OPEN
Compute Project

BEST PRACTICES FOR FIRMWARE CODE SIGNING

AUTHORS:

Kenneth Goldman, IBM Research Division

Elaine Palmer, IBM Research Division

Tim Block, IBM Cognitive Systems Division

Christopher Engel, IBM Cognitive Systems Division

David Heller, IBM Cognitive Systems Division

Executive Summary

What protects firmware in a supply chain, along its journey from an authorized developer, all the way to the device in which it ultimately runs? How does a device (or system administrator) determine that firmware has come from a legitimate supplier, or if it was accidentally or intentionally modified along the way?

The answer recommended by the National Institute of Standards and Technology[1], is to apply digital signatures to the firmware at its origin, then to verify those signatures at its destination. What then, are the consequences of an intentional attack on or accidental misconfiguration of a supplier's digital signing systems? At best, the firmware might be rejected at the destination. At worst, an attacker can supply malicious firmware that goes undetected.

In its focus to support the growing security demands on the compute infrastructure, the Open Compute Project (OCP) has published this white paper. The goal is to provide specific recommendations to firmware suppliers on how to secure the systems that apply digital signatures to firmware that runs in servers and devices throughout the compute infrastructure.

Table of Contents

1 Introduction

The recent National Institute of Standards and Technology (NIST) white paper "*Security Considerations for Code Signing*"[1] identifies the need for a secure software supply chain, which includes the protection of firmware throughout secure distribution and update processes. NIST recommends that platform manufacturers deliver digitally signed firmware and associated verification support to provide assurance of the integrity and authenticity of their delivered code. Verification of the authenticity and integrity of firmware is clearly necessary at delivery, but it is also important at boot time of a platform, and every time the firmware is updated. What, then, are the consequences of an intentional attack or accidental misconfiguration of a manufacturer's digital signing infrastructure?

1.1 Scope

This paper focuses on policies and best practices for implementation and administration of signing server facilities to provide digital signatures for firmware images. The paper provides one template for best practices, while intending to allow innovation and advancement for platform manufacturers and their unique supplier/partner relationships.

These facilities and practices can be logically extended beyond signing of firmware builds to include other software and metadata. For example, it is advisable to sign application software, OS drivers, device drivers, release notes, delivery manifests, and configuration information for installation packages. Out of scope for this paper are chain of trust requirements, firmware development practices, supply chain and manufacturing processes, as well as usage of certificates, certificate authorities, and trusted application stores. Also out of scope for this paper but needing attention, is establishing trust in audit logs. Are the audit logs protected against tampering? Can the audit logs be falsified from the start? Are automated procedures in place to detect anomalous activity related to signing server audit logs?

2 Firmware Signing in the Open Compute Environment

The Open Compute Project (OCP) Security workgroup lists a number of in-scope threats, including "Execution of unauthorized firmware", "Compromised private signing keys", and "[Firmware] update mechanism compromised".[2] These threats, as well as other in-scope threats, highlight the need for a secure supply chain that supplies authentic, unadulterated firmware for OCP-Accepted™ and OCP-Inspired™ products.

Although there are potentially many components and participants in a firmware supply chain, one mandatory component is a secure signing server. It is used to digitally sign firmware. After the firmware is signed in the signing server, it travels through the supply chain, eventually reaching its destination OCP platform (or device). There, the platform verifies the signer's digital signature. If the verification succeeds, the platform installs and / or executes the firmware. If the verification fails, because the wrong key was used, or the signature was modified, or the firmware was modified somewhere along the way, a platform policy dictates the next action, whether to reject the failing firmware, isolate it, or allow it to run.

Therefore, the security of the signing server is critical, because it applies the “protective wrapper” (digital signature) that protects the firmware on its journey from the supplier to the OCP product. For this reason, the signing server must be configured securely, it must be placed in a secure environment, and there must be safeguards against its misuse by associated administrators and users.

This paper identifies best practices for implementation and usage of a firmware signing server in the context of protecting firmware destined for OCP devices. It also includes pointers to open source signing tools and a signing server framework that have provided the basis for development of these best practices across several production environments. Finally, it relates these practices to the OCP tenets of efficiency, scalability, openness, and impact.

3 Background

This section is provided for those unfamiliar with the terms “signing keys,” “imprint keys,” and “hardware security modules” (also referred to as “HSMs”). For those already familiar with those terms, the recommendations begin in Section 6.

3.1 Signing Keys

Every platform (and device) manufacturer wants to ensure the integrity of their own firmware and digitally sign it with their own keys. In this paper, these keys are called signing keys. Signing keys that sign firmware for production level firmware are called production keys. Signing keys used to sign development level firmware are called development keys. Another type of key, called an imprint key, is described below.
Proprietary platforms typically allow for firmware signing by a single authority, controlled by the manufacturer, and make it difficult, if not impossible, to establish a different authority. Open platforms, however, must provide

a way for an end-user or Original Design Manufacturer (ODM) to install a different authority, and sign platform firmware with the authority of their choosing.

To support this change in authority, open platform vendors must provide a way to change the authority at the machine level (to allow users to install a key of their choosing). When the identity of the recipient is known in advance, and when initialization can be performed during manufacturing, the open platform vendor can install a “temporary transport key” as described in [3]. This key is known only to the two parties involved (platform vendor and end-user/ODM). However, when the identity of the recipient is unknown, the open platform vendor must provide a way to ship the machine in an “open state”, to allow the installation of a new key in a secure manner. This may be done by establishing a special “imprint key” where the private part as well as the public part of the key are advertised and well-known. This enables the end-user to transition the platform key to the key of their choosing, without disabling the secure boot function of the machine. This transition establishes ownership of the platform firmware. From that point on, the new owner controls what keys and firmware are allowed onto the platform.

Imprint keys also simplify development and manufacturing, since anyone can sign development-level firmware with these keys. This allows for flexibility in the tuning of manufacturing tests, and allows developers to update and test their code without needing a full firmware build process, all without the need to override platform secure mode functionality in any way. When a machine is shipped in a fully open state, the imprint keys are installed and anyone can install test firmware signed with these keys.

Imprint keys must not be used for production mode.

If a system is shipped from a platform manufacturer with imprint keys installed rather than production level root keys, the system must be viewed as running with a set of well-known default keys, and it is vulnerable to exploitation. Systems must be transitioned to production level keys for customer environments. The System Access Administrator must work with the platform manufacturer to ensure that a key transition process is utilized, so that a hardware-based chain of trust can be enabled as part of any code assurance functionality.

It is worth noting that use of imprint keys provides a mechanism to ship systems without needing to exchange shared secrets between the platform manufacturer and recipient of the system and is therefore predicated upon a trustable supply chain. Installation of rogue firmware or keys by an intermediary would be detectable through tamper-protected logging and would be correctable via a system reflash operation.

3.2 Hardware Security Modules

Typically, the private half of production keys is protected by a hardware security module (HSM) or equivalent protected storage internal to the manufacturing facility of the key owner.

The “Best Practices Template” as provided in this paper refers to an HSM as a required physical device. This may be viewed by some as an implementation specific detail. In that spirit, one could substitute the following set of objectives or requirements for a private firmware key repository for that of a physical HSM device.

- Ability to withstand both physical and logical attacks
- Perform cryptographic operations for creation of key pairs and secure processing of data (signing operations, authenticity & integrity checks) reliant on those keys
- Protect the private keys (even in the face of adversarial destructive analysis)
- Provide a set of secure APIs to enable signing framework interaction
- Run only programs that it is supposed to run, with confidence that those programs are not modified
- Remotely distinguish between the device / application and impostors
- Assure that keys are used only by those authorized to use them
- Assure that keys are used for the intended purpose, e.g., code-signing keys only sign code, not certificates

Note: The specific HSM referred to in this document makes use of a cryptographic coprocessor architecture that generates a set of master keys that reside only in the coprocessor. This architecture enables HSM cloning as well as wrapped key storage outside of the HSM. Other HSM or equivalent protected store options may not have the same master key concept, but may have other keys requiring similar key backup considerations.

4 Firmware Signing Process

4.1 Goals

- Support remote code signing and project audit capability
- Provide a simple and secure process that can be used by internal build processes and by contracted suppliers to sign code.
    - Use existing open-source tools wherever possible
    - Support multiple machine architectures (e.g., x86_64, ppc64le, ARM)
- Validate authentication of person requesting the code signing

    - Accomplished, for example, with a two-step operation of ssh keys and ssh key passphrase used to access signing server. The ssh key passphrase MUST be entered by the signer for each request and not stored in a file.
- Validate authority of person requesting the code signing to the project keys requested (or even finer granularity e.g., to project keys for a specific use case only)
    - Accomplished, for example, with ssh keys used to access signing server which is correlated to a specific signer. Then the signer access rights are verified against the project configuration
    - Encrypted password for HSM card may also be used to validate authority
- Log all signing requests for later auditing. Project auditors can retrieve the logs (pull model) or can be setup to receive them periodically (push model)
- Send notifications to project auditors for all signing requests
- Automate access removal (e.g., through LDAP lookup) of signer by signing framework.

4.2 Generic Signing Server Architecture

The signing framework design limits access to the signing server by firmware signers via isolated access control processes. These processes include appropriate authentication and authorization procedures.

Figure 1 illustrates the flow of a typical signing request. Firmware signers use secure transfer protocols to upload a request to the signing server access control facility, which is monitored by the signing server framework. When the framework sees a new request, it performs all required validations and authorizations, and if successful, it then carries out the signing request. When the framework completes the request, it stores the result back in the access control facility, ready for the signer to download via a secure transfer.

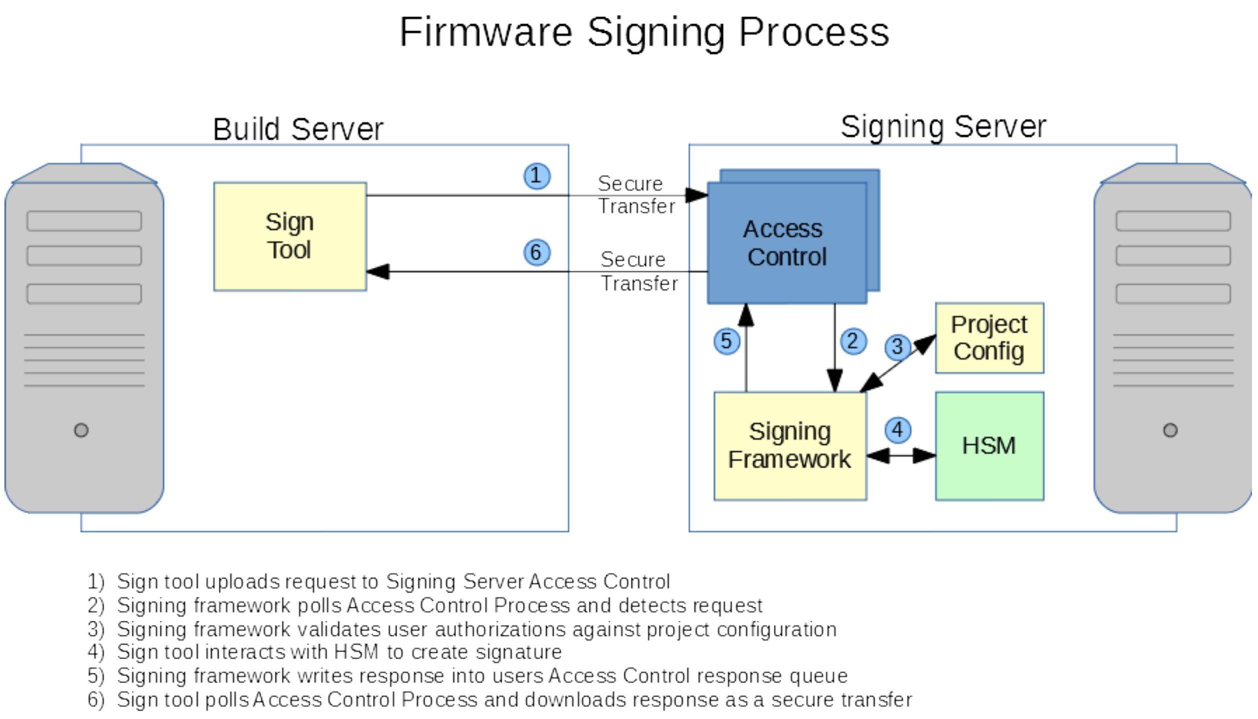

Firmware Signing Process
Build Server
Signing Server
Sign Tool
1
6
Secure Transfer
Secure Transfer
Access Control
5
2
3
Project Config
Signing Framework
4
HSM
1) Sign tool uploads request to Signing Server Access Control
2) Signing framework polls Access Control Process and detects request
3) Signing framework validates user authorizations against project configuration
4) Sign tool interacts with HSM to create signature
5) Signing framework writes response into users Access Control response queue
6) Sign tool polls Access Control Process and downloads response as a secure transfer

Figure 1

4.3 Signing Server Roles

The following roles have been defined such that access to the signing server(s) is limited as follows (more details on these roles can be found in Section 6.4):

- **Signing server system administrator:** has signing server root access and HSM security programming interface (SAPI) access. Backup and clone HSM in conjunction with Master Key Holders.
- **Master Key Holders:** have temporary root access & HSM SAPI access, removed after HSM Master Key generation, before signing key generation
- **Signing framework developer/project administrator:** has signing framework source and configuration file change access
- **Signing project auditor:** has audit project access and configuration(read only)
- **Firmware signers:** have signing project signature permissions

5 Threats and Vulnerabilities

5.1 Threats & Attacks

The following threats and attack vectors should be considered in a signing server design. An attacker may attempt to:

- Steal or expose private firmware signing keys
- Delete or corrupt private signing keys, or physically destroy a secure container holding the keys
- Steal an HSM access key or HSM master password
- Steal a signing server access key, or a password protecting such a key
- Trick a user into exposing a signing server access key or password
- Trick an administrator into authorizing another signer
- Trick a signer into sharing their credentials
- Share signing credentials to be helpful but subverting the audit process
- Send bad passwords to lock out an authorized user
- Inject or submit unauthorized payload for signing
- Inject an unauthorized signature into the firmware packaging process
- Inject unauthorized code (i.e. a rogue program) into the build server or signing server
- Delete or modify audit logs related to the build or signing operations
- Use a backup key to create a rogue HSM with legitimate credentials

5.2 Threat Actors

The following threat actors (attackers) should be considered in a signing server design:

- An external actor gaining sufficient access, via network or physical presence, to mount any of the aforementioned attacks.
- A bad actor internal to the signing organization (or company), having general network access but not authorized to the access signing infrastructure, able to mount any of the aforementioned attacks.
- A bad actor from a partner, customer or vendor relationship, gaining sufficient access to mount an attack, or subvert the code signing process through opportunities presented by the relationship.
- A trusted actor gone rogue, knowingly compromising or subverting the signing processes, or leaking credentials or other sensitive information necessary to compromise or subvert.
- A trusted actor unknowingly hacked or compromised, tricked into subverting the signing processes, or leaking credentials or other sensitive information.

5.3 Natural or Intrinsic Threats

The following threats, arising naturally or inherent to vulnerabilities in the signing infrastructure, should be considered in a signing server design:

- The HSM, server or storage device containing private signing keys may be vulnerable to destruction or exposure by fire, flood or other natural disaster.
- Backup or offsite storage copies of signing keys may be vulnerable to exposure by a natural disaster.

5.4 Vulnerabilities

- The signing server system administrator is a system trust anchor. A compromised administrator can install programs that remove authentication and authorization checks, tamper with audit logs, or defeat other security measures. The administrator could grant signing authority to unauthorized users and may have access to encrypted passwords in the access control facility and project files that could be replaced, and thus could impersonate any signer.
- The signing framework developer is a system trust anchor. A compromised developer can, for example, rewrite the framework to remove authentication and authorization checks, or tamper with audit logs.
- The signer's credentials are project trust anchors. If the signer holds a password (authorizing a signing operation or access to the signing server), the password is vulnerable to theft or exposure. If the signer holds a private key authorizing such operations, the key is likewise vulnerable. If the key is stored in an encrypted file or container (a best practice), the file is vulnerable to theft and potentially a brute-force (password guessing) attack to expose the key.
- The firmware builder is a trusted entity. The builder has the authority to sign any image without the need for any approvals and as such must be trusted. Project auditors will receive notification and can audit the signing logs after the request has been completed.

6 Firmware Signing Server Design Best Practices

6.1 Network Security

Build server(s) to signing server(s) communications

- Keep all private signing keys and signer passwords in the signing server
- Follow network security best practices to provide end to end secure communications across the sign tool / sign framework interface.
    - Secrets such as encrypted signer passwords may be passed across the secure interface.
    - Without this protection, a man-in-the-middle attacker could modify what is being signed to cause a trusted builder to sign a compromised binary belonging to the attacker.
- Use industry standard encryption algorithms and strengths deemed adequate for protecting sensitive information on the build server / signing server connections. AES with a minimum 128 bit encryption key length is recommended.
- Establish a subnet firewall for the signing server to limit inbound and outbound traffic
    - Allow only connections to signing agent via predefined IP addresses and port numbers or agreed upon equivalent (e.g., don't allow any incoming connections and have the server poll)
    - Require administrator authentication for other connections
    - Examples:
        - Pass port 53 (DNS) from the signing server to the Intranet
        - Allow port 22 (SSH) from the intranet to the signing server
          Note: Allowing general remote login is risky. Consider setting up sshd to only accept signing request
        - Allow UDP port 123 (NTP)

6.2 Workplace & Physical Security

- Appropriate physical controls to prevent unauthorized physical access, damage or interruption of the build and signing servers shall be implemented and maintained.
- Physical security perimeters (e.g., fences, walls, barriers, cages, guards, gates, electronic surveillance, physical authentication mechanisms, reception desks, and security patrols) shall be implemented and maintained as appropriate.
- General access areas where unauthorized persons may enter the premises shall be monitored, controlled and, if possible, isolated from controlled access areas containing the build and signing server facilities.
- The signing server facility must be physically located in a Controlled Access Area and shall restrict access by users and support persons.

6.3 Authorization, Identification and Access Control

The Platform Manufacturer must maintain appropriate logical access control measures designed to protect system and application integrity. These measures are intended to prevent unauthorized access to build and signing servers used to provide firmware images for subject systems.

- Access controls must ensure individual accountability is maintained.
- Access controls must authenticate the identity of the user (e.g. passwords, pass phrases, digital certificates, smart cards, trusted platform module, etc.).
- Access controls must ensure the correct relationships between user identities and operational roles. If multiple signing projects are hosted on the same server, enforce isolation between equivalent roles for different signing projects.
- Access authorization must be based on valid business need.
- A process must exist for revoking access and must include a defined time frame for removal. The removal process must revoke access when an individual's business need ends (e.g., end of employment, change in job responsibilities).
- A directory access protocol server, such as an LDAP server, may reside locally on the signing server system, or accessible as a remote trusted server, to authenticate signers
- Name resolution of build and signing server is required i.e., reverse IP lookup can be used for name validation
- Accounts must auto-logoff if inactive for a predetermined number of minutes.

6.4 Operational Security

To ensure integrity of the signing process, broad or sweeping authority to perform critical operations should not be granted to a single actor or an unacceptably small group of actors. Granting broad authority increases the risk of compromise (or mistake) by any single actor, and prevents effective checks and controls from being implemented. Instead, roles and responsibilities should be divided, with a clear understanding of the authority, and the vulnerabilities, associated with each role.

Roles & Responsibilities:

- Signing Server System Administrator
    - Provides support of signing server system and OS updates/patches
    - Has local console root login access to signing server framework

  - Carries out system / framework backup and recovery procedures
  - Controls signing requester access to signing server
  - Has HSM Security Application Programming interface (SAPI)administrator access
  - Must NOT have access to the HSM master keys
- Master Key Holders (more generally, a key backup role)
  - HSM SAPI administrator access during master key setup phase
  - Hold copies of the HSM master keys
  - Must NOT have access to signing keys
- Signing Framework Developer / Project Administrator
  - Maintains framework and signing programs
  - Maintains project configurations
  - Has framework access through access control facility

    Note: To reduce the trust level required for this role, one approach is to maintain the framework source in a source code repository. Access is setup such that the developer can only make changes to the repository source, which can be audited by the Signing Project Auditors. After the changes are in the source repository, then the Signing Server System Administrator downloads and rebuilds the framework. In this manner, less trust is needed in Signing Framework developer.
- Signing Project Auditor
  - Monitors project audit logs and configuration
  - Archives project logs on, for example, a 90 day rotation (assumes some limited audit log storage capacity)
  - Requests add/remove of projects and signers
  - Monitors changes to framework and configuration files done by framework developer
  - Has framework access through access control facility
- Firmware Builder
  - Initiates an automated firmware build process
  - May have the opportunity to modify build scripts or tooling, or subvert other security controls affecting the automated build process
  - Automated build process is trusted to build the correct executable from source code (verified using the build process logs)

- Firmware Signers
    - Small controlled list of users with authority to request production code signing
    - Unique IDs to enable correlation with signing requests
    - Unique passphrase that is NOT stored in a filesystem
    - Has framework access through access control facility
    - May be the same individual as Firmware Builder

      Note: The signer should be a person with knowledge of the firmware being signed. Every developer should not have authority to sign. In fact, the ideal signer is perhaps not a developer at all. It may be the test department, the QA person with the final authority to ship the code after all testing is complete.

6.5 Password Management

- For build and signing servers, controls must be enforced to create strong passwords of at least 8 characters with alpha and non-alpha characters.
- Systems and applications may want the ability to lockout the user after 5 unsuccessful logon attempts depending on the design in place.
- Signers HSM SAPI account passwords will be encrypted.
- Signers will have command support to change their own HSM SAPI account password without administrator intervention.
- Signer HSM SAPI account passwords will have expiration dates as set by the system administrator. (Once per year or immediately in the event of user removal or role changes for an existing signer.)
- Signer HSM SAPI authentication may be supplemented by two factor authentication, required at the point of signing

6.6 Private Encryption/Signing Key Management

- The private firmware keys MUST be Hardware Security Module (HSM) protected
    - Depending on the HSM architecture, all firmware signing keys may be encrypted and stored outside the HSM, or the keys may be kept internal to the HSM device. In either case, the HSM architectures allow for some form of secure transfer of private keys to another HSM device for backup. Regardless of the architecture used, secure backups of the private firmware keys must be maintained outside the HSM device. These keys must be backed up on disk or removable media to allow copying to a new system for disaster recovery procedures.
    - Similar to master keys, keep protected copies on and off site.

- The HSM master keys must be protected from system failure and compromise.
- Upon creation of Master Keys for the HSM, save the key share information (each key may have multiple parts) to removable media (NOT the system hard drive) as well as write down the values. Keep protected copies on and off site. This can be done in parts so that no one person can recreate the master keys. This information can be used to clone a Master Key in another HSM if the HSM were to fail or be destroyed. If the Master Key is lost, there is no ability to recover the signing keys.
- At a minimum, all parts of each key must go to each of at least two individuals for safe keeping, external to the signing server site. This protects against a site facility disaster but not against a compromise situation, since either key owner has complete key information. Ideally, parts of each key would go to separate individuals each with their backups if resources allow for such.
- All parts of a given key must be brought back together to recover a given key.
- NIST SP800-57 Part 1 Rev.4 suggests one to three years originator usage period for private signature keys.[4]
- Encrypted signing keys may be stored outside of the HSM and can therefore be decrypted if the HSM master keys are known to more than one party. Therefore, for maximum security, a distinct HSM should be used for each ODM customer. At a minimum, in the best interest of all parties, the sets of firmware signing keys must be distinct for each ODM customer.

6.7 Audit Logging and Review

- The platform manufacturer must implement audit logging that will record activity in build and signing servers.
- Logging can be implemented using device (e.g., HSM), operating system, or application processes and/or procedural methods
- Logging methods must capture all attempts to access devices, systems or applications, whether privileged, administrative, or unknown users, and those logs must be available for review on demand.
- Minimum logging requirements:
    - Clear audit trail of any key pair creation
    - Requester's ID, date, time, request origination
- Signing requests
    - Requester's ID, date, time, request origination, project name, hash being signed, the resultant signature, and the public key.
- Password changes
- Management Domain (e.g., Integrated Management Module (IMM)) logs will indicate system activity and status
    - Includes logs for under the covers access where HSM lives

- Audit logs should be reviewed on a regular basis for anomalous behavior. Reviews must be done consistent with the audit log storage capacity of the signing server framework to insure logs do not get overwritten.
    - Audit log capacity should be capable of storing at least a 90-day window. As a result, review and archiving of the audit logs must be done on timetable appropriate for the storage capacity window.
- Audit logs should be tamper evident to support detection of unauthorized edits.

6.8 Security Advisory Patches

- The platform manufacturer must have a process for receiving security advisories and assessing the threats against the build and signing servers, relevant applications and networks.
- The platform manufacturer's process must include in the assessment of threats, the severity assigned to the advisory, and the risk of implementing the patch.
- The platform manufacturer must install security advisory patches in a timely manner and ensure that corrective action is taken for advisories impacting systems used for building and signing firmware images.

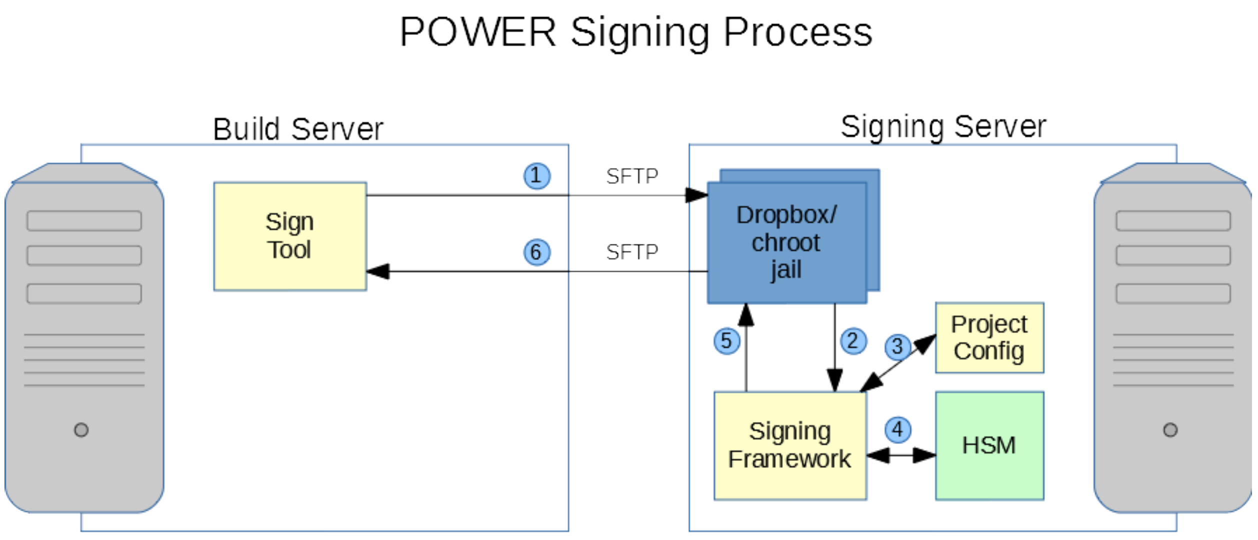
  Note: keeping the signing server off the network may be more secure such that patching is never done to avoid the potential for adding additional attack vectors through the network.

6.9 Security Incident Management

- The platform manufacturer will have a security incident response and notification process that will be used to address incidents impacting build and signing servers.
- Security incidents include:
    - Unauthorized access to a build or signing server
    - Loss of removable media, or portable hardware devices containing private keys related to subject firmware images.
    - Breach of firmware private keys

7 Open Source Signing Tools

The Secure and Trusted Boot firmware in OpenPOWER systems requires an open, efficient, and flexible signing framework that can be easily integrated with build systems at IBM® and other vendors. Figure 2 below is a high level illustration of the open source components developed to meet those requirements, as well as a typical signing request flow.



1) Sign tool uploads request to dropbox via SFTP
2) Signing framework polls dropboxes and detects request
3) Signing framework validates user authorizations against project configuration
4) Sign tool interacts with HSM to create signature
5) Signing framework writes response into users dropbox
6) Sign tool polls dropbox and downloads response via SFTP

Figure 2

7.1 Secure Boot Firmware Signing Tools

Tools enabling the signing of open-source firmware should themselves be based on open-source code, extending the same level of transparency, security and flexibility inherent in the firmware to the code signing. The following tenets and best-practices should be adhered to for code-signing tools:

- Should support signing in all environments and processes required for production, development and test; i.e. it should be possible to use the same tool for everything.

- Should support signing as part of the firmware build process, or externally to the firmware build process, as required for production, development or test.
- Should support the use of different key management systems, ideally in a pluggable manner, as required for production, development or test. That is: the tool should not be dependent on any particular HSM, signing server or keystore.
- Should not rely on a process that requires private signing keys to be exposed during production operations (although this may be acceptable for test or development). Instead, the tool should be able to request a signature from a dedicated signing server or HSM that can sign without exposing the private key.
- Should support authentication of sufficient granularity (e.g. user-centric, endpoint-centric) to identify and authorize the originator of any signing operation (signature generation or signature retrieval).
- Should support in-situ verification; that is, should be able to check the validity of a signature immediately upon generation or retrieval, during the signing process.
- Should support verification of signing keys against an independent reference, such as a public key hash or certificate, delivered by a channel separate from the one used to retrieve private keys or signatures.
- Should be free of security vulnerabilities or weaknesses, as determined by industry standard Code Scanning and Threat Modeling practices, such as
  - https://csrc.nist.gov/publications/detail/sp/800-154/draft
  - https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html
  - https://www.owasp.org/index.php/Static_Code_Analysis

7.2 Firmware Signing Utility

An example utility for signing firmware components for OpenPOWER Secure and Trusted Boot, adhering to the above tenets,

is available at https://github.com/open-power/sb-signing-utils

with companion documentation at https://github.com/open-power/sb-signing-utils/wiki .

This tool is designed for integration into the op-build environment for signing operations done during firmware build, or for standalone signing outside of op-build.

7.3 Secure Boot Signing Server Framework

A Secure Boot signing server framework is available at https://github.com/open-power/sb-signing-framework

with companion documentation at https://github.com/open-power/sb-signing-framework/wiki .

This framework is designed to run on systems that meet the requirements described in Section 6 “Firmware Signing Server Design Best Practices” above.

8 OCP Tenets

The OCP has defined four tenets to follow when designing products for the compute infrastructure: efficiency, scalability, openness, and impact.[5]

8.1 Efficiency

One goal of a firmware signing server is to minimize latency in the firmware build process when signing is requested. A balance must be established between security considerations for strong authentication of signing requests and the speed at which intervening process steps can be completed. These steps can be complex and involve various roles and responsibilities, key management, auditing, exchanges between build servers and signing servers, and other operations as described in Section 6.

Firmware builds (new release packages as well as update packages) are required on 24x7 schedules and must provide throughput suitable for emergency situations as well as daily routine build scenarios. Additionally, the signing server must accommodate production, development, and testing keys. Efficiency is paramount, not only in the single operation of signing, but also in the additional process steps that must be implemented in production signing servers. The open source signing framework described in section 7 above, has been deployed in multiple production environments, and provides these capabilities efficiently.

8.2 Scalability

The open source signing framework provides for management of multiple distinct signing projects with appropriate administrative roles and audit controls. It is managed for operational security on a project basis and can be expanded to include many projects. At times, it may be desirable to replicate signing servers. For example, during development, firmware builds may involve developers from multiple countries (and companies) working 24x7 on their parts of the firmware. Having multiple local signing servers may be more efficient than funneling them all through a centralized one.

8.3 Openness

While the focus of this paper is to identify best practices for implementation and usage of a firmware signing server in the context of an OCP environment, it points to open source sign tools and a signing server framework that allow for signing environments that make use of these best practices. Vendors can freely modify these tools as needed. Examples include substituting other digital signature algorithms, auditing mechanisms, or key management libraries. Regulatory requirements may dictate that vendors use locally sourced cryptographic signing hardware and software libraries. These tools (and their licenses) support such modifications.

8.4 Impact

As a platform manufacturer, we have found a need for documenting best practices that we follow for our internal firmware build processes, as well as explicitly defining requirements in our agreements with suppliers. Defining a set of public documents to which the industry can refer, will help develop a consistent and more secure firmware supply chain.

9 Conclusion

Firmware signing and the underlying signing hardware, framework, and policies to administer the process address several of the OCP Security Workgroup's in-scope threats. This paper identified best practices for implementation and usage of a firmware signing server in the context of an OCP environment. It also included pointers to open source sign tools and a signing server framework that have provided the basis for development of these best practices across several production environments. Finally, it related the concepts in this paper to OCP's four tenets.

10 References

1. NIST Cybersecurity White Paper Security Considerations for Code Signing, https://csrc.nist.gov/CSRC/media/Publications/white-paper/2018/01/26/security-considerations-for-code-signing/final/documents/security-considerations-for-code-signing.pdf, retrieved on 11/27/2018.

2. OCP Common Security Threats, https://docs.google.com/document/d/13I-meE6BxiLB_c-Mjr3cLLK9S0SjuPuRjPfS9yTG6P8/edit#heading=h.ou65h8wmxru, retrieved on 11/27/2018.

3. E. Palmer, T. Visegrady, and M. Osborne, Ownership and Control of Firmware in Open Compute Project Devices, http://files.opencompute.org/oc/public.php?service=files&t=f4171bae8c7a32f05b0401378ee08483&download, retrieved on 1/7/2019.

4. Elaine Barker, Recommendation for Key Management, Part 1, General, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf, retrieved on 11/27/2018.

5. OCP Tenets Explained, https://www.opencompute.org/files/OCP-Tenets-FINAL2-1.pdf, retrieved on 11/11/2018.

11 License

12 About Open Compute Foundation