrusted, and these processors will generally meet the run-time protection mechanisms set forth in this specification.

The most important software design observation when using run-time protected environments is that if an RTRes that is part of a larger component (e.g. the RTRes is running in TrustZone on an ARM processor, or System Management Mode on Intel architectures), the RTRes will not be able to reliably provide service to software in the environment that it shares. For instance, an RTRes in TrustZone may be able to service the main/rich OS but will not in general be trustworthy in assessing the health and remediating the Trusted Execution Environment itself. In other words, if the other code in the run-time protected environment is complex and buggy, then overall resiliency will be reduced.

For these reasons, processor vendors SHOULD provide execution modes that can be under the exclusive control of the RTRes rather than shared with other functions. Software vendors are urged to exclusively dedicate these capabilities to resiliency functions.

### Authenticated Watchdog Timer (AWDT) Requirements

The Authenticated Watchdog Timer (AWDT) is a programmable timer that causes control to be transferred to the RTRes when the timer expires, but in contrast to the simple Latchable Watchdog, the Authenticated Watchdog timeout can be deferred by cryptographically verified statements from the Authorized Cloud Service. The AWDT removes the requirement that devices periodically perform a Platform Reset to service the RTRes when they are healthy and policy compliant.

The AWDT is configured by setting the timeout period (as with any watchdog timer), but also by setting the entity authorized to issue Deferral Tickets. The AWDT can use symmetric or asymmetric cryptosystems, or a one-time-pad configured by the RTRes. In the case of symmetric or asymmetric cryptosystems, the AWDT must ensure that Deferral Tickets can only be used once. In the case of the one-time-pad, the watchdog must ensure that the pad-entries are ONLY used once and perform a Platform Reset if the one-time-pad is exhausted.

This specification contains high-level functional requirements. A companion specification includes specific cryptosystems that should be implemented. Vendors are encouraged to build devices that also comply with the companion specification.

The detailed normative requirements are:

1. The AWDT performs no functions until configured by firmware
2. The AWDT SHALL allow firmware to set the timeout delay before the RTRes is invoked
3. The AWDT SHALL allow firmware to set an authentication token (e.g. symmetric or public key) of entities authorized to defer AWDT timeouts
4. If the AWDT expires, the timer SHALL cause a Platform Reset
5. The timer SHALL ONLY be cancelled through either:
   1. Platform Reset
   2. Power cycles
6. The AWDT timer SHALL be restarted if a cryptographically authenticated Deferral Ticket is presented
7. The Deferral Tickets SHALL be single use
8. After a Platform Reset, the AWDT SHALL be re-startable with the same or different authorizing entity and timeout delay (i.e. requirement 2)

### Protection Latches for RAM

On reset, the RTRes must be reloaded from write-protected non-volatile storage to maintain the platform’s security guarantees. Copying code from NV to RAM, or executing code directly from NV, has lower performance than running from RAM. However, RAM-contents are generally under the exclusive control of the (potentially buggy) TCB, so RAM contents must be considered untrustworthy by the RTRes (and certainly not be executed directly before being checked).

Vendors may incorporate Write-Protection and Read-Protection latches for RAM that allow early-boot code and data to be safely cached in fast RAM, rather than reloaded from write-protected NV-storage.

### Hardware Security Processor

Vendors may include hardware security processors offering higher levels of protection for keys than can be achieved by the Read and Write-Protection Latches of section 5.2.

Hardware security processors SHOULD include an asymmetric key pair and an attendant cryptographic engine that can provide a highly-protected long-term cryptographic identity for the device.

Hardware security processors SHOULD also support cryptographic reporting (attestation) of the firmware that booted on the device.

Hardware Security Processors SHOULD provide a foundation for secure boot (e.g. to store a secure boot policy and/or code measurement and authorization checks.)

### Physical Access Protections

The normative requirements in this specification address Internet and software-based attacks on the platform. Some devices may also require protection against hardware-based attacks. Vendors SHOULD incorporate mitigations for simple hardware or hardware-mediated software attacks on the platform (e.g. power-glitching protection, or use of simple debug interfaces to harvest long-lived secret keys.) Vendors MAY include protections for more sophisticated attacks.

# Conclusions

This specification describes platform/processor mechanisms that can be used to build cyber-resilient systems.

# References

|  |  |
| --- | --- |
| [1] | A. Regenscheid, *Platform Firmware Resiliency Guidelines - SP 800-193 (DRAFT),* 2017. |
| [2] | *Trusted Platform Architecture - Hardware Requirements for a Device Identifier Composition Engine (DRAFT),* 2017. |
| [3] | P. England, A. Marochko, D. Mattoon, S. Thom and D. Wooten, *RIoT – A Foundation for Trust in the Internet of Things,* 2016. |

1. The acronym “RTR” denotes the *Root of Trust for Reporting* in systems that support attestation. Cyber Resilient Platforms support attestation (and the RTRes will usually contain RTR functions), so the longer acronym is used. [↑](#footnote-ref-2)