Zero-Day Vulnerabilities in Container Images: Risks and Detection using Generative AI

Karthikeyan Thirumalaisamy

Independent Researcher, Washington, USA
Corresponding Author Email: *kathiru11[at]gmail.com*

Abstract: ***Containerization has changed the way we deploy software by giving us the ability to deploy lightweight, portable, and scalable applications. However, it has also introduced a larger attack surface were risk surfaces from usage of unpatched or unknown vulnerabilities in containers. Zero-day vulnerabilities are those flaws that are exploited before being made public or patched, they can pose a serious threat to anything running in a containerized environment because they can easily propagate without any detection. Conventional security scanners are heavily reliant on the known vulnerability databases and do not detect zero-day risks. This paper examines the zero-day vulnerabilities, describes the boundaries of known detection mechanisms, and proposes a new way to identify and reduce risks utilizing generative AI. Using large language models (LLMs) and generative systems, it illustrates how generative AI can enhance both static and dynamic analysis, automate threat pattern recognition, identify relationships between threats, and produce real-time contingent remediation. We argue that generative AI as a solution in DevOps provides a better level of proactivity and reactivity to address the ever-evolving threat landscape in containerized applications.***

Keywords: Zero-day, Zero-day vulnerabilities, Container Image vulnerabilities, AI mitigation, AI detection

1. Introduction

Container technology is rapidly replacing traditional methods and becoming the primary way of developing and deploying software. Tools like Docker and Kubernetes have given developers the ability to "containerize" applications and dependencies in portable, lightweight units called containers. Containers are technologies that allow the packaging and isolation of applications with their entire runtime environment (all the files necessary to run). This makes it easy to move the contained application between environments (dev, test, production, etc.) while retaining full functionality. As enterprises increasingly adopt cloud-native and microservices based architecture, the usage of containers in both the enterprise and open source communities have grown exponentially.

Despite the advantages that containers provide, it also creates its own significant security challenges. Container images contain full operating systems, third-party libraries, and custom code from the application. Each of these are an entry point for an attack, and the most pernicious attack vector are zero-day vulnerabilities. A zero-day vulnerability is a software bug or defect not previously known (to the developers/security team) and is exploited by an attacker prior to the organization becoming aware of its existence. Once a zero-day vulnerability exists within a container image, it can spread across multiple environments and possibly result in thousands of deployments from a single compromised base image.

Normal security tools, including static-image scanning or signature-based threat detection, are not really capable of detecting the exploitation of zero-day threats. Current tools are all fundamentally reactive - they look for vulnerabilities from what's in the database (e.g., CVE lists) and, as such, will work only against what vulnerabilities are known and published. Zero-day vulnerability exploit will usually be either a new exploit or an obfuscated exploit. Secondly, containers are ephemeral and distributed; once a container is instantiated, exploited, and destroyed there is literally no timeframe for static detection programming to ascertain what is an adequate response to a containerized malicious exploit ahead of a threat actor.

To fill the gaps in traditional tools for security, Generative AI, most notably large language models (LLMs) is ascending as a viable option. Generative AI has the ability to analyze terabytes of code at high speeds; recognize patterns that indicate unusual behavior; and even posit the existence of vulnerabilities that had not yet been reported. Trained on an array of datasets, including open-source repositories and logs, Generative AI gets an overall feel for how software systems behave.

The primary advantage is the speed of detection. The sooner you can detect a potential threat, the sooner you can begin looking for anomalies in static code, segregating unsafe binaries, and replacing insecure components with secure components. Generative AI can also help to automate (or semi-automate) security policies, generate remediation, and repair faulty configurations - allowing you to eliminate or decrease risk before it gets exploited.

This paper talks about the threat of zero-day vulnerabilities that may be present in container images and presents a new method of using Generative AI to identify and reduce them. We will describe existing security tool limitations, explain how AI can enhance and extend the current layers of protection, and conclude with a few actionable steps organizations can take to minimize the risk to zero-day vulnerabilities in a containerized environment.

2. How Zero-Day Vulnerabilities Are Exploited in Container Images

Container images are convenient, but they also can contain undiscovered zero-day exploits. Container images include applications, but they also include all dependencies, libraries, and possibly part of the operating system, all packed up together. Here are examples of how attackers can exploit zero-day vulnerabilities within the container images:

a) **Pre-Packaged Vulnerable Components in Base Images:** When creating a container image, many developers use the publicly available base images (i.e., Ubuntu, Alpine, Node.js, etc.) to include their application into a container. If the base image contains an existing, yet undiscovered zero-day exploit vulnerability (think, Linux kernel, system library, network utility, etc.), when the container starts, it can be exploited.
b) **Compromised Dependencies:** Often containerized applications depend on additional open-source libraries and tools. If one or more dependencies that are zero-day is included in the container, it can become a very well hidden backdoor that can be exploited.
c) **Malicious or Compromised Container Images:** An attacker can publish or inject a malicious container images within a public or private registry. Malicious container images can contain zero-day exploits to be executed when the container runs the image, bypassing the normal security checks, as the zero-day vulnerability is unknown.
d) **Supply Chain Attack:** The supply chain can also introduce zero-day exploits if the CI/CD pipeline or container registry is compromised. Attackers can easily introduce zero-day exploits into the image unknowingly to the developer.
e) **Attackers exploiting the container at runtime:** When a container with a zero-day exploit is running, attackers can exploit that container to:
   - Escape the container to the host system
   - Escalate privilege within the container
   - Execute arbitrary code or install malware
   - Move laterally across a cloud environment or cluster
f) **Delayed Detection:** As zero-minus days are unknown, container vulnerability scanners and traditional security tools will generally not flag zero-day vulnerabilities. Because of the unknown, a zero-day exploit can go undetected for some amount of time in containerized environments.

3. The Importance of Container Vulnerability Scanning

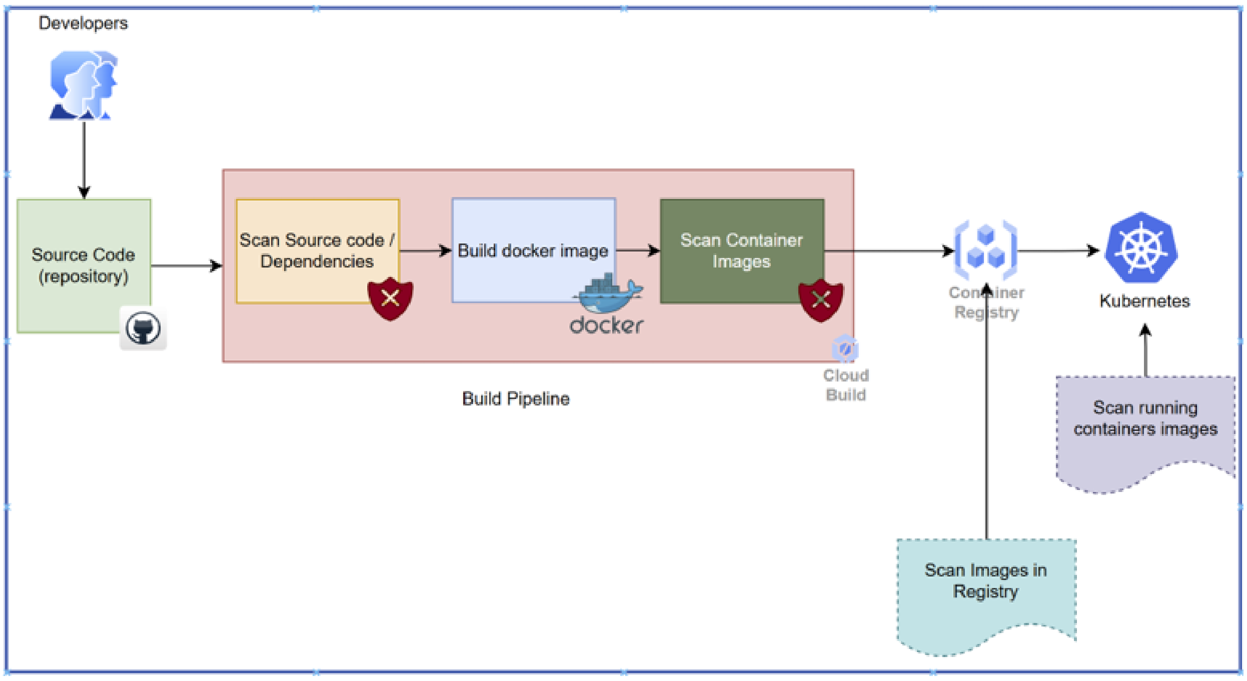
Container vulnerability scanning is critical for several reasons as it is a foundational element in identifying and managing the risk associated with containerized applications. Containers have become ubiquitous in today's software development and deployment practices, necessitating a focus on security.

Here are some reasons container vulnerability scanning is critical:

a) **Identify Vulnerabilities Earlier:** Vulnerability scanning for containers allows organizations to identify security vulnerabilities in container images before they ever enter a production environment. Identifying the vulnerabilities early allows organizations to address them prior to release, reducing their risk around security breaches.
b) **Ensure Compliance:** Many organizations are bound to various regulatory compliance requirements with very established regulatory requirements around controls and practices. Vulnerability scanning for containers provides organizations with the ability to demonstrate compliance by scanning their container images for known vulnerabilities.
c) **Reduce Attack Surface:** Scanning for vulnerabilities in container images aids organizations to strive for minimizing attack surface while ensuring that potential codes/packages/dependencies that could create risk are removed.
d) **Continuous Security within DevOps:** Software is developed, tested and released rapidly in today's DevOps or CI/CD (Continuous Integration/Continuous Development) practices. Vulnerability scanning provides continuous security at each stage of the pipeline by scanning container images while ensuring that security is always current with rapid release delivery models.
e) **Third-party dependencies:** Container images often rely on various third-party libraries, packages, base images, etc. that have known vulnerabilities. Scanning container images lays the groundwork to identify and access risks around this third-party sourced software.

4. Current Method of Containers Scanning

Container vulnerability scanning is typically an established process that scans containers from the build stage to the runtime stage. Integrated scanning through DevOps pipelines, container registries, and orchestration layers ensures that the transient workloads are as secure as the more permanent workloads. The diagram below depicts the current method of scanning containers as well as the breakdown of the primary scanning stages and how they form a coherent security cycle.



4.1 Scan Source code and dependencies

The initial step of the build pipeline is scanning the application's source code and dependencies for vulnerabilities. This approach makes sure that no new vulnerabilities are introduced**.** It scans the source code to identify hardcoded secrets (API token, password), unsafe functions or patterns (command injection, SQL injection), insecure configurations (missing input validation, open ports) and potential logic flows that may be exploited in production using known vulnerabilities databases. Static Application Security Testing (SAST) tools are often used for code scanning. The goal of static scanning is to review the source code without executing it and helps to identify vulnerabilities and fix it early in the development lifecycle which is easiest and cheapest to fix.

4.2 Dependency Scanning

Applications are typically reliant on open-source libraries, frameworks, and modules, which means that dependencies will likely comprise a significant portion of the application code. Dependencies are most frequently packaged or included in:

- package.json (JavaScript/Node.js)
- requirements.txt or Pipfile (Python)
- pom.xml (Java)
- Gemfile (Ruby), etc.

Dependency scanning will scan the above mentioned files and generate a Software Bill of Materials (SBOM) listing all components used. It also cross-reference the included packages and their versions against available vulnerability databases (NVD, GitHub Security Advisories, etc.). Finally, it will locate the deprecated and outdated packages and recommend the updated or safer package versions. The next step (Build docker image) will be executed only when these steps are completed without any detections.

4.3 Scanning Container Images

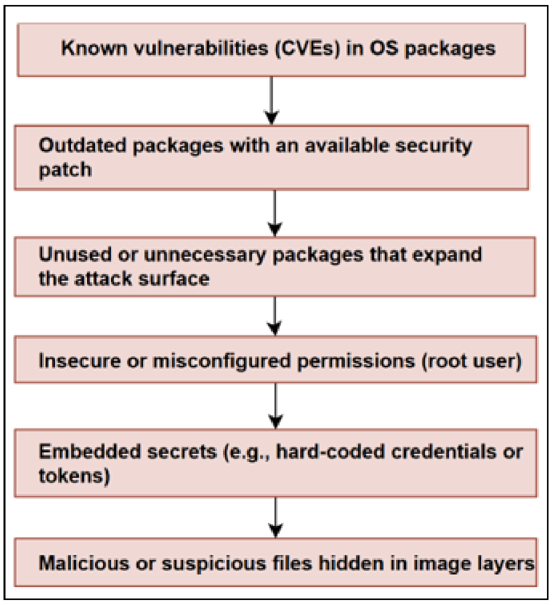
Once the application code and dependencies are packaged into a container image, it is necessary to scan that image to ensure it will not inject vulnerable code into our production environments. The images are often built using the base images (e.g., Ubuntu, Alpine, or Debian) that may themselves contain vulnerable packages or dependencies.

Container image scanning tools, such as Anchore, Clair, and Trivy, are commonly used with the purpose of improving container security by scanning each individual layer of a container image. These container scanning tools efficiently and methodically scan everything that has been bundled together including the base image, application code, and any added dependencies for security vulnerabilities.

These tools not just identify these issues, but also generally provide risk severity ratings, remediation recommendations, and integration capabilities with CI/CD pipelines, which includes the ability for developers and DevOps teams to automatically scan images as part of a development workflow. Therefore, vulnerabilities will be caught early and remediate before the application is deployed to production.

Container images are created as layers, and each layer is an individual change or addition applied to the base image throughout the image build process. This layered format is critical to the functioning of containers, but also provides a good degree of efficiency, modularity, and reusability. Usually, each image build starts with a base image (e.g., from a Linux distribution flavor such as Ubuntu, Alpine, or Debian), which provides the core operating system files and utilities the application is expected to run against. Once a base image is established, the developer incrementally adds application-specific code, third-party libraries, runtime dependencies, environment variables, configuration files, and potentially some initialization or application run-time scripts. Each instruction in a Dockerfile (e.g., RUN, COPY, ADD) implements a new layer in the final container image.

During the scanning phase, scanning tools analyze the image layers against a multitude of vulnerabilities and issues, such as:


Known vulnerabilities (CVEs) in OS packages
Outdated packages with an available security patch
Unused or unnecessary packages that expand the attack surface
Insecure or misconfigured permissions (root user)
Embedded secrets (e.g., hard-coded credentials or tokens)
Malicious or suspicious files hidden in image layers

After container scanning is completed successfully, the image will be published to registries such as Docker, Azure Container Registry, Amazon Elastic Container Registry, Google container registry or private registries. If the container scanning fails then developer must fix the vulnerabilities in the application code, dependencies or even base images and restart the whole process again.

4.4 Scanning Images in Container Registry

Image scanning in a container registry is the process of automatically scanning the container images stored in a single registry (ex. Docker Hub, Amazon Elastic Container Registry (ECR), Azure Container Registry (ACR), or private registries) for potential security vulnerabilities prior to them being pulled into production.

Container registries are centralized locations for the storage and distribution of container images. When a developer builds an image, they will almost always push it to a registry so the image can be commonly shared, reused, and deployed into various environments. However, these images in registries might still be outdated, misconfigured and contain vulnerabilities. This, obviously, can cause problems. Periodic, consistent and ongoing scanning will ensure any image, whether that image is new or old and whenever it was built, can be scanned for vulnerabilities.

Most private and cloud container registries offer the following capabilities for scanning stored images:

- Automated scanning triggers
- Most contemporary container registries offer automated scanning, with the following triggers:
- With an image being added or pushed to the registry (push-time scanning)
- Following a defined schedule (i.e. daily, weekly, etc.)
- When a new CVE is published (event-driven scanning)

4.5 Scanning images in Kubernetes

Container images are the basic building blocks for deploying applications in Kubernetes. The images are pulled from the registry and deployed as pods. Thus, we need to make sure that the images are safe to run and do not contain known vulnerabilities. Scanning images in Kubernetes allows you to only run secure, compliant, trusted container images in the clusters. Images can be scanned at pre-deployment or during deployment in the cluster to identify any vulnerabilities in the images.

An image scan can be integrated at different stages in the Kubernetes lifecycle:
- Admission Controller Scanning
- In-cluster Image Scanning

4.5.1 Admission Controller Scanning
Admission Controller Scanning is a powerful security mechanism utilized in Kubernetes to enforce policies at the time a workload is submitted to the cluster. Admission controllers are different than pre-deployment scanning because they perform their inspections at deploy time—during the request's lifecycle—before the request persisted to etcd or allowed to run in the cluster.

This real-time inspection into a workload ensures that only compliant and secure workloads are placed into a Kubernetes environment. An Admission Controller is a Kubernetes component that intercepts (validating or mutating) requests to the Kubernetes API after authentication/authorization has succeeded but before the object persisted. Admission controllers can enforce image scanning, can reject workloads, can enforce workloads to have certain conditions to be placed (e.g., not have high-severity CVEs, not be a root user, etc.).

4.5.2 In-cluster Image Scanning
In-cluster image scanning helps organizations perform security assessments on container images in running Kubernetes clusters. Rather than pre-deployment scans that take place as part of the build or CI/CD process, in-cluster scanning provides organizations visibility into what has been deployed to their environment in real-time. This helps organizations uncover vulnerabilities that may have not been deployed with the image at the time in builds or workflows or that have been identified with the release of new CVEs after they were deployed.

In-cluster image scanning deploys a scanning agent (e.g. Trivy Operator, Starboard) into the Kubernetes cluster. The scanning agent will discover workloads by querying through the Kubernetes API for running Pods, Deployments, StatefulSets, etc. For each container, the scanning agent will extract the image digest (SHA) and related metadata.

Optionally pull and scan the image locally, or use existing scans located in the external database or other store. The scanning agent will compare each image layer against known CVE databases (NVD, GitHub Advisories, vendor advisories, etc.). The resulting data will be stored in a centralized storage, dashboards, or within an external security system. This data will also be used to trigger alerts and remediation automatically.

4.6 Summary

Container vulnerability scanners that work on the traditional method are good at identifying known threats and improving baseline security; however, they may provide limited or no protection against zero-day vulnerabilities, which are unknown at the time of scanning. In order to reduce the risk of zero-day vulnerabilities, organizations should combine traditional scanning with real-time continuous monitoring, behavioral anomaly detection, and AI-driven scanning.

5. Container Scanning using Generative AI

Traditional container vulnerability scanning tools like Clair, Anchore, and Trivy have been important in exposing known vulnerabilities. Their function involves obtaining the contents of container images (like OS packages, libraries, binaries) and checking them against public vulnerability databases like the common vulnerability and exposures (CVE) list. These tools are effective, they operate quickly, are automated, and are well integrated into CI/CD pipelines, making them a crucial part of secure DevOps process.

However, traditional scanners are naturally reactive and can only find things that have already been reported and recorded. There will always be a gap with zero-day vulnerabilities, flaws that may have been recently introduced or are undiscovered to the security community. Any unknown vulnerabilities can persist undetected within container images, only to be found through active exploitation in the wild.

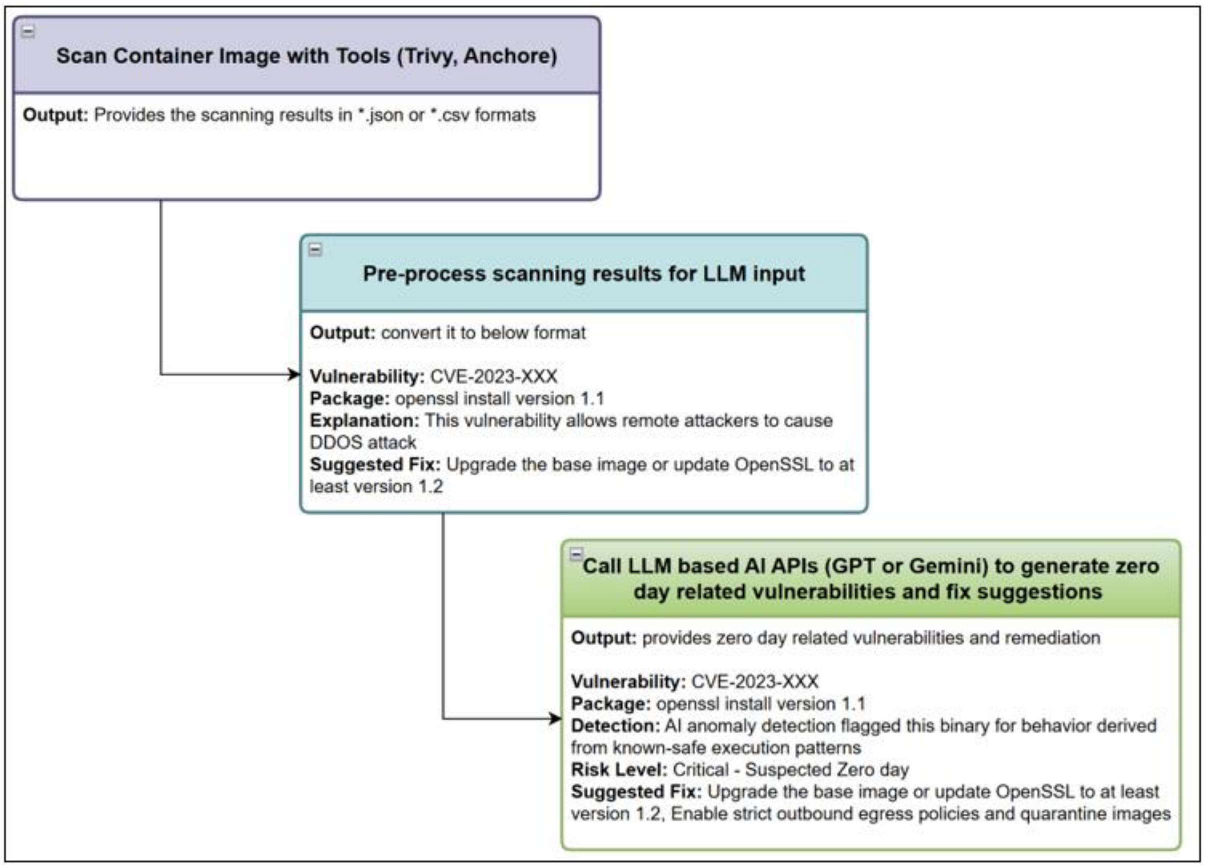
This is where Generative AI, particularly large language models (LLMs) and code-based AI systems, provide significant value. Generative AI leverage semantic, behavioral, and contextual analysis of container image contents and AI models have been trained on vast data that includes source code, configuration files, logs, threats and vulnerability reports. It can also identify the code snippets that may lead to vulnerable software, unsafe configurations or anomalous patterns regardless of established CVEs or signatures. Generative AI can reason through complex code logic, identify insecure callable and usage practices, suggest remediations, as well as rewrite portions of code or configuration in a more secure manner. This allows security teams to address not only known issues but also the potential risks associated with previously unidentified vulnerabilities, a significant improvement in the integration of scanning and exploitation, even when the specific vulnerability has not been discovered.

5.1 Integrate Traditional Container Scanning Tools with LLMs Based AI

The best way to improve container security and fill the gap in the detection of zero-day vulnerabilities is to use the hybrid approach of combining traditional container scanning tools with LLM’s based AI systems such as GPT, Gemini, Claude. This hybrid approach uses the strengths of signature-based scanners and combines them with the reasoning, context-awareness and predictive capabilities of generative AI. We will use LLM based GPT API, but this can be replaced with any LLM based AI APIs.

This approach can be implemented using any traditional scanning tools and here we are using Trivy as a selected tool. Trivy is an open source, comprehensive, cross platform and versatile security scanner. Trivy has scanners that look for security issues such as OS packages, known vulnerabilities (CVEs), Sensitive information and secrets, and targets including Container Image, File system, Git repository, Kubernetes. Trivy will provide an output with details of vulnerability and fixes.

The below section explains the details of the diagram, provides the high-level overview and explanation of integration with Trivy (traditional container scanning tool) and LLM based GPT AI.


Scan Container Image with Tools (Trivy, Anchore)
Output: Provides the scanning results in *.json or *.csv formats
Pre-process scanning results for LLM input
Output: convert it to below format
Vulnerability: CVE-2023-XXX
Package: openssl install version 1.1
Explanation: This vulnerability allows remote attackers to cause DDOS attack
Suggested Fix: Upgrade the base image or update OpenSSL to at least version 1.2
Call LLM based AI APIs (GPT or Gemini) to generate zero day related vulnerabilities and fix suggestions
Output: provides zero day related vulnerabilities and remediation
Vulnerability: CVE-2023-XXX
Package: openssl install version 1.1
Detection: AI anomaly detection flagged this binary for behavior derived from known-safe execution patterns
Risk Level: Critical - Suspected Zero day
Suggested Fix: Upgrade the base image or update OpenSSL to at least version 1.2, Enable strict outbound egress policies and quarantine images

5.1.1 Scan Images with Trivy

The first step is to scan the container images against the known vulnerabilities databases (e.g., NVD, Red Hat, Debian Security Tracker) and provide an output with Vulnerability Id (e.g., CVE-2025-xxxx), affected package version, severity (Low/Medium/High/Critical) and suggested fix if its available as a json format.

5.1.2 Pre-process Scanning results for LLM input

Container vulnerability scanners (like Trivy, Clair, or Anchore) output the results in structured and meaningful format (.json, .csv, etc.) that a Large Language Model (LLM) cannot understand or contextualize better. So, pre-processing step is required to convert scanner output to AI readable format which consists of the following steps,

a) **Parse the Output:** Take the raw scan output (csv or json) and clean up and organize it into a simple format like tables or lists.
b) **Filter by Severity:** Only focus on the most severe issues (e.g. high or critical vulnerabilities), or those that impact standard components.
c) **Add Additional Context:** Add further details about what the package is being used for, what service it is part of and where is it located in the container.
d) **Summarize the Vulnerabilities:** Reformat each CVE in a way that is easier to consume. As part of the summarization, be sure to include, the CVE ID, a general description of what it impacts, how severe it is, if there is a fix, and how easily it can be exploited.
e) **Flatten Deeply Nested Data:** If the output is complex (i.e. multi-layered, standard dependencies), re-organize it into flat or more easily consumable format.
f) **Add Environment Context:** As part of the vulnerability context for recommended actions, add details about the operating system, application stack and where it will be deployed (i.e. Kubernetes cluster).

5.1.3 Sending Pre-processed Input to GPT API to identify zero days vulnerabilities

After preprocessing completes, send the preprocessed input to GPT API to identify potential zero days vulnerabilities. GPT reads the information and tries to spot signs of deeper risk, GPT can connect the links between different risks. It can predict weaknesses that attackers might exploit even if they aren't officially reported. It brings contextual awareness (e.g., "this misconfiguration + this service = potential exploit"). It will look at the suspicious behaviors that lead to zero-day exploits. Connect the dots with configurations mistakes, unpatched software, insecure behaviors that attackers may exploit before existing public CVEs become available. Model an attacker's thinking based on a known attack path (i.e., TTPs from the MITRE ATT&CK framework).

Once an in-depth contextual analysis of the preprocessed input is completed, GPT will then provide a full vulnerability report including potential zero-day vulnerabilities. These vulnerabilities are not only based on existing CVEs, or common vulnerabilities and exposures, but also inferred vulnerabilities based on patterns of insecure misconfiguration, lack of appropriate patch management of core components or application dependencies, or exposed secrets, as well as behavior combinations that attackers may exploit even if the CVEs have not been reported or published.

Every report typically includes:

a) **Vetted Vulnerabilities:** GPT will indicate both known vulnerabilities and inferred vulnerabilities based on the container image structure, behavior, detailed source code, and configuration.

b) **Risk Assessment Levels:** GPT will assign a risk level to each actual or inferred vulnerability based upon context such as whether the container is running root, what sensitive services it may be communicating with, or if it is exposed on the public internet etc.

c) **Inferred Zero Day Indicators:** GPT will infer/recognize possible zero-day by utilizing reasoning and pattern matching either using insecure defaults, or un-patching behaviors, and recognizing combinations or dependencies with possible unusual patterns, inconsistency, or poor permission.

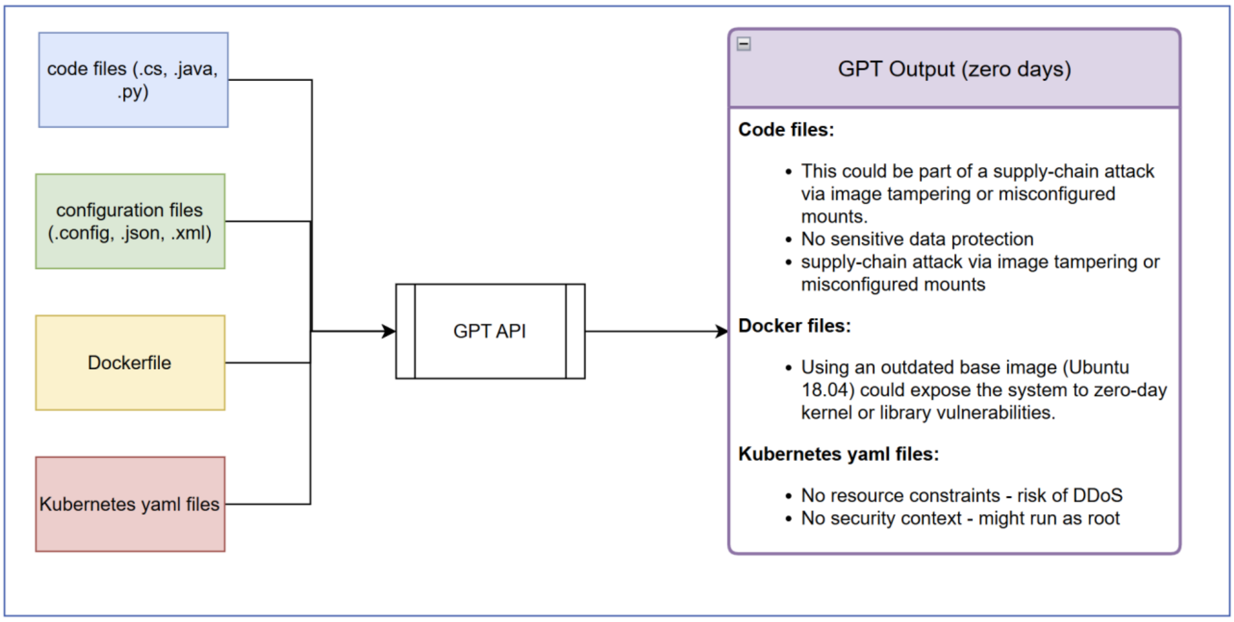
d) **Remediation Guidance:** For each identified or suspected vulnerability, GPT offers detailed, human-level recommendations for remediation. These include instructions on how to proceed, what updates are necessary, and guidance on reconfiguring settings appropriately. These may include:
   - Updating to a safer base image
   - Patching or replacing outdated packages
   - Removing unused exposed ports or services
   - No hard coded secrets
   - Using non-root users in docker files.

Providing not only detection, but contextually aware explanations and actionable fixes, GPT becomes a powerful resource for security and DevOps teams to help secure containers even against emerging, or unknown (zero-day) threats.

5.2 Scan source code and configuration files using LLMs Based AI

Generative AI, and especially large language models (LLMs) trained on enormous codebases, inherently offers a model shift in the way we analyze code for containerized applications. Unlike traditional scanners which will only rely on pattern matching and known signatures, generative AI can understand a code's logic, intent, and structure. This enables generative AI to detect a wider range of vulnerabilities including zero-day vulnerabilities, risky coding patterns, insecure configuration, etc., that static tools may miss.

Below diagram explains the flow of scanning the source code and configuration using GPT API:


code files (.cs, .java, .py)
configuration files (.config, .json, .xml)
Dockerfile
Kubernetes yaml files
GPT API
GPT Output (zero days)
Code files:
This could be part of a supply-chain attack via image tampering or misconfigured mounts.
No sensitive data protection
supply-chain attack via image tampering or misconfigured mounts
Docker files:
Using an outdated base image (Ubuntu 18.04) could expose the system to zero-day kernel or library vulnerabilities.
Kubernetes yaml files:
No resource constraints - risk of DDoS
No security context - might run as root

As shown in the above diagram, the process starts with feeding the source code, the Dockerfiles, and the configuration files (e.g., the Kubernetes manifests) to GPT API. The model then performs a semantic and contextual analysis of the inputs; the model is not just searching for known vulnerabilities and risks, it identifies potential zero-day vulnerabilities for input examples through its ability to recognize unusual patterns, unsafe logic, or insecure configurations that may be missed by traditional analysis tools. The output is a comprehensive security report that describes the potential risks and vulnerabilities (including zero-day vulnerabilities, i.e., not documented yet) and offers potential alternatives securely based on secure coding best practices and the AI model's rationale.

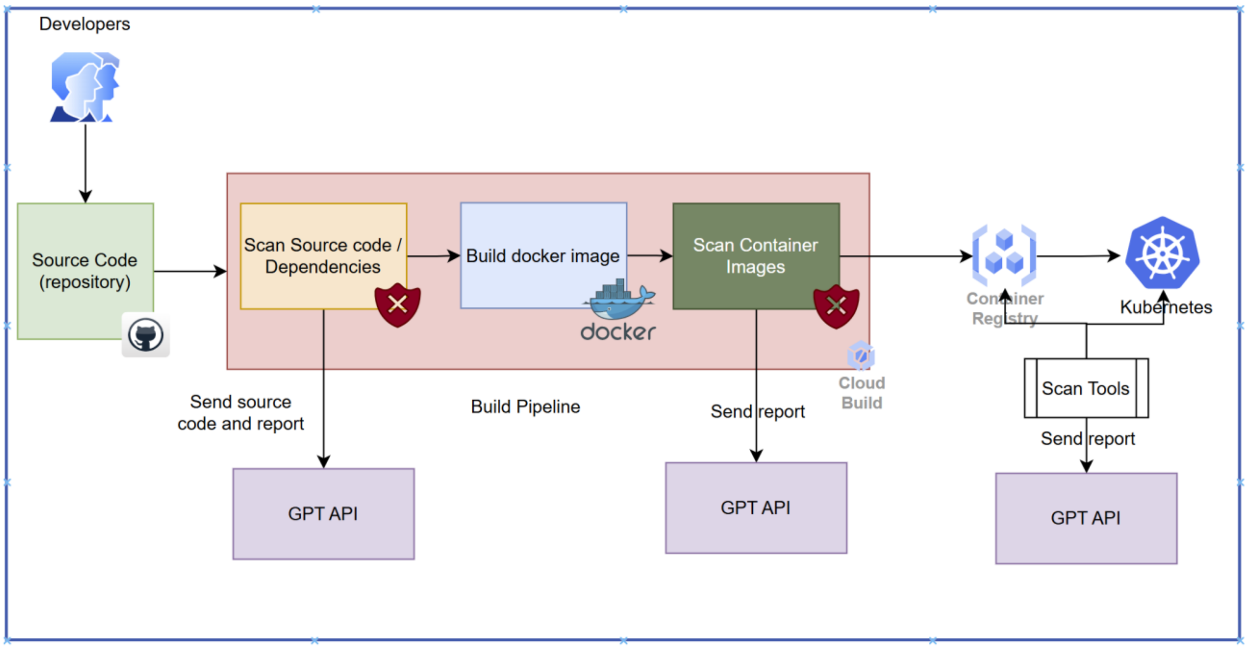
6. Proposed Approach: Generative AI powered Container Scanning

In this section we will discuss a complete approach to detect zero-day vulnerabilities by integrating multiple scanning points into the AI powered container image scanning pipeline. This will include scanning of the source code, scanning of the reports from CI/CD pipeline, container registry scan reports, and Kubernetes cluster scan report data. All reports and artifacts will be preprocessed and submitted to the Generative AI model (e.g., GPT) for analysis.

When combining static analysis (e.g., code scans), container image scan results, and the runtime environment information, the Generative AI could show the hidden patterns, misconfigurations, or risky behavior that might reflect the existence of zero-day vulnerabilities, even if they are not known or able to be identified through a public database.

In addition, we could maintain continuous monitoring and proactively detect these risks before exploitation occurs. The AI model runs complex reasoning across the different layers and there is the potential to identify risks, prioritize them based on severity and remediate it before any attacker could act on them.

Refer the diagram below for fully integrated AI powered Scanning pipeline:



7. Conclusion

This paper provides an overview of the emerging challenge of securing containerized applications against growing threat landscapes, especially in relation to zero-day vulnerabilities. Conventional container scanning tooling is also effective at preventing known vulnerabilities with static databases and signature matching but is incapable of contextualizing or anticipating novel or evolving threats.

This paper outlined a solution for zero-day vulnerabilities in the containers with Generative AI, specifically large language models (LLMs) like GPT, introduced a new paradigm around container security. By analyzing programming source code, container images, and runtime config through contextualization and prediction, AI predicts possible vulnerabilities, misconfigurations, and anomalous behavior that otherwise may go unnoticed. With the implementation of CI/CD pipelines, Container Registry scanning, and Kubernetes image scanning enables near real-time prevention of zero-day vulnerabilities and misconfigured containers. This is achieved through continuous and proactive security measures, allowing for the association of new findings to an evolving landscape of potential threats.

This paper also explored an option to combining traditional scanning tools with AI-powered analysis. Thus, organizations have a new and powerful defense mechanism to not only detect known risks but predict future risks. The proposed method advanced the performance of threat detection, reduces zero-days exposure and improves security posture by better securing containerized applications in cloud-native environments.

References

[1] IBM security, What is a zero-day exploit?, 2024. [Online]. Available: https://www.ibm.com/think/topics/zero-day
[2] Wikipedia, Zero-day vulnerability, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Zero-day_vulnerability
[3] TIGERA, What Is Container Vulnerability Scanning?. [Online]. Available: https://www.tigera.io/learn/guides/container-security-best-practices/container-vulnerability-scanning/
[4] Jeffrey Schwartz, 87% of Container Images in Production Have Critical or High-Severity Vulnerabilities, 2023. [Online]. Available: https://www.darkreading.com/vulnerabilities-threats/87-of-container-images-in-production-have-critical-or-high-severity-vulnerabilities
[5] James Berthoty, How Container Vulns Get Fixed, 2024. [Online]. Available: https://pulse.latio.tech/p/how-container-vulns-get-fixed

[6] Google Cloud, Container scanning overview, 2025. [Online]. Available: https://cloud.google.com/artifact-analysis/docs/container-scanning-overview
[7] OWASP, Container Vulnerability Scanning, 2023. [Online]. Available: https://owasp.org/www-project-devsecops-guideline/latest/02f-Container-Vulnerability-Scanning
[8] NVIDIA-AI-Blueprints, NVIDIA AI Blueprint: Vulnerability Analysis for Container Security. [Online]. Available: https://github.com/NVIDIA-AI-Blueprints/vulnerability-analysis
[9] Babula parida, Docker vulnerability assessment with Trivy and Azure DevOps, 2022. [Online]. Available: https://blog.devgenius.io/docker-vulnerability-assessment-with-trivy-and-azure-devops-60c95a2d05c5
[10] Microsoft, Scan registry images with Microsoft Defender for Cloud, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/container-registry/scan-images-defender
[11] SentinelOne, Container Vulnerability Scanning: A Comprehensive Guide, 2025. [Online]. Available: https://www.sentinelone.com/cybersecurity-101/cloud-security/container-vulnerability-scanning/
[12] OWASP, Source Code Analysis Tools. [Online]. Available: https://owasp.org/www-community/Source_Code_Analysis_Tools
[13] The Jit Team, Top 10 Container Scanning Tools for 2025, 2025. [Online]. Available: https://www.jit.io/resources/appsec-tools/container-scanning-tools-for-2023
[14] Chainguard Academy, Using Trivy to Scan Software Artifacts, 2024. [Online]. Available: https://edu.chainguard.dev/chainguard/chainguard-images/staying-secure/working-with-scanners/trivy-tutorial/
[15] Ramkrushna Maheshwar, Scanning Docker Images for Vulnerabilities: Using Trivy for Effective Security Analysis, 2023. [Online]. Available: https://medium.com/@maheshwar.ramkrushna/scanning-docker-images-for-vulnerabilities-using-trivy-for-effective-security-analysis-fa3e2844db22
[16] Andrew Blooman, Container Security Scanning, 2024. [Online]. Available: https://itnext.io/container-security-scanning-f16b438db58d
[17] Liron Biam, When and How to Use Trivy to Scan Containers for Vulnerabilities, 2024. [Online]. Available: https://www.jit.io/resources/appsec-tools/when-and-how-to-use-trivy-to-scan-containers-for-vulnerabilities
[18] Vaishnavi, Can AI Be Used for Zero-Day Vulnerability Discovery? How Artificial Intelligence is Changing Cybersecurity Threat Detection, 2025. [Online]. Available: https://www.webasha.com/blog/can-ai-be-used-for-zero-day-vulnerability-discovery-how-artificial-intelligence-is-changing-cybersecurity-threat-detection
[19] Zscaler, Can AI Detect and Mitigate Zero Day Vulnerabilities?, 2025. [Online]. Available: https://www.zscaler.com/blogs/product-insights/can-ai-detect-and-mitigate-zero-day-vulnerabilities
[20] Casey Charrier, James Sadowski, Clement Lecigne, Vlad Stolyarov, Hello 0-Days, My Old Friend: A 2024 Zero-Day Exploitation Analysis, 2025. [Online]. Available: https://cloud.google.com/blog/topics/threat-intelligence/2024-zero-day-trends
[21] SentinelOne, Zero-Day Attack Vectors: A Complete Guide, 2025. [Online]. Available: https://www.sentinelone.com/cybersecurity-101/threat-intelligence/zero-day-vulnerabilities-attacks/
[22] Venu Shastri, What is a Zero-Day Exploit?, 2025. [Online]. Available: https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/zero-day-exploit/
[23] ZerodayInitiative, Published Advisories, 2025. [Online]. Available: https://www.zerodayinitiative.com/advisories/published/
[24] Microsoft, Mitigate zero-day vulnerabilities, 2025. [Online]. Available: https://learn.microsoft.com/en-us/defender-vulnerability-management/tvm-zero-day-vulnerabilities
[25] Zero-day, Zero-day Vulnerability Database, (2006-2025). [Online]. Available: https://www.zero-day.cz/database/
[26] Nedim Marić, 5 Examples of Zero Day Vulnerabilities and How to Protect Your Organization, 2024. [Online]. Available: https://www.brightsec.com/blog/5-examples-of-zero-day-vulnerabilities-and-how-to-protect-your-organization/
[27] Defendify, How To Respond to Zero Day Vulnerabilities Once They Become Public. [Online]. Available: https://www.defendify.com/blog/how-to-handle-zero-day-vulnerability/