Need for Trustworthy Systems

In Cisco IOS XR Release 7.0.1, this section is applicable *only* to the following Cisco NCS 540 variants:

- N540-28Z4C-SYS-A/D
- N540X-16Z4G8Q2C-A/D
- N540-12Z20G-SYS-A/D
- N540X-12Z16G-SYS-A/D

Starting Cisco IOS XR Release 7.3.1, this section is also applicable to the following Cisco NCS 540 variants:

- N540X-6Z18G-SYS-A/D
- N540X-8Z16G-SYS-A/D
- N540-FH-CSR-SYS

Global service providers, enterprises, and government networks rely on the unimpeded operation of complex computing and communications networks. The integrity of the data and IT infrastructure is foundational to maintaining the security of these networks and user trust. With the evolution to anywhere, anytime access to personal data, users expect the same level of access and security on every network. The threat landscape is also changing, with adversaries becoming more aggressive. Protecting networks from attacks by malevolent actors and from counterfeit and tampered products becomes even more crucial.

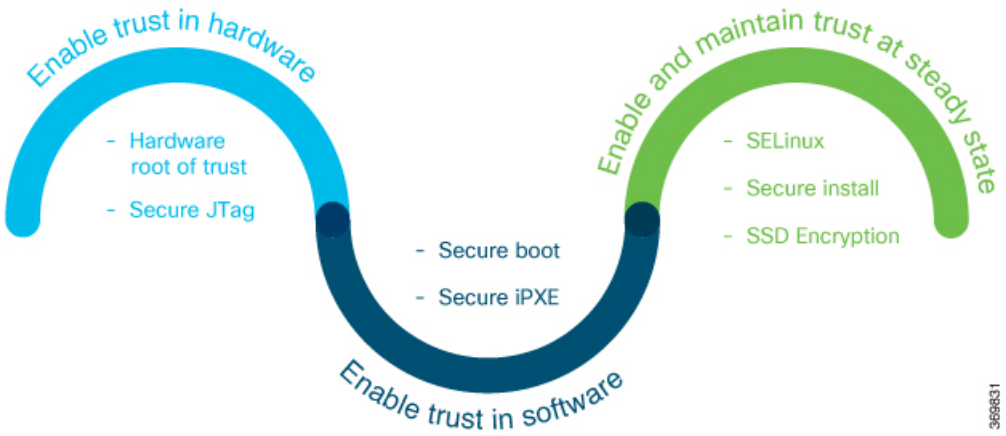
Routers are the critical components of the network infrastructure and must be able to protect the network and report on system integrity. A “trustworthy solution” is one that does what it is *expected* to do in a *verifiable* way. Building trustworthy solutions requires that security is a primary design consideration. Routers that constitute trustworthy systems are a function of security, and trust is about preventing as well as knowing whether systems have been tampered with.

In trustworthy systems, trust starts at the lowest levels of hardware and is carried through the boot process, into the operating system (OS) kernel, and finally into runtime in the OS.

The main components of implementing a trustworthy system are:

- Enabling trust in hardware with Hardware root-of-trust and secure JTAG
- Enabling trust in software with secure boot and secure iPXE
- Enabling and maintaining trust at steady state with Security-Enhanced Linux (SELinux), Secure install, and SSD Encryption

Figure 1: Ecosystem of Trustworthy Systems



Trustworthy systems must have methods to securely measure hardware, firmware, and software components and to securely attest to these secure measurements.

For information on key concepts used in this chapter, see the Understanding Key Concepts in Security.

- Enable Trust in Hardware, on page 2
- Enable Trust in Software, on page 6
- Establish and Maintain Trust at Steady State, on page 8
- How Trustworthiness is Implemented, on page 27
- Understanding Key Concepts in Security, on page 27

Enable Trust in Hardware

The first component in implementing a trustworthy system is to enable trust in hardware.

Because software alone can’t prove a system's integrity, truly establishing trust must also be done in the hardware using a hardware-anchored root of trust. Without a hardware root of trust, no amount of software signatures or secure software development can protect the underlying system from becoming compromised. To be effective, this root of trust must be based on an immutable hardware component that establishes a chain of trust at boot-time. Each piece of code in the boot process measures and checks the signature of the next stage of the boot process before the software boots.

A hardware-anchored root of trust is achieved through:

- Anti-counterfeit chip: All modules that include a CPU, as well as the chassis, are fitted with an anti-counterfeit chip, which supports co-signed secure boot, secure storage, and boot-integrity-visibility. The chip ensures that the device's software and hardware are authentic and haven’t been tampered with or modified in any way. It also helps to prevent unauthorized access to the device's sensitive data by enforcing strong authentication and access control policies.
- Secure Unique Device Identifier (SUDI): The X.509 SUDI certificate installed at manufacturing provides a unique device identifier. SUDI helps to enable anti-counterfeit checks along with authentication and remote provisioning. The SUDI is generated using a combination of the device's unique hardware identifier (such as its serial number or MAC address) and a private key that is securely stored within the device. This ensures that each SUDI is unique and cannot be easily duplicated or forged. When a device attempts to connect to a network, the network uses the SUDI to authenticate the device, and ensure that it’s

authorized to connect. This helps to prevent unauthorized access to the network and ensures that only trusted devices are allowed to connect.

- Secure JTAG: The secure JTAG interface is used for debugging and downloading firmware. This interface with asymmetric-key based authentication and verification protocols prevents attackers from modifying firmware or stealing confidential information. Secure JTAG typically involves a combination of hardware and software-based security measures. For example, it may include the use of encryption and authentication protocols to secure communications between the JTAG interface and the debugging tool. It may also involve the use of access control policies and permissions to restrict access to the JTAG interface to authorized users only.

Note Hardware-anchored root of trust is enabled by default on Cisco NCS 540 Series routers.

Verification

You can verify if trust is enabled in the hardware by executing the following command:

```
Router#show platform security integrity hardware
Wed Apr 17 11:19:03.202 UTC

+-------------------------------------+
   Node location: node0_RP0_CPU0
+-------------------------------------+
TPM Name: node0_RP0_CPU0_aikido
Uptime: 52050
Known-good-digests:
Index   value
  0     hh4jzFBlxSGHZ4hKqnC2FEjqHg4tpx/chZ7YcTwLCco=
observed-digests:
Index   value
  0     hh4jzFBlxSGHZ4hKqnC2FEjqHg4tpx/chZ7YcTwLCco=
PCRs:
Index   value
  15    Dl1BGskyzeJ1LNYKuZK8Qqllwkth0ru+0xWydL9YMdc=
```

Secure Hardware for Strong Cryptography

All Cisco IOS XR7 supported-platforms ships with a non-tamperable Trust Anchor module (TAm) in the hardware.

TAm houses known-good-values (KGVs) of the hardware components along with keys and certificates rooted to Cisco, which are used to verify components of the hardware during the BIOS boot.

Chip Guard and Attestation are security features implemented in TAm to enable trust in hardware.

- Chip Guard detects tampering attempts and responds by initiating actions such as disabling access to the device, erasing sensitive information stored in the device, or triggering a security alarm.
- Attestation provides a mechanism for verifying the integrity and authenticity of the software and hardware components of a device.

A Cisco router with SUDI is authenticated and verified remotely for uniquely identifying that it’s an authentic Cisco device.

Note Some Cisco NCS 540 Series Routers have the older generation of chips with lesser capabilities compared to the latest TAm chips present on the newer generation of hardware.

Hardware Integrity Check Using Chip Guard Functionality

Table 1: Feature History Table

Feature Name	Release Information	Feature Description
Hardware Integrity Check Using Chip Guard Functionality	Release 7.4.1	Support for the Chip Guard functionality is now extended to the following Cisco NCS 540 router varaint: • N540X-6Z18G-SYS-A/D • N540X-8Z16G-SYS - A/D • N540X-4Z14G2Q-SYS-A/D

The chip guard feature helps detect if attackers have replaced a Cisco router's Application Specific Integrated Circuit (ASIC) chip or CPU chip with a counterfeit one when the device is in the manufacturing supply chain. The ASIC performs critical functions, such as scanning an egress queue for error causes and a CPU runs the operating system. If these chips are counterfeited, the performance, reliability, and security of the router is compromised. During the hardware integrity check, at the time of device boot, if the chip guard feature identifies a counterfeit ASIC or CPU, it halts the secure boot process and displays a warning to inform that the supply chain integrity has been compromised.

Why do We Need Chip Guard

The increased hardware supply chain attacks compromise physical components, where attackers replace the ASIC or CPU on a router with malware-infested chips. Once the ASIC or CPU is replaced, the integrity of the hardware is compromised. Counterfeit chips in a router may have hidden functionalities to create a larger security vulnerability. Cisco's chip guard feature detects counterfeit chips before the router is deployed in the network.

Stages of Chip Guard Implementation

The table shows the various stages through which chip guard is implemented on the router.

Stage	Process/Action	Result
1. Router Manufacturing	SHA 256 hashes of the electronic chip IDs of both the CPU and ASIC are programmed in the TAm chip and stored in a database known as Imprint DB.	The Imprint DB inside the TAm chip contains the SHA 256 hashes, which cannot be modified during the router's lifetme.

Stage	Process/Action	Result
2. Router Deployed in the Field and Powered Up	During the secure boot process, the chip guard feature recomputes the SHA 256 hashes of the electronic chip IDs of both the CPU and ASIC and creates a database known as Observed DB.	The Observed DB values are stored inside the TAm chip.
3. Comparison of Imprint DB and Observed DB	DBs match	The router continues to boot. Depending on the capability of the underlying router, the chip guard feature validates either the CPU, ASIC, or both.
	DBs do not match	The router notifies that either the CPU or ASIC is counterfeit, and the secure boot process halts. A message is displayed on the console about the chip guard validation failure.

Action to be Taken on Hardware Validation Failure

If you receive a chip guard warning about integrity check failure, you must create a service request on the Products Returns & Replacement (RMA) website.

Attestation

Table 2: Feature History Table

Feature Name	Release Information	Feature Description
Support for Attestation	Release 7.4.1	**Attestation** is a mechanism used by a trusted entity to validate the software integrity of a platform. Support for attestation is now extended to the following Cisco NCS 540 router variant: • N540-24Q8L2DD-SYS

Attestation enables external verifiers to check the integrity of the software running on the host. Implementing this feature on Cisco hardware helps you validate the trustworthiness of the hardware and software of network devices.

Once a router is up and running, you can send a request to an external verifier. The external verifier requests an attestation quote from the router. The TAm chip can output the PCR quote and audit log, and it signs the quote using an attestation private key for that specific router and responds to the verifier. The verifier uses Cisco-provided KGV hashes and the Attestation Public Certificate to verify the attested PCR quotes and audit logs. This verification is protected against repeat attacks using a nonce. Besides this, the verification ensures that the attestation is specific to a particular router by using attestation key pairs. These attestation key pairs

are unique to each router. This ensures that attackers do not tamper with the router hardware, boot keys, boot configuration, and running software.

Proof of hardware integrity is recorded in the TAm as part of Chip Guard. This proof is made available through the following command:

Note The same data is also available through NETCONF for a remote attestation server: Cisco-IOS-XR-remote-attestation-act.yang.

```
RP/0/RP0/CPU0:NCS-540-C-LNT#show platform security attest pcr 0 trustpoint ciscoaik nonce
4567 location 0/RP0/CPU0
Thu Apr 11 05:44:10.808 UTC
Nonce: 4567

+--------------------------------------+
   Node location: node0_RP0_CPU0
+--------------------------------------+
Uptime: 1198700
pcr-quote:
/1RDR4AYACCkyXSBYFKZwSNurif7DxQRMrBTg6q91hecKFZW0kp0RQACRWcAAAAABX5fDQAAA97/////AQAAACQAAAALAAAAAQALAwEAAAAgrE798LlCkKp1kryIv50kG0/V46lIQutuSVgCUwjG8q4=
pcr-quote-signature:
mC8oPWYzgSTge31DLXCs/Ez7BRKsZyDVb4auuhJagWHa3aHSa9eMf34Y/FMuTitjeAhcs——<truncated>——dJYpsPKMGkcro1IquThaD1gKIH+Gh4QBewdNky3Igiw==
pcr-index       pcr-value
  0             sL3H+Em2xzxXrNUoDF+kC47IXxN4V/d/7hYUXApXNoY=
```

See the System Security Command Reference guide for more commands.

Enable Trust in Software

The second component in implementing a trustworthy system is to enable trust in software.

In Cisco IOS XR7, trust in the software is enabled through:

- Secure Boot
- Secure iPXE

Secure Boot

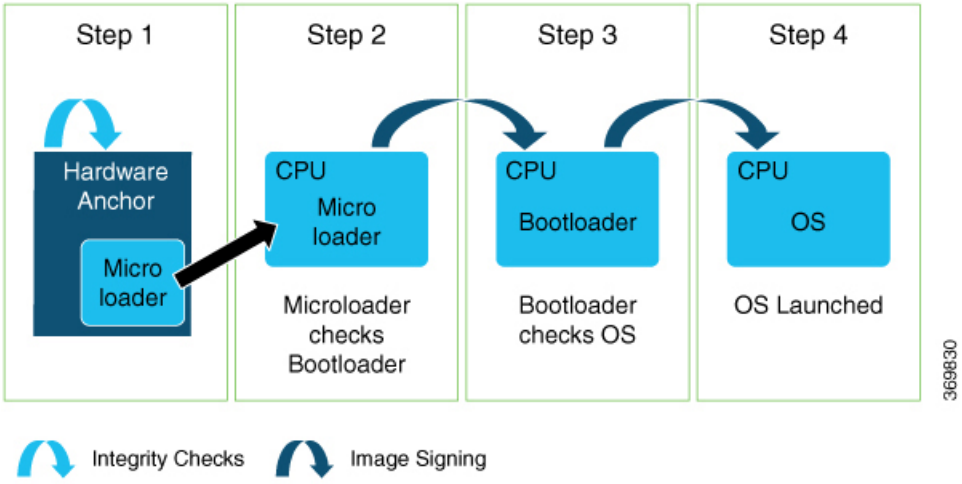
Table 3: Feature History Table

Feature Name	Release Information	Feature Description
Support for Secure Boot	Release 7.4.1	Support for Secure Boot is now extended to the following Cisco NCS 540 router variant: • N540-24Q8L2DD-SYS

Cisco Secure Boot helps to ensure that the code that executes as part of the software image boot up on Cisco routers is authentic and unmodified. Cisco IOS XR7 platforms support the hardware-anchored secure boot which is based on the standard Unified Extensible Firmware Interface (UEFI). This UEFI-based secure boot

protects the microloader (the first piece of code that boots) in tamper-resistant hardware, establishing a root of trust that helps prevent Cisco network devices from executing tainted network software.

Figure 2: Secure Boot



The intent of Secure Boot is to have a trust anchor module (TAm) in hardware that verifies the bootloader code. A fundamental feature of secure boot is the barrier it provides that makes it that it is extremely difficult or nearly impossible to bypass these hardware protections.

Secure boot ensures that the bootloader code is a genuine, unmodified Cisco piece of code and that code is capable of verifying the next piece of code that is loaded onto the system. It is enabled by default.

When secure boot authenticates the software as genuine Cisco in a Cisco device with the TAm, the operating system then queries the TAm to verify whether the hardware is authentic. It verifies by cryptographically checking the TAm for a secure unique device identifier (SUDI) that comes only from Cisco.

The SUDI is permanently programmed into the TAm and logged by Cisco during Cisco's closed, secured, and audited manufacturing processes.

Booting the System with Trusted Software

In Cisco IOS XR7, the router supports the UEFI-based secure boot with Cisco-signed boot artifact verification. The following takes place:

Step 1: At bootup, the system verifies every artifact using the keys in the TAm.

Step 2: The following packages are verified and executed:

- Bootloader (Grand Unified Bootloader (GRUB), GRUB configuration, Preboot eXecution Environment (PXE), netboot)
- Initial RAM disk (Initrd)
- Kernel (operating system)

Step 3: Kernel is launched.

Step 4: Init process is launched.

Step 5: All Cisco IOS XR RPMs are installed with signature verification.

Step 6: All required services are launched.

Secure iPXE – Secure Boot Over the Network

The iPXE server is an HTTP server discovered using DHCP that acts as an image repository server. Before downloading the image from the server, the Cisco router must authenticate the iPXE server.

Note A secure iPXE server must support HTTPS with self-signed certificates.

The Cisco router uses certificate-based authentication to authenticate the iPXE server. The router:

- Downloads the iPXE self-signed certificates
- Uses the Simple Certificate Enrollment Protocol (SCEP)
- Acquires the root certificate chain and checks if it's self-signed

The root certificate chain is used to authenticate the iPXE server. After successful authentication, a secure HTTPS channel is established between the Cisco router and the iPXE server. Bootstrapper protocol (Bootp), ISO, binaries, and scripts can now be downloaded on this secure channel.

Establish and Maintain Trust at Steady State

The third component in implementing a trustworthy system is to maintain trust in the steady or runtime state.

Attackers are seeking long-term compromise of systems and using effective techniques to compromise and persist within critical infrastructure devices. Hence, it is critical to establish and maintain trust within the network infrastructure devices at all points during the system runtime.

In Cisco IOS XR7, trust is established and maintained in a steady state through:

- SELinux
  - SELinux Policy
  - SeLinux Mode
- Secure Install
  - RPM Signing and Validation
  - Third-Party RPMs
- SSD Encryption

SELinux

Security-Enhanced Linux (SELinux) is a Linux kernel security module that provides a mechanism for supporting access control security policies, including mandatory access controls (MAC).

A kernel integrating SELinux enforces MAC policies that confine user programs and system servers to the minimum amount of privileges they require to do their jobs. This reduces or eliminates the ability of these programs and daemons to cause harm when compromised (for example, through buffer overflows or

misconfigurations). This confinement mechanism operates independently of the traditional Linux access control mechanisms. SELinux has no concept of a "root" super-user and does not share the well-known shortcomings of the traditional Linux security mechanisms (such as a dependence on setuid/setgid binaries).

On Cisco IOS XR7 software, only Targeted SELinux policies are used, so that only third-party applications are affected by the policies; all Cisco IOS XR programs can run with full root permission.

With Targeted SELinux, using targeted policies, processes that are targeted run in a confined domain. For example, the httpd process runs in the httpd_t domain. If a confined process is compromised by an attacker, depending on the SELinux policy configuration, the attacker's access to resources and the possible damage that can result is limited.

Note Processes running in unconfined domains fall back to using discretionary access control (DAC) rules.

DAC is a type of access control defined as a means of restricting access to objects based on the identity of the subjects or the groups (or both) to which they belong.

SELinux Policy

Each Linux user is mapped to an SELinux user through an SELinux policy. This allows Linux users to inherit the restrictions placed on SELinux users.

If an unconfined Linux user executes an application, which an SELinux policy defines as an application that can transition from the unconfined_t domain to its own confined domain, the unconfined Linux user is subject to the restrictions of that confined domain. The security benefit is that, even though a Linux user is running in unconfined mode, the application remains confined. Therefore, the exploitation of a flaw in the application is limited by the policy.

A confined Linux user is restricted by a confined user domain against the unconfined_t domain. The SELinux policy can also define a transition from a confined user domain to its own target confined domain. In such a case, confined Linux users are subject to the restrictions of that target confined domain.

SELinux Mode

There are three SELinux modes:

- Enforcing: When SELinux is running in enforcing mode, it enforces the SELinux policy and denies access based on SELinux policy rules.
- Permissive: In permissive mode, the SELinux does not enforce policy, but logs any denials. Permissive mode is used for debugging and policy development. This is the default mode.
- Disabled: In disabled mode, no SELinux policy is loaded. The mode may be changed in the boot loader, SELinux config, or at runtime with **setenforce**.

Role of the SELinux Policy in Boot Process

SELinux plays an important role during system startup. Because all processes must be labeled with their proper domain, the init process performs essential actions early in the boot process that synchronize labeling and policy enforcement.

Secure Install

The Cisco IOS XR software is shipped as RPMs. Each RPM consists of one or more processes, libraries, and other files. An RPM represents a collection of software that performs a similar functionality; for example, packages of BGP, OSPF, as well as the Cisco IOS XR Infra libraries and processes.

RPMs can also be installed into the base Linux system outside the Cisco IOS XR domain; however, those RPMs must also be appropriately signed.

All RPMs shipped from Cisco are secured using digitally signed Cisco private keys.

There are three types of packages that can be installed:

- Packages shipped by Cisco (open source or proprietary)
- Customer packages that replace Cisco provided packages
- Customer packages that do not replace Cisco provided packages

RPM Signing and Validation

Table 4: Feature History Table

Feature Name	Release Information	Feature Description
Support for RPM Signing and Validation	Release 7.4.1	Support for RPM Signing and Validation is now extended to the following Cisco NCS 540 router varaint: • N540-24Q8L2DD-SYS

RPMs are signed using Cisco keys during the build process.

The install component of the Cisco IOS XR automatically performs various actions on the RPMs, such as verification, activation, deactivation, and removal. Many of these actions invoke the underlying DNF installer. During each of these actions, the DNF installer verifies the signature of the RPM to ensure that it operates on a legitimate package.

Cisco RPMs are signed using GPG keys. The RPM format has an area dedicated to hold the signature of the header and payload and these are verified and validated via DNF package managers.

Third-Party RPMs

The XR Install enforces signature validation using the ‘gpgcheck’ option of DNF. Thus, any Third-Party RPM packages installation fails if done through the XR Install (which uses the DNF).

SSD Encryption

Table 5: Feature History Table

Feature Name	Release Information	Feature Description
Solid State Drive (SSD) Encryption	Release 7.3.1	This feature enables trust and security in the system's steady state by encrypting data at the disk level. The encrypted data can be accessed *only* with a specific key stored in the TAm. In this release, the feature is supported on: • N540-28Z4C-SYS-A/D • N540X-16Z4G8Q2C-A/D • N540-12Z20G-SYS-A/D • N540X-12Z16G-SYS-A/D • N540X-6Z18G-SYS-A/D

Customers are concerned about the security of sensitive data present on persistent storage media. User passwords are limited in their capability to protect data against attackers who can bypass the software systems and directly access the storage media.

In this case, only encryption can guarantee data confidentiality.

Cisco IOS XR Software introduces SSD encryption that allows encrypting data at the disk level. SSD encryption also ensures that the encrypted data is specific to a system and is accessible *only* with a specific key to decrypt them.

Data that can be encrypted is sensitive information such as, topology data, configuration data, and so on.

Encryption is an automatic process and can be achieved through the following:

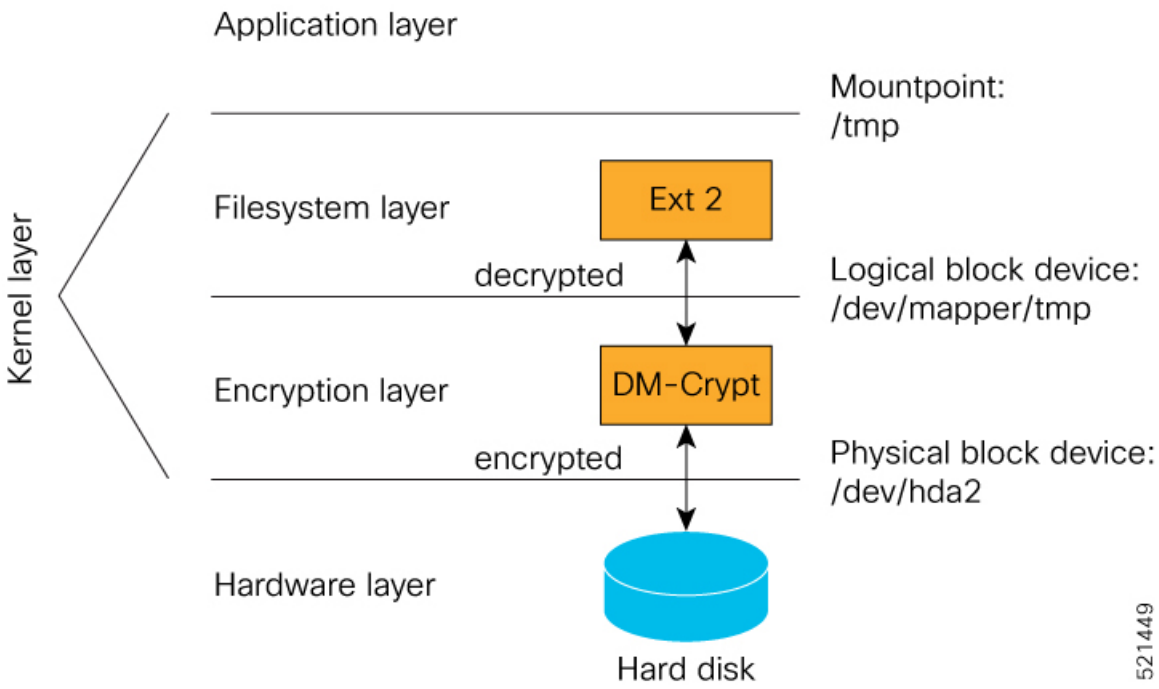
- DM-Crypt
- CPU with AES-NI support
- CryptSetup

DM-Crypt

DM-Crypt is a Linux kernel module that provides disk encryption. The module takes advantage of the Linux kernel's device-mapper (DM) infrastructure. The DM provides a way to create virtual layers of block devices.

DM-crypt is a device-mapper target and provides transparent encryption of block devices using the kernel crypto API. Data written to the block device is encrypted; whereas, data to be read is decrypted. See the following figure.

Figure 3: DM-Crypt Encryption

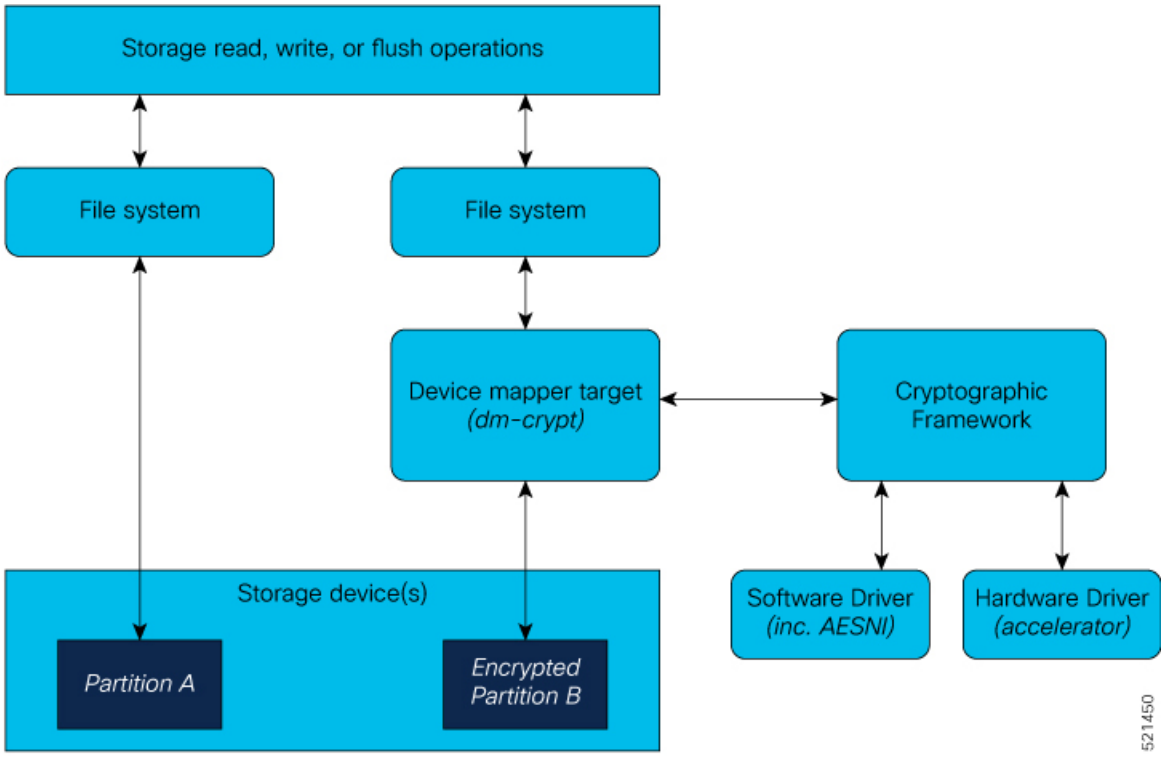


AES-NI Support

Intel's Advanced Encryption Standard New Instructions (AES-NI) is a hardware-assisted engine that enables high-speed hardware encryption and decryption. This process leaves the CPU free to do other tasks.

When the input-output operations are started, the read-write requests that are directed at the encrypted block device are passed to the DM-Crypt. DM-Crypt then sends multiple cryptographic requests to the Cryptographic Framework. The crypto framework is designed to take advantage of off-chip hardware accelerators and provides software implementations when accelerators are not available. See the following image.

Figure 4: AES-NI Support



CryptSetup

DM-Crypt relies on user space tools, such as cryptsetup to set up cryptographic volumes. Cryptsetup is a command-line-interface (CLI) tool that interacts with DM-Crypt for creating, accessing, and managing encrypted devices.

Encrypted Logical Volume

An encrypted logical volume (LV) can be created during software installation on the following Cisco NCS 540 router variants:

- N540-28Z4C-SYS-A/D
- N540X-16Z4G8Q2C-A/D
- N540-12Z20G-SYS-A/D
- N540X-12Z16G-SYS-A/D
- N540X-6Z18G-SYS-A/D

You can activate or deactivate the encrypted disk partition on demand. In addition to being activated, all sensitive files are also migrated from the unencrypted disk partition to the encrypted disk partition. The encrypted files can be migrated back during deactivation.

You can activate the data encryption by using the `disk-encryption activate` *`location`* command. A sample output is as follows:

```
Router#disk-encryption activate location 0/RP0/CPU0
Tue Apr 16 14:35:00.939 UTC

Preparing system for backup. This may take a few minutes especially for large configurations.

 Status report: node0_RP0_CPU0: START TO BACKUP
Router# Status report: node0_RP0_CPU0: BACKUP HAS COMPLETED SUCCESSFULLY
[Done]
```

The encrypted logical volume capacity is 150MB of disk space and is available as /var/xr/enc for applications to access.

Note Although applications can choose to use this space for storage, that data is not be part of the data migration if the software image is downgraded to a version that does not support encryption.

SSD Binding

When encryption is activated on a system, each card generates a random encryption key and stores it in its own secure storage—the Trust Anchor module (TAm). During successive reboots, the encryption key is read from the TAm and applied to unlock the encrypted device. Since each card stores its encryption key locally on the TAm, an SSD that is removed from one card and inserted into another cannot be unlocked by the key stored on that card, thereby making the SSD unusable.

If encryption is activated, the encrypted LV can only be unlocked by using the key stored in the TAm. So, if an encrypted SSD is removed and moved to another line card, the SSD cannot be unlocked. In other words, when you activate encryption, the SSD is bound to the card it is inserted in.

Data Zeroization

Zeroization refers to the process of deleting sensitive data from a cryptographic module.

Note In case of a Return Material Authorization (RMA), you must *factory reset* the data.

You can perform zeroization by using the `factory reset` *`location`* command from the XR prompt.

Caution Running this command while encryption is activated, deletes the master encryption key from the TAm and renders the motherboard unusable after the subsequent reload.

Boot Integrity and Trust Visibility

Table 6: Feature History Table

Feature Name	Release Information	Feature Description
Support for Boot Integrity and Trust Visibility (BIV)	Release 7.4.1	Support for the BIV functionality is now extended to the following Cisco NCS 540 router varaint: • N540X-6Z18G-SYS-A/D • N540X-8Z16G-SYS - A/D • N540X-4Z14G2Q-SYS-A/D

The secure boot first stage is rooted in the chip and all subsequent boot stages are anchored to the first successful boot. The system is, therefore, capable of measuring the integrity of the boot chain. The hash of each software boot image is recorded before it is launched. These integrity records are protected by the TAm. The boot chain integrity measurements are logged and these measurements are extended into the TAm.

Use the **Router#show platform security attest pcr 15 trustpoint ciscoaik nonce 4567** command to view the boot integrity and boot-chain measurements. Given below is a sample output:

```
RP/0/RP0/CPU0:ios# show platform security attest PCR 15 trustpoint CiscoAIK nonce 4567
location 0/RP0/CPU0

Sun Jun 21 03:07:18.394 UTC
Nonce: 4567

+-------------------------------------+
   Node location: node0_RP0_CPU0
+-------------------------------------+
Uptime: 495270
pcr-quote: /1RDR4AYACCBG/wltf4TEwfdUjtjun7S3rXC90eAb0G0ytrYRv3ExwACRWcAAAAAAD8hUwAAAEf/////
AQAAACQAAAALAAAAAQALAwCAAAAgae1J8QIYe06nS2RUx0JYeoG8tM3bqeVdpW7CObwBt+g=
pcr-quote-signature:
EZbzSUge89jSjH8ZqTgKJrZJBopEbd818C+h1Ec780qi7Li1WfCZQPIP6KCDV6HsRCVzLoFijgmlMLoZE2rakQq+/
1TgZOWSLjMY7RbjSFr8z/zbpVI+YLnOG+wytVYWuY33uKHBn/
YWokHwo+qVf7u9aLGhnrXKvRUaFknBiZtQGiyAdis6GbPTToqn0WSN1y6DPh4UHZj1vLVwJsI48mbQUrAyCZrz/
XBHLM38tVJjqSrC0jw/6LF2DDoT5ks0VUFT7sqbysw4F56y+z/IlDBrrRW3GFOY46MOxDxLwSl1/
n6zdoVjiKKeqKOnmhpBh72bJQAdeu/GVOYTrOSy4Q==
pcr-index       pcr-value
  15    oYk8yqrzudIpGB4H++SaV0wMv6ugDSUIuUfeSqbJvbY=

RP/0/RP0/CPU0:ios# show platform security integrity hardware digest-algorithm SHA1 trustpoint
 CiscoAIK nonce 4567 location 0/RP0/CPU0

Sun Jun 21 03:09:14.594 UTC
Nonce: 4567

+-------------------------------------+
   Node location: node0_RP0_CPU0
+-------------------------------------+
TPM Name: node0_RP0_CPU0_aikido
Uptime: 495385
Known-good-digests:
Index   value
  0     3TDUS9iUDCFX3VkICcOnySOQTPA=
observed-digests:
```

```
Index   value
  0     3TDUS9iUDCFX3VkICcOnySOQTPA=
PCRs:
Index   value
  15    1Y3uKqNv1UJQUNZQxmZkiuG4blk=

RP/0/RP0/CPU0:ios# show platform security integrity hardware digest-algorithm SHA256
trustpoint CiscoAIK nonce 4567 location 0/RP0/CPU0

Sun Jun 21 03:09:31.110 UTC
Nonce: 4567

+--------------------------------------+
   Node location: node0_RP0_CPU0
+--------------------------------------+
TPM Name: node0_RP0_CPU0_aikido
Uptime: 495401
Known-good-digests:
Index   value
  0     3TDUS9iUDCFX3VkICcOnySOQTPA=
observed-digests:
Index   value
  0     3TDUS9iUDCFX3VkICcOnySOQTPA=
PCRs:
Index   value
  15    1Y3uKqNv1UJQUNZQxmZkiuG4blk=

RP/0/RP0/CPU0:ios# show platform security integrity hardware digest-algorithm SHA256
trustpoint CiscoAIK nonce 4567 location 0/RP0/CPU0

Sun Jun 21 03:09:43.782 UTC
Nonce: 4567

+--------------------------------------+
   Node location: node0_RP0_CPU0
+--------------------------------------+
TPM Name: node0_RP0_CPU0_aikido
Uptime: 495414
Known-good-digests:
Index   value
  0     y3n/SsvyNb8g3o7FFRGCZwfbs8EGxvMZg/PeN0NA71k=
observed-digests:
Index   value
  0     y3n/SsvyNb8g3o7FFRGCZwfbs8EGxvMZg/PeN0NA71k=
PCRs:
Index   value
  15    oYk8yqrzudIpGB4H++SaV0wMv6ugDSUIuUfeSqbJvbY=
Cisco AIK Certificate used for signing PCR
pcr-quote: /1RDR4AYACCBG/wltf4TEwfdUjtjun7S3rXC90eAb0G0ytrYRv3ExwACRWcAAAAAAD8hywAAAEf////
/AQAAACQAAAALAAAAAQALAwCAAAAgae1J8QIYe06nS2RUx0JYeoG8tM3bqeVdpW7CObwBt+g=
pcr-quote-signature:
qyKbK7ndJbrgxeVnOodLWQzT7++NzrxJ9ERRvJzvTe4+8r6p0HGSepHUhZHzYkXw4DbniHAK0Cs3dwg/
hGKGe4M8Lz+/k682yIjaFYYip0DHMaV2ny/lT7RSqM/6u3j/JZrZv39MaeHa3MyjjonzRf9oe7EBSFAKsa/D54eTR0eFtaxFy/
XdtM0VVQe2JRdoBVxnIBLGiVmGRlVVlmHvwwgX1lAN6e3/soC1Vk3I5gjLldPHUYuJ/
7PTGyAwZsbdeigx8d4ViUUUjSMzK7JISwXa8k4GiPQVLBHtqqR+RA9scmMZTbKLsG3luIWKQeyCtXMYE1VOeW8WQlAvioMICw==
RP/0/RP0/CPU0:ios#show platform security integrity hardware digest-algorith$
Sun Jun 21 03:09:56.794 UTC
Nonce: 4567

+--------------------------------------+
   Node location: node0_RP0_CPU0
+--------------------------------------+
TPM Name: node0_RP0_CPU0_aikido
Uptime: 495427
```

```
Known-good-digests:
Index   value
  0     3TDUS9iUDCFX3VkICcOnySOQTPA=
observed-digests:
Index   value
  0     3TDUS9iUDCFX3VkICcOnySOQTPA=
PCRs:
Index   value
  15    1Y3uKqNv1UJQUNZQxmZkiuG4blk=

RP/0/RP0/CPU0:ios#
```

You can also use Cisco-IOS-XR-remote-attestation-act.yang to fetch the boot integrity over the NETCONF protocol.

The command displays both, the integrity log values and the assurance that these values have not been tampered. These measurements include the following parameters:

- Micro loader hash
- Boot loader hash
- Image signing and management key hashes
- Operating system image hash

```
platform-pid string Platform ID
Event log [key: event_number]: Ordered list of TCG described event log
                               that extended the PCRs in the order they
                               were logged
    +-- event_number   uint32 Unique event number of this even
    +-- event_type     uint32 log event type
    +-- PCR_index      uint16 PCR index that this event extended
    +-- digest         hex-string The hash of the event data
    +-- event_size     uint32 Size of the event data
    +-- event_data     uint8[] event data, size determined by event_size
PCR [index] - List of relevant PCR contents
    +-- index    uint16 PCR register number
    +-- value   uint8[] 32 bytes - PCR register content
PCR Quote binary TPM 2.0 PCR Quote
PCR Quote Signature binary Signature of the PCR quote using TAM-held ECC  or RSA restricted
 key with the optional nonce if supplied
```

Note

- Platform Configuration Register (PCR) 0-9 are used for secure boot.
- Signature version designates the format of the signed data.
- The signature digest is SHA256.
- The signing key is in a Trusted Computing Group (TCG) compliant format.

Note

Use the **show platform security tam** command to view the TAm device details. The following example shows a truncated output of the command:

```
Router#show platform security tam all location all
Mon Apr 15 14:42:34.649 UTC
-------------------------------------------
Node - node0_RP0_CPU0
-------------------------------------------
Device Type            -     AIKIDO Extended
Device PID             -     N540X-12Z16G-SYS-A
Device Serial Number   -     FOC2333NJ0J
Device Firmware Version-     0x24.000b
Server Version         -     3
Server Package Version -     9.4.1
Client Package Version -     9.4.1

Sudi Root Cert:
-------------------------------------------
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            01:9a:33:58:78:ce:16:c1:c1
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: O=Cisco, CN=Cisco Root CA 2099
        Validity
            Not Before: Aug  9 20:58:28 2016 GMT
            Not After : Aug  9 20:58:28 2099 GMT
        Subject: O=Cisco, CN=Cisco Root CA 2099
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:d3:b6:e3:35:7e:0d:3e:f4:67:e5:8a:4e:1a:c6:
        Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Key Identifier:
                38:95:57:0F:34:23:4E:F3:A1:26:20:BA:14:91:C7:41:88:1D:A3:5B
    Signature Algorithm: sha256WithRSAEncryption
         8d:e2:99:a3:ee:31:77:4e:53:16:da:bd:f6:72:a7:58:0d:09:

Sudi Sub CA Cert:
-------------------------------------------
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0a:64:75:52:4c:d8:61:7c:62
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: O=Cisco, CN=Cisco Root CA 2099
        Validity
            Not Before: Aug 11 20:28:08 2016 GMT
            Not After : Aug  9 20:58:27 2099 GMT
        Subject: CN=High Assurance SUDI CA, O=Cisco
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:bd:dc:de:49:67:43:23:a9:51:64:36:11:bc:0e:
```

```
                Exponent: 65537 (0x10001)
            X509v3 extensions:
                X509v3 Key Usage: critical
                    Certificate Sign, CRL Sign
                X509v3 Basic Constraints: critical
                    CA:TRUE, pathlen:0
                Authority Information Access:
                    CA Issuers - URI:https://www.cisco.com/security/pki/certs/crca2099.cer
                    OCSP - URI:http://pkicvs.cisco.com/pki/ocsp

                X509v3 Authority Key Identifier:
                    keyid:38:95:57:0F:34:23:4E:F3:A1:26:20:BA:14:91:C7:41:88:1D:A3:5B

                X509v3 Certificate Policies:
                    Policy: 1.3.6.1.4.1.9.21.1.30.0
                      CPS: http://www.cisco.com/security/pki/policies/

                X509v3 CRL Distribution Points:

                    Full Name:
                      URI:http://www.cisco.com/security/pki/crl/crca2099.crl

                X509v3 Subject Key Identifier:
                    EA:6B:A3:B9:C1:13:97:7E:1B:FB:3A:8D:68:60:07:39:5F:87:48:FA
        Signature Algorithm: sha256WithRSAEncryption
             5c:a9:81:0e:80:01:e1:19:62:a7:77:03:3d:d3:55:d7:d8:49:

Sudi Cert:
-----------------------------------------------
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 29200071 (0x1bd8ec7)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=High Assurance SUDI CA, O=Cisco
        Validity
            Not Before: Sep  5 03:39:36 2019 GMT
            Not After : Aug  9 20:58:26 2099 GMT
       Subject: serialNumber=PID:N540X-12Z16G-SYS-A SN:FOC2333NJ0J, O=Cisco, OU=ACT-2 Lite
 SUDI, CN=Cisco NCS 540 System with 12x10G+4x1G Cu+12x1G AC Chassis
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:ca:2a:8a:b4:87:8b:43:68:17:d3:b2:43:44:ca:

                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Alternative Name:
                0...N.
+.........@917C927B4B340B908703945A7A0A6D14D0207ADB2FB622DFA8C83538FD7E63B5.
B..+.........5.3ChipID=QvZQd9q9psveoAz6QJQeNAAAAAAAAAAAAAAAAAAAAAA=
    Signature Algorithm: sha256WithRSAEncryption
         5b:67:da:2e:e5:d4:07:f2:ff:9c:17:c9:54:78:8b:da:16:df:
```

The boot integrity verification is automatic and the BIOS reports the values to the PCR. The boot integrity verification process consists of the following steps:

1. Report Boot 0 version and look up the expected integrity value for this platform and version.

2. Report bootloader version and look up the expected integrity value for this platform and version.
3. Report OS version and look up the expected integrity value for this platform and version.
4. Using the integrity values obtained from steps 1-3, compute the expected PCR 0 and PCR 8 values
5. Compare the expected PCR values against the actual PCR values.
6. Verify the nonced signature to ensure the liveliness of the response data (this assumes unique nonced are being passed). Note that this signature verification must be performed only with the platform identity verified using SUDI.
7. (Optional) Verify the software image (IOS XR) version is with what is expected to be installed on this platform.

A failure of any of the above steps indicates either a compromised system or an incomplete integrity value database.

Secure gRPC

gRPC (gRPC Remote Procedure Calls) is an open source remote procedure call (RPC) system that provides features such as, authentication, bidirectional streaming and flow control, blocking or nonblocking bindings, and cancellation and timeouts. For more information, see https://opensource.google.com/projects/grpc.

TLS (Transport Layer Security) is a cryptographic protocol that provides end-to-end communications security over networks. It prevents eavesdropping, tampering, and message forgery.

In Cisco IOS XR7, by default, TLS is enabled in gRPC to provide a secure connection between the client and server.

Integrity Measurement Architecture (IMA)

The goals of the Linux kernel integrity subsystem are to:

- detect whether files are accidentally or maliciously altered, both remotely and locally
- measure the file by calculating the hash of the file content
- appraise a file's measurement against a known good value stored as an extended attribute
- enforce local file integrity

There are three components in the Linux kernel integrity subsystem:

- IMA Measurement
- IMA Appraisal
- IMA Audit

Note These goals are complementary to the Mandatory Access Control (MAC) protections provided by SElinux.

IMA Measurement

IMA maintains a runtime measurement list and—because it is also anchored in the hardware Trusted Anchor module (TAm)—an aggregate integrity value over this list. The benefit of anchoring the aggregate integrity value in the TAm is that the measurement list cannot be compromised by any software attack without being detectable. As a result, on a trusted boot system, IMA-measurement can be used to attest to the system's runtime integrity.

For more information about IMA, download the IMA whitepaper, An Overview of The Linux Integrity Subsystem.

IMA Appraisal

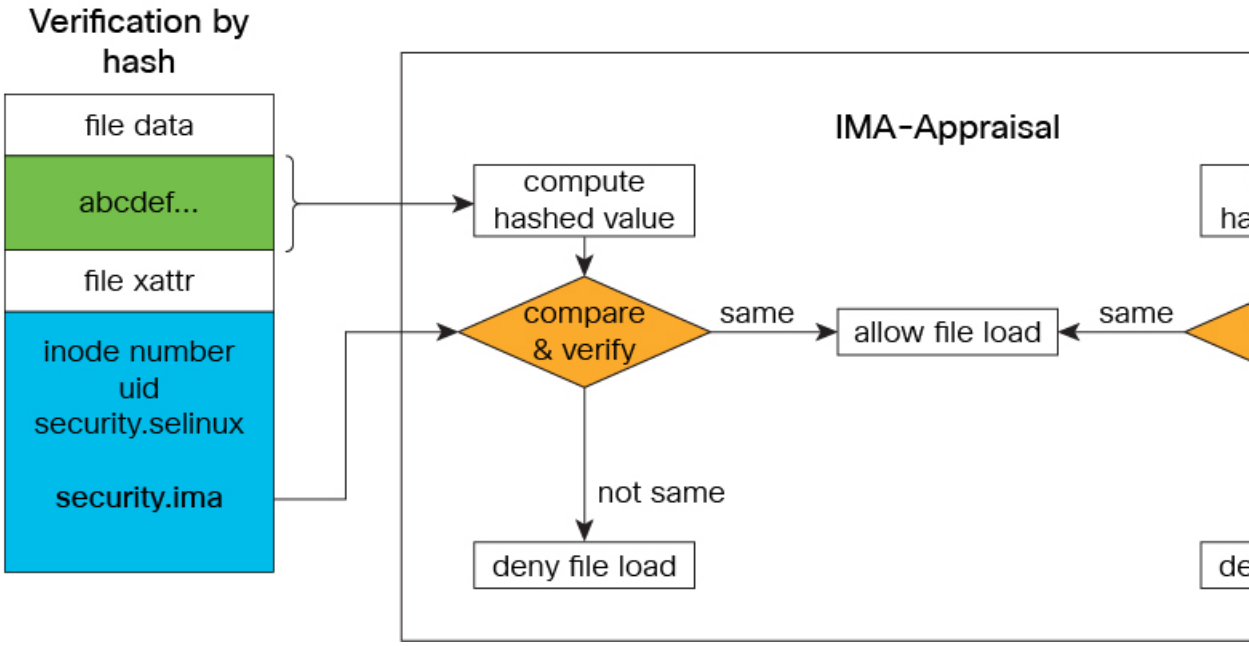
Table 7: Feature History Table

Feature Name	Release	Description
IMA Appraisal	Release 7.4.1	IMA maintains an integrity measurement list for remote attestation. The IMA-appraisal extension adds local integrity validation and enforcement of the measurement against a 'good' value. In this release, IMA policy *enforces* an appraisal instead of simply logging an integrity violation by blocking any operation (open or run) for a compromised executable. This *enforced* mode of appraisal ensures a higher level of trust and security at runtime.

IMA appraisal provides an added runtime security level that can detect if a file has been modified – either accidentally or maliciously and either remotely or locally.

The kernel achieves this by validating the hash measurement of the file against a known good value (KGV). The encrypted KGV in the form of a signature is stored in the file’s extended attribute and enforces local file integrity. The **enforced** mode strictly enforces the file integrity check whenever a file is opened for either reading or executing.

Figure 5: IMA Appraisal



There are three categories of system files that require protection – Linux, XR, and third-party applications.

1. Linux System Files: System files are comprised of Executable and Linkable Format (ELF) binary executables, shared libraries, scripts (such as, Bash, Python, PERL, and Tcl), configuration files, and password files that are part of the Linux distribution packages. Integrity protection of the said files ensures that remote or local modification of the data does not remain undetected and access to such tampered data is either forbidden or logged or both. To guarantee the integrity of these files, they must have a valid IMA signature for the lifetime of the files. Executables and scripts must be appraised and measured. All other immutable files must be measured. Files that don't require appraisal and measurement are runtime files, logs, memory-mapped files like devices, and shared memory objects.
2. XR System Files: XR system files are comprised of XR applications, shared libraries, kernel modules, scripts, data files, configuration files and secret files like keys and user credentials. Integrity of these files must be maintained in order for XR to operate properly. To keep the integrity of these files protected all system files must have a valid IMA signature for the lifetime of the files. Executables and scripts must be appraised and measured. All other immutable files must be measured. Files that don't require appraisal and measurement are runtime files, logs, memory-mapped files like devices, and shared memory objects.
3. Third-party Applications (TPAs): All TPAs are not appraised. There are two types of TPAs:
   - native running applications: For native running applications the system files are installed on the disk from an rpm package or directly copied to the disk. All immutable files are only measured. Executables and scripts must be appraised and measured. All other immutable files must be measured. Files that don't require appraisal and measurement are runtime files, logs, and memory mapped-files like shared memory objects.
   - containerized applications: For containerized applications the system files are packaged in the container image such as docker as part of the filesystem layers. When the container is launched, the system files are only accessible from within the container unless it is bind mounted on the host. In this case, only container image files are measured.

There are other frequently updated files that are created by the IOS XR (Linux, XR) at runtime, such as runtime files, logs, memory mapped files like devices and shared memory objects. These files contain runtime data and logs that are constantly updated by the applications. By default, they do not require an IMA signature and are excluded from appraisal to avoid possible access failure.

In this release, the following files are *not* signed with an IMA key, so they do not have an IMA signature. However, the system still allows their execution:

- Zero Touch Provisioning (ZTP) bash scripts with execute permission
- ZTP bash scripts without execute permission
- Third-party bash scripts without execute permission
- Bash scripts downloaded through file transfer operation like secure copy (SCP) or Secure File Transfer Protocol (SFTP)
- Open Programmability System (OPS) 1.0 scripts, whether downloaded or created on the router

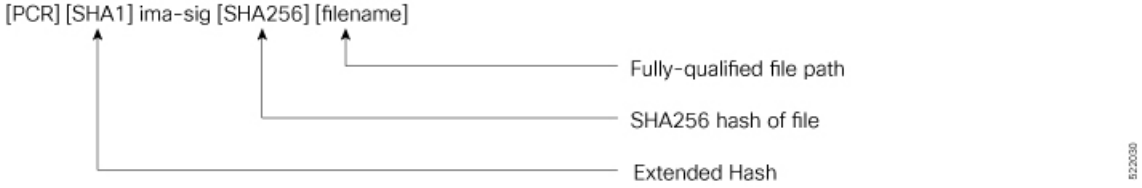
Note In Cisco IOS XR Release 7.4.1, enforced appraisal is enabled only on the XR system files.

IMA Measurement Log

When a file covered by an IMA measurement policy is opened for reading or execution IMA must measure the file by calculating its sha256 hash and record it in the IMA measurement log. To read the integrity log as registered by the IMA subsystem, review the **/sys/kernel/security/ima/ascii_runtime_measurements** file. The columns (from left to right) are:

- **PCR (Platform Configuration Register)** in which the values are registered. This is applicable only if a Trusted Platform Module (TPM) chip is in use.
- **Extended hash** that is stored in the PCR.
- **Template** that registered the integrity value (ima-sig).
- **SHA256 hash** of the file.
- **Filename** that has the fully-qualified file path.

```
[node0_RP0_CPU0:/]$cat /sys/kernel/security/ima/ascii_runtime_measurements | grep "/usr/bin/zip" |
fold -w 100 | more
10 baa85aaf01d7966b54d206fae0f0f628b5c7e6e3 ima-sig sha256:b6b25a5fce5f139a8daa1a902f71092ff810a7d2b
13d71f61f3175df31e51e02 /usr/bin/zip 0302046ebaed8301005d434285f32f4c65840568e8fa08b7b4f06789c7f1c98
c63d9cba8e4f41e35d20591285874682b040fad2419590f1a7574a60efa5ac3b36ee3f3336ed5ed277040d8346f766b49ea1
ae3475ea4867abf6ecbf1ccc045fa08a84078d14fa529caee5c27b0ef4df8694c8d6241b7c630c84a4fe304c345532321b33
d415758031ec411216bb1a16352dc0937cc23ff9f6165c91cd8ce973e21bfb77769ac496ab50ecb3207498c274aae48e5c4e
44ce548af2764d0598e74dce70c918756c7bcfde3c4d55ecab806b55381608920f2289e9c4338dd34bcda6f097c7b76de692
4b252dca325d9c0e8d8eba811d12a89fc4c4a7999f96bc494e7f885bb613dc5b278
```



IMA Audit

IMA audit generates an event log every time it finds a file opened for reading or executing that has a mismatch between the measured file hash and the one stored in the extended attribute.

This data integrity verification event is recorded in the audit log.

There are three reasons an integrity log is recorded in the audit log – invalid signature, invalid hash and missing hash. The audit log has the following key information:

- type - INTEGRITY_DATA - Triggered to record a data integrity verification event run by the kernel.
- pid - Process ID of the calling process that opened the file with integrity verification failure.
- subject - SELinux file context label. SELinux runs in Permissive mode. Any access control violation is only logged in the audit log and the application is still allowed to run.
- op - Operation (appraise_data).
- cause - Reason for integrity verification failure (invalid-signature, invalid-hash, missing-hash).
- comm - Calling process.
- name – Name of the file with full path that was appraised.

The following output shows an instance where the IMA appraisal causes the execution of a tampered binary executable to fail. The integrity violation logged is **Invalid Signature**, the integrity violation log type is **Integrity Data**, and the appraised file is **/usr/bin/zip**.

```
RP/0/RP0/CPU0:NCS-540-C-LNT#run
Mon Apr 29 08:39:26.793 UTC
[node0_RP0_CPU0:~]$cat /var/log/audit/audit.log | grep -i integ | grep zip | fold -w 100
type=INTEGRITY_DATA msg=audit(1714378019.193:866): pid=2236 uid=0 auid=4294967295
ses=4294967295 sub
j=iosxradmin_u:iosxradmin_r:iosxradmin_t:s0 op="appraise_data" cause="invalid-signature"
comm="sh" n
ame="/usr/bin/zip" dev="dm-14" ino=147881 res=0
type=INTEGRITY_DATA msg=audit(1714378019.197:867): pid=2236 uid=0 auid=4294967295
ses=4294967295 sub
j=iosxradmin_u:iosxradmin_r:iosxradmin_t:s0 op="appraise_data" cause="invalid-signature"
comm="sh" n
ame="/usr/bin/zip" dev="dm-14" ino=147881 res=0
```

The following output shows an instance when an audit log was recorded because the file was missing an IMA signature and was opened for either reading or execution. This resulted in a “missing-hash” event log.

```
[node0_RP0_CPU0:/ima-appraisal]$zip --version | head -2
sh: /usr/bin/zip: Permission denied
[node0_RP0_CPU0:/ima-appraisal]$
[node0_RP0_CPU0:/ima-appraisal]$cat /var/log/audit/audit.log | grep -i integ | fold -w 100
type=INTEGRITY_DATA msg=audit(1714500558.187:556): pid=52560 uid=0 auid=4294967295
ses=4294967295 su
bj=iosxradmin_u:iosxradmin_r:iosxradmin_t:s0 op=appraise_data cause=missing-hash comm="sh"
 name="/us
r/bin/zip" dev="dm-11" ino=1507384 res=0
```

IMA Policy

The IMA policy is not user-defined and is created by default. It contains a policy rule set that defines exactly which files on the file system should be measured or appraised.

Each policy rule must start with one of the following directives:

- measure: Perform IMA measurement
- dont_measure: Exclude from IMA measurement
- appraise: Perform IMA appraisal
- dont_appraise: Exclude from IMA appraisal

Note IMA policy is protected at runtime – it cannot be read or modified.

Verifying the IMA Appraisal "enforce" Mode

To display the content of the IMA appraisal mode, query the kernel command line and look for "ima_appraise=enforce".

```
$ cat /proc/cmdline
```

To query the content of the IMA measurement logs:

```
$ cat /sys/kernel/security/ima/ascii_runtime_measurements
```

To display the total number of files measured:

```
$ cat /sys/kernel/security/ima/runtime_measurements_count
```

To display the total number of integrity violations:

```
$ cat /sys/kernel/security/ima/violations
```

To access other user space interfaces in sysfs that are specific to the cisco_ima measurement:

```
$ ls /sys/kernel/security/cisco_ima
```

IMA Signatures

The IMA appraisal provides local integrity, validation, and enforcement of the measurement against a known good value stored as an extended attribute—security.ima. The method for validating file data integrity is based on a digital signature, which in addition to providing file data integrity also provides authenticity. Each file (RPM) shipped in the image is signed by Cisco during the build and packaging process and validated at runtime using the IMA public certificate stored in the TAm.

All RPMs contain Cisco IMA signatures of the files packaged in the RPM, which are embedded in the RPM header. The IMA signature of the individual file is stored in its extended attribute during RPM installation. This protects against modification of the Cisco RPMs.

The IMA signature format used for IMA can have multiple lines and every line has comma-separated fields. Each line entry will have the filename, hash, and signature as illustrated below.

- File – Filename with the full path of the file hashed and signed
- Hash – SHA256 hash of the file
- Signature – RSA2048 key-based signature

How Trustworthiness is Implemented

The following sequence of events takes place automatically when the Cisco routers that support the IOS XR7 operating system are powered up:

1. At power UP, the micro-loader in the chip verifies the digital signature of BIOS using the keys stored in the Trusted Anchor module (TAm). The BIOS signature verification is logged and the measurement is extended into a PCR.
2. The BIOS then verifies the signature of the boot-loader using keys stored in TAm. The boot-loader signature verification is logged and the measurement is extended into the PCR.
3. If the validation is successful, the BIOS launches the bootloader. The bootloader uses the keys loaded by the BIOS to verify the sanctity of the kernel, initial RAM disk (initrd) file system, and grub-config file. Each verification operation is logged, and the PCR in TAm is extended.
4. The initrd is loaded to create the initial file system.
5. The kernel is launched and the kernel keyrings are populated with the appropriate keys from the TAm.
6. The init process is launched. Whenever an executable or a shared library is invoked, the IMA kernel hook validates the signature using the certificates in IMA keyring, which is then used to validate the signature attached to the file.
7. The Cisco IOS XR7 RPM is installed with the signed verification. The results of RPM verification are logged.
8. Cisco IOS XR7 processes are launched with IMA measurement.
9. TAm services are launched.
10. Cisco IOS XR7 application runs the initial admin user configuration and stores the credentials into TAm secure storage.

    Manual provisioning of user credentials is now complete.

The Cisco routers perform the above steps, which is a holistic approach to integrate trust. Trust begins in hardware, next the system verifies the trustworthiness of the network operating system, after bootup, the system maintains trust at runtime, last, the system visualizes and reports on trust. You can verify the boot status by executing the following command:

```
Router#show platform security integrity log secure-boot
Fri Apr 12 17:13:43.867 UTC

+--------------------------------------+
   Node location: node0_RP0_CPU0
+--------------------------------------+
Secure Boot Status: Enabled
```

Understanding Key Concepts in Security

Attestation

Attestation is a mechanism used to attest the software's integrity. The verifier trusts that the attested data is accurate because it is signed by a TPM whose key is certified by the CA.

Attestation Identity Key

An Attestation Identity Key (AIK) is a restricted key that is used for signing attestation requests.

Bootloader

The bootloader is a piece of code that runs before any operating system begins to run. Bootloaders contain several ways to boot the OS kernel and also contain commands for debugging and modifying the kernel environment.

Certificates and Keys in TAm

All database keys are signed by the KEK. Any update to the keys requires the KEK or PK to sign in, using time-based authentic variables. Some of the keys on the database are:

- Image signing certificate: This is the X.509 certificate corresponding to the public key and is used for validating the signature of grub, initrd, kernel, and kernel modules.
- IOS-XR Key: A public key certificate signed by the KEK. This key is common to all Cisco NCS 540 Series routers and is used to sign GRUB, initrd, kernel and kernel modules.
- RPM key: Used for signing RPMs.
- IMA public key certificate: Used for Integrity Measurement Architecture (IMA), and used to validate the IMA signature of the files.
- BIOS or Firmware Capsule Update key: Used to sign the outer capsule for BIOS or firmware updates. It is the same as the secure boot key.
- Platform key (PK) and Key Enrollment Key (KEK): These are public keys and certificates used to manage other keys in the TAM.
- LDWM Key: In the Cisco IOS XR7, the LDWM key is stored in the hardware trust anchor module and is used for validating the BIOS.

Golden ISO (GISO)

A GISO image includes a base binary artifact (an ISO) for the Linux distribution that is used on the server fleet, packages, and configuration files that can be used as a base across all servers.

The GISO image for Cisco IOS XR7 software contains the IOS XR RPMs and third-party RPMs.

GRand Unified Bootloader (GRUB)

GNU GRUB (or just GRUB) is a boot loader package that loads the kernel and supports multiple operating systems on a device. It is the first software that starts at a system boot.

Hash Function

A hash function is any function that is used to map data of arbitrary size onto data of a fixed size.

Initramfs

Initramfs, a complete set of directories on a normal root filesystem, is bundled into a single cpio archive and compressed with one of the several compression algorithms. At boot time, the boot loader loads the kernel and the initramfs image into memory and starts the kernel.

initrd

initial RAM disk is an initial root file system that is mounted before the real root file system is made available. The initrd is bound to the kernel and loaded as part of the kernel boot procedure.

JTAG

JTAG is a common hardware interface that provides a system with a way to communicate directly with the chips on a board. JTAG is used for debugging, programming, and testing on embedded devices.

Nonce Value

A nonce value is an arbitrary number that can be used only once in a cryptographic communication. It is a random or pseudo-random number that is issued in an authentication protocol to ensure that the old communications are not reused in replay attacks.

Platform Configuration Register (PCR)

A PCR is a shielded register or memory region large enough to hold the contents of a hash operation. A PCR is initialized to a well-known value at power-up, and typically cannot be reset.

PCR Extend

The only way to change the value held in a PCR is to perform an “extend” operation, which is defined as:

```
PCR[x]new = hash ( PCR[x]old || hash ( measurement value ) )
```

Trust Anchor module (TAm)

The Cisco Trust Anchor module (TAm) helps verify that Cisco hardware is authentic and provides additional security services.

Trusted Platform Module (TPM)

A Trusted Platform Module (TPM) is a specialized chip on an endpoint device that stores RSA encryption keys specific to the host system for hardware authentication. This key pair is generated by the TPM based on the Endorsement Key and an owner-specified password.

Root of Trust for Storage

TPM 2.0-compliant Platform Configuration Registers (PCRs) form the Root of Trust for Storage.