anchore

VIPERR Software Supply Chain Security Framework

The VIPERR framework was created by Anchore engineers to serve as a blueprint for organizations to implement reliable and secure software development environments with the smallest possible lift. VIPERR stands for the core elements of software supply chain security: visibility, inspection, policy, enforcement, remediation and reporting. This 50 point checklist aims to be highly actionable by finding the right balance between giving broad guidance on software supply chain security that is flexible for all organizations and providing implementation specifications based on industry expertise that reduce options and help organizations make decisions faster.

VIPERR is broken down into five high-level sections (visibility, inspection, policy enforcement, remediation, and reporting) with 10 opinionated implementation specifications per section. These specifications take software supply chain security best practices from SSDF, SLSA, and NIST and applies them in ways that have been proven to be the right balance between easy to implement, highly secure, and fast to complete.

1.0 Visibility

Build an accurate software bill of materials (SBOM) from source to establish software provenance metadata at the component level (i.e. embedded software dependencies) of each software artifact. Visibility into software supply chain security is founded on the quality of the depth of what is measured. SBOMs standardize what is measured and how those measurements are persisted. Below is a list of critical components of an SBOM.

1.1 Identification of OS and language package

One of the most important features of container images is the ability to support minimal functionality per container. This will result in images that consist of the base operating system, plus an application, often written in one language. Identifying the base operating system, or base image, is important as that is generally not something that can be modified at will.

On top of that base image will be applications that often consist of a single language ecosystem. The content of that application can be modified at will to add features and upgrade components.

1.2 Identification of package licensing

All of the components in a container image are covered by some sort of copyright and license. Many container components will be open source libraries, each of which has a license that comes with license terms. By tracking the various package licenses, it's possible to ensure all license terms are being met.

1.3 Identification of package origin

There are a variety of ways an open source package can be obtained. It could come from the base image, it could be installed during a build, or it could be manually downloaded and included. Understanding the source of a given package helps understand why it is included in a container image and how to verify its integrity.

1.4 Identification of package size

The size of most packages is very small and not something that will ever be important, but having the ability to monitor sizes over time can help uncover problems. If a package unexpectedly doubles or triples in size it could be a sign of a larger problem.

1.5 Identification of all files

Any container image will have many files within it. Some of those files will be owned by packages, some will not. Being able to track all files and the relationships can help uncover unexpected data being added to an image.

1.6 Identification of file size

Just like package size, the size of most packages is usually very small. Tracking file sizes can be a useful tool for detecting problems in the future.

1.7 Identification of file permissions

All files have permissions and owners associated with them. Those file permissions should be checked for insecure configurations, such as global write permissions or set UID/GID. Likewise, the permissions of files should be tracked over time and unexpected changes should be investigated.

1.8 Identification of files unique identifiers

Identifying the unique hash of a file allows you to understand if the contents of the file have changed. Watch for unexpected changes during the development lifecycle of a container image, as well as unexpected changes in future deployments of a container image.

1.9 Identification of relevant image/package metadata

The metadata for a given package or file consists of all the above attributes. Being able to store, search, and retrieve that metadata is important in the present and will be important in the future.

1.10 Collect, safeguard, maintain, and share provenance data for all components of each software release

The ability to look back in time and ask questions about your container image data is a critical aspect of the software supply chain. Knowing the status of things right now is important, but the historical view should not be understated. Storing all container images forever may not be possible, but storing the metadata is a good idea given its relatively small size.

2.0 Inspection

Be able to query or ask questions about the data collected and stored in an SBOM. This typically comes in the form of security checks or scans inspecting the SBOMs for vulnerabilities, secrets, permissions, or malware.

2.1 Inspect files for malicious content

Malware can come in at various points of your software supply chain. It's important to utilize an SBOM tool that can catalog all files to identify any malware embedded in your software.

Related Compliance Controls:
NIST 800-171: SI-2
NIST 800-53: SI-2

2.2 Inspect packages for malicious content

Malware can be embedded as a file or directly in a software package. Typically, it's not labeled malware.exe either. This is why it is important to catalog all software packages in an SBOM and then analyze those packages against a malware scanner.

Related Compliance Controls:
NIST 800-171:SI-2
NIST 800-53: SI-2

2.3 Analyze each vulnerability to gather sufficient information about risk to plan its remediation

Vulnerability sprawl is another major threat to organizations but can be addressed by using the vulnerability analysis from an SBOM to generate a risk profile for your organization. This way, organizations can prioritize vulnerabilities attached to actual exploits and threats relevant to their software supply chain while disregarding irrelevant low-risk vulnerabilities.

Related Compliance Controls:
NIST 800-171:RA-5
NIST 800-53- RA-5

2.4 Inspect for license abuse/misuse

Inspect SBOMs to track your license compliance. It's important that organizations protect themselves from legal risks associated with misusing proprietary software and open source software.

2.5 Inspect source code repositories for vulnerabilities

Track the code your organization is consuming as early in the delivery lifecycle as possible. If you can identify vulnerabilities in your source code early on it will help in zero-day events or identifying where/when a vulnerability was introduced.

Related Compliance Controls:
NIST 800-171: SA 12
NIST 800-53: SA 12

2.6 Inspect & monitor file permissions

Identify the proper permissions in a file and validate that these permissions do not change. Unauthorized permission elevations to files could be a red flag of malicious activity.

Related Compliance Controls:
NIST 800-171: AC 6-(10)
NIST 800-53: AC 6-(10)

2.7 Inspect for misconfiguration in the Dockerfiles

Misconfigured Dockerfiles are a great way to spot malicious activity in your software supply. Alterations to Dockerfiles can be a sign of a developer mistake or malicious activity. Depending on the change this could lead to wide-scale attacks.

Related Compliance Controls:
NIST 800-171: CM 7
NIST 800-53: CM 7

2.8 Detect known exploited vulnerabilities

It's important to tag vulnerabilities that are known to be actively exploited by threat actors and threat groups. These vulnerabilities should be addressed first by validating that proper countermeasures and motivations are in place if any of these vulnerabilities are found in your environment.

Related Compliance Controls:
NIST 800-171: RA-5
NIST 800-53: RA-5

2.9 Inspect for vulnerabilities inherited by base images

Every organization should monitor the number of vulnerabilities in their base image, identify the types of vulnerabilities, and treat those as the baselines. This will make assigning responsibility for each vulnerability easier as development teams scale without mistakenly attributing a vulnerability in a base image to a developer in the organization.

Related Compliance Controls:
NIST 800-171: RA-5
NIST 800-53: RA-5

2.10 Inspect relational analysis to identify vulnerabilities across images and SDLC stages

Identify which vulnerabilities are attached to each piece of software. Drill down and see where that vulnerable piece of software is found across the entire software supply chain (source, build, registry, deploy, production).

3.0 Policy Enforcement

Enforce compliance with external or internal standards. After you've gained visibility and identified the threats in your software supply chain you must create policy rules that automatically enforce protections to mitigate the threats specific to your organization.

3.1 Policy alerts and blocking for malware findings

Automate your policy enforcement to automatically detect and block malware at any stage that it is detected in your software supply chain. You should use this information to be able to hunt for a specific piece of malware throughout your organization using the policy data.

Related Compliance Controls:
NIST 800-171: SI-2/SI-3
NIST 800-53: SI-2/SI-3

3.2 Create a policy that doesn't cripple velocity

It's important that the policy is tailored to your organizational needs. Prioritize the risks and create accurate policy enforcement for those risks to keep the noise down. Too many policy gates will cause friction and slow down software development processes. Too few and risks will rise dramatically in production environments.

3.3 Enforce control of license abuse or misuse with policy

Take automated enforcement actions to track license abuse and misuse across your organization. Place guardrails to alert on this activity immediately.

Related Compliance Controls:
NIST 800-53: AC-6(10)

3.4 Enforce secret and password monitoring in policy

An organization must be able to detect passwords and sensitive information from being leaked intentionally or unintentionally into their software supply chain. Organizations must be able to detect this using automated enforcement mechanisms to alert security teams when sensitive information is detected in unauthorized places.

Related Compliance Controls:
NIST 800-171: IA-5(7)
Nist 800-53: IA-5(7)

3.5 Policy checks for known exploited vulnerabilities across SDLC

An organization must automatically detect and block any known exploited vulnerabilities in their software supply chain. KEV lists are published by CISA. Automated tooling should be utilized to automatically check for items on the KEV at multiple stages in the SDLC rather than at one point in time. It is possible to utilize packages in the KEV database if mitigations are in place for that vulnerability and it does not introduce additional risk to your system.

3.6 Enforce building from approved images

Build from hardened base images that are useful to your organization that utilize minimal risk when adopting hardened base images. Developers should not be pulling from unauthorized registries that have not been vetted and/or secured by your own organization.

3.7 Detect and block misconfigurations in images

Blocking misconfigurations in images is imperative to prevent a software supply chain attack that could proliferate across the enterprise. Utilize automated enforcement mechanisms to alert when images enter the software development pipeline, ideally at the earliest possible stage. This can be done during the source or build stage by having code check-in require a scan for misconfiguration. If a misconfiguration is detected that could introduce significant risk.

Related Compliance Controls:
NIST 800-53: CM-7

3.8 Policy blocks unauthorized software from reaching the deployment orchestrator

Preventing software that violates an organization's security policy from reaching the deployment orchestrator is an important gate to implement. This is the last opportunity to stop vulnerable or known exploitable software from reaching public access. Ideally, vulnerabilities and exploits are caught well before this stage but if they aren't this is the last line of defense.

Related Compliance Controls:
NIST 800-53 : CM-7

3.9 Policy blocks unauthorized images reaching the registry

Blocking misconfigurations in images is imperative in order to prevent a software supply chain attack that could proliferate enterprise-wide. Utilize automated enforcement mechanisms to block images from entering both your registry and production environments if a misconfiguration is detected that could introduce significant risk.

Related Compliance Controls:
NIST 800-53 : CM-7

3.10 Don't allow builds that violate CVE thresholds

Establish a CVE threshold acceptable specific to your organization. It's important to identify a threshold specific to your organization and even specific to each development team depending on the project(s) they are working on at the time. For example, it may be useful to give more flexibility to a research and development team or a non-production environment than to a team working on the production workloads.

Related Compliance Controls:
NIST 800-53: CM-7, RA-5

4.0 Remediation

Recommendations and automation techniques to help the team resolve issues quickly. Give the developer the tools, resources and information they need to move fast and everyone benefits.

4.1 Provide remediation steps for software violating secure coding standards

Providing clear instructions on how to resolve a secure coding standard takes the burden of solving a violation off of the developer and reduces the time to remediation for security violations.

4.2 Automate the flow of CVE fix information to developers

Knowing that a CVE exists in a software component is helpful but researching the vulnerability and implementing a fix can be time consuming. Sourcing this information for a developer reduces the burden for a developer.

4.3 Provide ownership of who owns the fix

When a vulnerability is discovered, providing clear information on who owns the resolution of the vulnerability prevents confusion due to ambiguity and reduces the potential that a vulnerability exists in limbo. Clarity of ownership is paramount.

4.4 Integrate remediation into your notification & build service

If a remediation can be safely automated, integrate it directly into your build, alerting, and reporting system. This reduces the load on developers to manually implement simple automated remediation and increases the velocity with which vulnerabilities are remediated.

4.5 Prioritize remediations that are actively being exploited

When triaging vulnerabilities, those that are being actively exploited should be prioritized and addressed first. Automating the process of checking new software for known KEVs and scanning all software for newly published KEVs ensures that actively exploited vulnerabilities do not impact an organization.

4.6 Provide recommended remediations in developer workflows

Any remediation recommendations that are created when a vulnerability is discovered should integrate directly into the developer's current workflow rather than existing as a separate system that the developer has to integrate into their workflow. Push the information to the developer rather than having them pull it from the vulnerability system.

4.7 Enforce remediation standards with policy

Use automated policies in the build and deploy pipeline to prevent vulnerabilities that don't meet the risk standards of the organization to progress through the SDLC. Do not allow the honor system to be the prevailing policy for when to ignore or remediate vulnerabilities. Developer and security incentives are often not aligned, it is critical to have overarching policies that govern the process.

4.8 Centralize recommended remediations into a plan of action

As an incident progresses, storing the remediation recommendations in a central location allows for all stakeholders to be informed of what has happened and what is still outstanding. This significantly reduces potential confusion amid an ongoing incident.

4.9 Automate remediation responses to stakeholders

As remediation steps are completed it is important to communicate that progress and automate this communication. Security incidents can be high-stress events where normal communication patterns may be missed in the moment, sharing remediation information ensures all stakeholders are informed.

4.10 Create timeouts for remediations using policy

Policy enforcement of remediation recommendations can be progressive in order to create a middle ground between strict security enforcement and developer velocity. By creating a timeout, this allows a software component to be flagged as in violation of a policy but allowed to proceed with the condition that it is resolved by a certain time. If that time expires the software can either be automatically removed from service or new updates prevented until the violation is remediated.

5.0 Reporting

Reporting timely information quickly at any step of the development process is critical. This not only keeps all parties informed but enables information to be elevated and distributed as necessary.

5.1 Provide timely reports that meet relevant compliance standards

Compliance standards require different levels of reporting and delivery of the reports. A system should be able to meet the required reporting needs for any relevant standards. Standards like continuous Authorization to Operate (cATO) require near real-time reporting to stay in compliance.

5.2 Run reports at regular intervals and store for comparison

Reports should be run at regular intervals allowing for comparative analysis against prior reports. Accurate temporal data within these reports enables proper trend analysis to be performed over time.

5.3 Reports should have relational context

Reports should include relevant contextual information so that data can be interpreted correctly. This includes how the data should look under normal circumstances and what it means when it deviates from the norm.

5.4 Reporting should be 100% automated

Do not rely on manually generated reports. Utilize a system that can automatically generate reports and deliver the reports to where they are consumed. From here you can scale out your software development without creating bottlenecks that slow velocity or create friction.

5.5 Reports should target accurate artifacts

Reporting on corrupted software artifacts or misconfigured systems will create bad data that will lead to poor decisions. Ensuring the accuracy of the software components and associated artifacts is important for proper decision making.

5.6 Reports should reference impacted artifacts

Reports should always be scoped to the most relevant software artifacts. Reporting is only effective when the correct information is provided. Additional information dilutes the effectiveness of a report and missing information runs the risk of bad conclusions being drawn from the data.

5.7 Reports should be provided to every team and developer

Reporting is only effective if it is consumed by the correct audiences that can utilize the data to improve the system. Reporting on software supply chain security needs to be communicated to all software development teams and the engineers on those teams. These individuals and teams can direct action on the feedback.

5.8 Reports should include accurate timestamps

Reports on the state of the software supply chain security of a system lose value over time. Providing accurate timestamps on the reports and automatically delivering the data in a timely manner ensures that the most value can be extracted from the report.

5.9 Include reports in the risk assessment and risk management process

Software engineering teams may have different priorities than business operations teams. Including risk management and assessment departments allows business priorities to be integrated as part of the prioritization process. This allows engineering teams to prioritize resources based on technical constraints as well as business operations constraints. This creates overall tighter alignment within an organization.

5.10 Regularly update reporting mechanisms for new checks

An automated software supply chain reporting system is software just like the software that it reports on. It needs continuous maintenance to ensure that it is accomplishing the previous nine points in this section and achieves its purpose of distributing the right information to the right systems at the right time with the right context.

Contact Us

Anchore, Inc. • 800 Presidio Ave. Ste. B • Santa Barbara, CA, 93101-2210 • United States

(805) 456-8981 • sales@anchore.com

anchore