Infrastructure Security Best Practice

Web3 infrastructure security covers a wide range of technical fields. This document outlines the technology related to node security in Web3 infrastructure, which has a high intersection with system security, e.g: Linux security baselines such as CIS and STIG, runtime protection, mandatory access control, sandboxing, runtime attestation, networking firewall, and more.

For more information about W3IF (Web3 Infrastructure Foundation), visit https://web3infra.foundation/

We would like to recognize the following contributors to the document (listed in alphabetical order):

Revision number	Description	Contributor	Date
0001	Initial release	Ivan.xmr, HardenedVault KSTP.eth, HardenedVault Prof. James Lei, HKUST Shawn Chang, HardenedVault Weihua Du, HardenedLinux	July 9 2023

Introduction

This infrastructure security specification aims to provide guidelines and best practices for securing an IT infrastructure. This document will cover various aspects of infrastructure security, including operating system security baselines, runtime protection, access control mechanisms, sandboxing, attestation, and network firewalls.

Linux Security Baseline

CIS (Center for Internet Security)

The CIS provides a set of security configuration benchmarks for Linux systems. These benchmarks are designed to help organizations improve their security posture by hardening their systems according to industry best practices. Key CIS recommendations include:

- Regularly applying software patches and updates
- Configuring system logging and auditing
- Implementing strong user authentication and authorization controls
- Disabling unused services and protocols
- Ensuring proper file and directory permissions

STIG (Security Technical Implementation Guide)

The STIG is a standardized security configuration guide developed by the United States Department of Defense (DoD). The STIG provides security hardening guidelines for various platforms, including Linux. Key STIG recommendations include:

- Ensuring proper system and application patch management

- Configuring SELinux or AppArmor for mandatory access control
- Implementing strong password policies and account lockout mechanisms
- Securing network services and protocols
- Enforcing the principle of least privilege

Runtime Protection

Runtime protection involves monitoring and protecting applications and services during their execution. Key runtime protection measures include:

- Using intrusion detection and prevention systems (IDPS) to monitor and block malicious activities
- Implementing runtime application self-protection (RASP), WAF (Web Application Firewall), HIDS/EDR (end-point) to detect and prevent attacks at the application level
- Monitoring system and application logs for signs of suspicious activity
- Regularly scanning for vulnerabilities and applying patches as needed

Linux kernel runtime protection

To more effectively prevent Linux kernel 0-day or N-day vulnerabilities, defense against the exploit vectors are more widely applicable than defense against specific vulnerabilities. That is, even if there are vulnerabilities, they are difficult to exploit. The following are several aspects of what Linux kernel runtime protection can achieve:

ROP (Return-Oriented Programming) mitigation

PaX RAP of PaX/GRsecurity, KCFI (after v6.1) of the upstream Linux kernel, PA and Shadowstack on the ARM platform, and VED's wCFI. Among them, only VED uses the implementation method of kernel modules, which is weaker than PaX RAP and Shadowstack. The current version of KCFI on x86 only implements backward indirect call CFI. VED ensures that functions commonly used for privilege escalation can only be called by a small number of code segments in the kernel segment, and implements incomplete forward address checking. In addition, the runtime context check adopted by LKRG/VED can also prevent some ROP.

Post-exploitation/Rootkit prevention

Checking the integrity of the kernel code segment, loadable modules, and critical data during runtime can detect whether the kernel has been contaminated or illegally implanted with loadable modules, thereby preventing rootkit implantation in the kernel. Both LKRG and VED have implemented this type of function. VED's wcfi can also prevent kernel module calls to privilege escalation functions and privilege escalation operation via rootkit.

Data corruption protection

Strengthening commonly used data structures for vulnerability exploitation: VED performs integrity checks on data structures that are frequently used for vulnerability exploitation, such as addresses and lengths, which can prevent some information leaks, UAF, ROP, etc. PaX/GRsecurity implements AUTOSLAB to comprehensively check memory through the mechanism of the slab.

Security module's self-protection mechanism

Regularly performing integrity checks and runtime state checks on the security module itself can prevent vulnerability exploitation after the security module is unloaded.

Mandatory Access Control

Mandatory access control (MAC) is a security model that enforces access control policies based on the classification of information and the clearance of users. Two common MAC implementations are SELinux and AppArmor:

- **SELinux (Security-Enhanced Linux)**: A Linux kernel security module that enforces access control policies for processes, files, and network resources.
- **AppArmor**: A Linux kernel security module that provides application-level access control using profiles.

Both SELinux and AppArmor should be properly configured and used to enforce strict access control policies.

Sandboxing

Sandboxing is a security mechanism that isolates applications or processes in a restricted environment to limit their access to system resources and other applications. Key sandboxing techniques include:

- Using containerization technologies (e.g., Docker, Kubernetes) to isolate applications and their dependencies
- Implementing virtualization technologies (e.g., KVM, Xen) to create isolated virtual environments
- Utilizing seccomp (secure computing mode) to restrict system calls made by processes

Runtime Attestation

Overview

Runtime attestation is the process of verifying the integrity and authenticity of applications and services during their execution. Key runtime attestation techniques include:

Runtime attestation is the process of verifying the integrity and authenticity of applications and services during their execution. Its key techniques include Integrity Measurement Architecture (IMA) in the integrity subsystem in the Linux kernel, Trusted Platform Module (TPM) implementation for hardware-based attestation, and remote attestation and :

- Implementing Trusted Platform Module (TPM) for hardware-based attestation
- Using Remote Attestation to verify the integrity of applications and services running on remote systems
- Leveraging Intel Software Guard Extensions (SGX) or ARM TrustZone for hardware-assisted secure execution

In this specification, remote attestation is selected as the default runtime attestation solution considering the security of the verification environment and the flexibility of the attestation management. And in the attestation process, the integrity of running applications and services is verified by a remote system using the IMA measurement values and the TPM output.

Key Components

- Trusted Platform Module (TPM): A standard (iso/iec 11889) for the security cryptographic processor. The latest TPM 2.0 specification is developed by the Trusted Computing Group (TCG). TPM chip is a microprocesser hardware integrated in the motherboard that complies the TPM standard.

- Integrity Measurement Architecture (IMA): Linux kernel built-in subsystem for system integrity measurement. It usually works with the TPM hardware to prevent the measurement process from tampering.

- Mandatory access control (MAC): The measurement policy of IMA can be enhanced using the LSM specific policies implemented in the MAC components, including SELinux and SMACK (Simplified Mandatory Access Control Kernel).

- Verifier: The application located in a remote node where the hashes of attested files are compared and verified.

- Keylime: A remote attestation and runtime integrity measurement solution based on TPM and IMA. The attestation operation depends on the agent, verifier, registrar, and command-line tool.

Workflow

1. Measured Boot: Measured Boot is a prerequisite to enable measured boot as a critical step of overall runtime attestation. It secures the system components in the booting process from being modified by malicious behavior which happens before IMA takes over the system-wide measurement.
    a. With the coreboot firmware, measured boot starts from the IBB (Initial Boot Block) which works as an S-CRTM (Static Core Root of Trust for Measurement).
    b. In UEFI, secure boot is implemented and enabled by default by most vendors.

2. IMA Enabling: The function of IMA measurement is enabled from the kernel command line. The option is added to the GRUB configuration file to make it persist after the rebooting.

3. IMA Measurement Policy Configuration: SELinux object is applied in the rules of the IMA policy to specify what files should be measured and therefore achieves a fine-grained policy control. The SELinux should have been enabled in the system. In addition, the context of the files to be measured is distinguishable from those not to be measured.

4. Deployment Environment The deployment of remote attestation requires both the attested machine and the remote verifier machine to be in good and secure status, as instructed in this specification.

In addition, to minimize the attack surface, the system should be in the following status before the deployment:

1. Measured boot has already been enabled

2. Packages on the system have been installed from a reliable source through a secure network channel and their digital signatures are verified.

3. Extended Hash: On the machine to be attested, the extended hash of the files covered under the measurement is obtained from the TPM PCR-10 register. It has the same value as the aggregated hash calculated from the runtime measurement list under a good integrity environment. Before the hash value is sent to the remote verifier through the network, it should be signed in TPM using Attestation Key (AK), which is also generated in TPM. The extended hash does not leave TPM before signing to avoid being modified maliciously during the attestation process.

4. Remote Verifier On the remote verifier, the good hash value of attested files is retrieved from a trusted source or generated from the files obtained from a trusted source and should be stored securely.

5. Verification Method It is possible for the remote verifier to verify the integrity by simply examining the static PCR extended hash, known as the golden value, and checking if it matches the aggregate value pre-calculated from the good file hashes. However, this method suffers from the nondeterminism issue. In other words, a slight variation on any measured files due to system change can finally create a different PCR extended hash, although the other integrity-critical files are in good status. This issue is particularly common when the IMA policy is not fine-grained enough.

A more practical approach is comparing good hashes of only those integrity-critical files to the runtime measurement list obtained from the attested system. To ensure the list is not modified by malicious behavior, further verification is required to ensure its aggregated hash matches the PCR extended hash. Since the PCR extended hash is signed by AK, its integrity can be verified by checking the signature.

Runtime Attestation Framework

This specification recommends applying existing attestation framework instead of developing a solution of one’s own.

As a recommendation, Keylime is a remote attestation and boot measurement framework on the GNU/Linux platform based on TPM and IMA. The project is hosted by the Cloud Native Computing Foundation (CNCF) and in active development by the community. Also, it is available on all major GNU/Linux distributions.

Keylime provides the following features that are helpful for the deployment of a secure and practical runtime attestation environment.

1. Main components: Keylime framework consists of four core components in terms of agent, verifier, register, and tenant. The agent is deployed on the machines to be attested to collect necessary measurement information for attestation. On the remote node, the verifier performs actual verification of measurement value, and the register is responsible for the agent node registration. The tenant is a command-line tool for users to manage the Keylime efficiently.

2. Measured boot: Keylime support measured boot verification by examining the validation of PCR value extended during the booting process.

3. Secure Payload: The secure payload is a mechanism for the remote note to send confidential data used by the agent to the attested machine, such as configuration files, keys, and scripts, in a secure way. An especially useful use case of the secure payload is that a customized action on the attested machine can be triggered when the verification is failed on a remote verifier.

4. Allow List: To avoid the nondeterminism issue in the measured boot attestation, instead of using static PCR values, Keylime involves a policy engine to verify UEFI Event log. Furthermore, the allowlist mechanism is provided, by which only the specified files are measured in the attestation. It is achieved by comparing the runtime measurement list obtained from the agent machine with the good hashes in a local allowlist.

Overall, with a mature solution broadly applied by users and actively maintained by the community, the attack surface created by programming mistakes and misconfigurations could be reduced to a large extent.

HSM (Hardware Security Module)

Overview

To ensure the security and integrity of digital assets, we propose a solution involving Hardware Security Modules (HSMs). HSMs are dedicated cryptographic devices designed to protect sensitive information and perform cryptographic operations. This solution will provide a secure environment for key generation, seed generation, and handling of virtual assets.

Key Components

1. **Hardware Security Module (HSM)**: A dedicated, tamper-resistant device for secure key management and cryptographic operations.
2. **Key Management System (KMS)**: A centralized system for managing cryptographic keys, access controls, and policies.
3. **Application Layer**: Software applications and APIs for interacting with the HSM and KMS, as well as performing virtual asset transactions.

Workflow

1. **Key Generation**: The HSM is used for generating and storing private keys. This ensures that the keys are never exposed outside the secure environment of the HSM.
2. **Seed Generation**: The HSM generates a cryptographically secure seed, which can be used to derive additional keys or as a recovery method for the primary key.
3. **Access Control**: Access to the HSM and KMS is protected by strong authentication mechanisms, such as multi-factor authentication and role-based access control. This ensures that only authorized personnel can access the sensitive data and perform operations on the HSM.
4. **Transaction Signing**: When a virtual asset transaction is initiated, the application layer communicates with the HSM to sign the transaction using the appropriate private key. This ensures that the private key is never exposed to the application or network, providing a high level of security for the transaction.
5. **Auditing and Monitoring**: All operations on the HSM and KMS are logged and monitored for any suspicious activity or potential security breaches. This helps to detect and prevent unauthorized access or manipulation of the cryptographic keys and sensitive data.

HSM Product Recommendation

We do not recommend any products, but there are several aspects that require attention.

1. FIPS 140-2/3 Certification: This certification ensures that the HSM meets stringent security requirements and provides sufficient protection for sensitive data and cryptographic operations.

2. Performance: it depends on the use-case, e.g: DeX custody might need greater need for high-speed signature than CeX.
3. Scalability: HSM should be easily integrated into existing environments and scaled to support growing needs.
4. Ecosystem: Tools, libraries, and services for managing and interacting with the HSM, making it easier to implement and maintain the digital custody solution.
5. Transparency: The advantage of transparency brought by open source is obvious, and auditability is a prerequisite. Increasing the frequency of audits under this prerequisite will significantly reduce the risk of vulnerabilities and backdoors.

Last but not least, ask for a 3rd-party security audit report from the HSM vendor if the HSM is not open source.

Model	Type	FIPS-140 certification	Open source implementation	Public audit report	Cryptographic APIs	Restful API	Performance	Storage
Thales Luna PCIe HSM A700	PCI HSM	FIPS 140-2 Level 3	N/A	N/A	PKCS#11, Java (JCA/JCE), Microsoft CAPI and CNG, OpenSSL	N/A	RSA-2048: 1,000 tps	?
Thales Luna USB HSM	USB HSM	FIPS 140-3 Level 3	N/A	N/A	PKCS #11	N/A	RSA-4096: 8 tps	?
ProtectServer 3 External	Network HSM	FIPS 140-2 Level 3	N/A	N/A	PKCS#11, CAPI/CNG, JCA/JCE, JCProv, OpenSSL	YES	RSA-2014: 3500 tps	?
IBM 4769	PCI HSM	FIPS 140-2 Level 4	N/A	N/A	PKCS #11	N/A	?	?
YubiHSM 2 FIPS	USB HSM	FIPS 140-2 Level 3	N/A	N/A	PKCS#11, native libraries (C and Python)	No	RSA-2048-PKCS1-SHA256: 8	127 RSA2048 or 255 ECC
NitroKey NetHSM	Network HSM	N/A	??	??	PKCS#11	Yes	??	??

Networking Firewall

A network firewall is a security device that monitors and controls network traffic based on predefined security rules. Key network firewall best practices include:

- Implementing both ingress and egress filtering to control incoming and outgoing traffic
- Regularly reviewing and updating firewall rules to ensure proper access control
- Using network segmentation to isolate sensitive systems and applications
- Deploying intrusion detection and prevention systems (IDPS) to monitor and block malicious network activities

Conclusion

Implementing a comprehensive infrastructure security specification is crucial for maintaining a secure and resilient IT environment. By following the guidelines and best practices outlined in this document, organizations can significantly reduce their risk of security breaches and ensure the confidentiality, integrity, and availability of their critical systems and data.

Reference