



Cybersecurity Focus

Project Documentation

Project Title:

Community-Powered Vulnerability Management

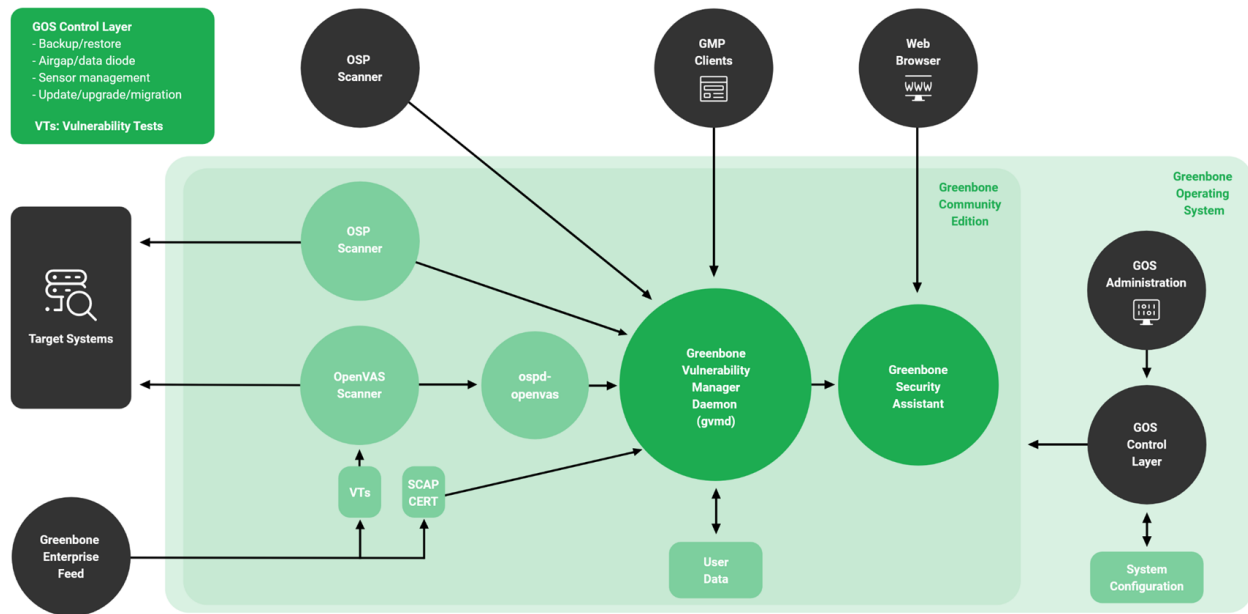
Team Members: Marlon Iglesias Diaz, Theodore Atis, Daniel Granados

Product Owner(s): Masoud Sadjadi

Mentor(s): Masoud Sadjadi

Instructor: Masoud Sadjadi

Contributions from Daniel Granados



Comprehensive Vulnerability Management:

The Greenbone Vulnerability Manager Daemon (gvmd) serves as the central hub for comprehensive vulnerability management, elevating basic vulnerability scanning to a sophisticated solution. By orchestrating the OpenVAS Scanner through the Open Scanner Protocol (OSP), gvmd ensures efficient coordination and execution of vulnerability assessments across diverse systems.

Robust Scanning Engine:

The OpenVAS Scanner, a powerful scanning engine, specializes in executing vulnerability tests (VTs) on target systems. Its effectiveness lies in the utilization of regularly updated and extensive feeds, ensuring that the scanning engine remains equipped to identify the latest vulnerabilities and potential threats.

Vulnerability Categorization and Identification:

OpenVAS employs a systematic approach to vulnerability categorization, utilizing Common Vulnerabilities and Exposures (CVEs) and Common Platform Enumeration (CPEs). This integration taps into the National Vulnerability Database (NVD) and leverages the Common Vulnerability Scoring System (CVSS) managed by the National Institute of Standards and Technology (NIST). This comprehensive framework ensures accurate identification, prioritization, and evaluation of vulnerabilities. The identification process is facilitated through Network Vulnerability Tests (NVTs) or Vulnerability Tests (VTs).

- Key Concepts:
- CVE (Common Vulnerabilities and Exposures): Standardized identification system.

- CPE (Common Platform Enumeration): Standardized naming method for platforms.
 - NVD (National Vulnerability Database): Repository offering detailed vulnerability information.
- CVSS (Common Vulnerability Scoring System): Framework for assessing severity.

German Security Advisories:

DFN-CERT and CERT-Bund advisories from the German Research Network and the German Federal Office for Information Security categorize, distribute, and rate advisories issued by various software vendors. These services play a crucial role in keeping organizations informed about potential security threats and vulnerabilities.

Installation Methods:

Source Installation:

The installation process involves cloning the source code using Git, installing dependencies, configuring the build settings, compiling the source code, and finally, installing the Greenbone components. This method provides a flexible and customizable approach to integrating Greenbone into Linux machines or virtual environments.

Docker Container:

Leveraging the concept of containerization, a Docker container encapsulates all necessary components for running applications. This containerized approach ensures consistency and portability, enabling seamless deployment across diverse environments with minimal dependencies.

Custom NVT Library Creation:

To create a custom Network Vulnerability Test (NVT) library, an effective strategy involves initiating with a simple device-specific Nessus Attack Scripting Language (NASL) script. This script serves as a foundational reference for developing more complex NVTs tailored to specific requirements. Alternatively, modifying existing NASL scripts to suit the unique characteristics of devices, networks, or systems is a practical approach for niche vulnerability testing. This iterative process allows for the continual refinement and expansion of the NVT library based on evolving security needs.

Risk Assessment:

Creating custom Network Vulnerability Tests (NVTs) and tweaking existing ones in the Greenbone system involves some important considerations. We need to be cautious about potential security risks because if these custom tests aren't developed and tested carefully, they could unintentionally introduce new vulnerabilities. To handle this, we must thoroughly review the code, conduct extensive testing, and stick to secure coding practices. Another thing to watch out for is compatibility issues, where the custom tests might not work well with all types of systems. To deal with this, we need to test them in different environments and clearly document what kinds of systems they're suitable for.

Maintaining these custom tests can be a challenge as software changes over time. To keep them effective, we'll need to regularly check and update them. Also, we want to be careful about false positives or negatives, meaning the tests might report problems that aren't really there or miss actual issues. Regular testing and working with the open-source community for feedback can help us refine these custom tests based on real-world experiences. Lastly, developing and maintaining these tests can be a bit resource-intensive, so we need to manage our resources well and prioritize what's most important.

When adapting these tests to specific situations, like in a unique or specialized environment, we need to follow a step-by-step plan. First, we figure out exactly what security needs we have and talk to the people involved to get their input. Then, we design these custom tests to match the special features of the environment and test them really well to make sure they work right. We also need to write down clear instructions on how to use these tests and any limitations they might have. This whole process is about continuous improvement, meaning we keep making these tests better based on what we learn from actually using them. And if we share our custom tests with the open-source community, it's like working together with others to make security tools that can help lots of people facing similar challenges.

Contributions from Marlon Iglesias Diaz

PURPOSE

Collective Expertise: Harnessing the knowledge of a diverse community to identify vulnerabilities.

Rapid Detection: Accelerating the discovery of security threats through collaborative efforts.

Effective Mitigation: Enhancing cybersecurity resilience by leveraging shared insights for prompt action.

1. Introduction to Vulnerability Management

Vulnerability management is a crucial aspect of cybersecurity aimed at identifying, assessing, and mitigating security vulnerabilities within an organization's IT infrastructure. This report delves into various components of vulnerability management and their significance in ensuring the security and integrity of digital systems.

2. Concept of Vulnerability Management and its Importance in Cybersecurity

Vulnerability management involves a structured process of discovering, assessing, prioritizing, and addressing security vulnerabilities in software, hardware, and networks. Its importance in cybersecurity lies in the proactive approach to reducing the organization's attack surface, enhancing threat detection, and minimizing potential risks. Effective vulnerability management helps in maintaining compliance with security standards and regulations.

3. National Vulnerability Database (NVD)

The National Vulnerability Database (NVD) is a repository of vulnerability management data maintained by the U.S. government. It utilizes the Security Content Automation Protocol (SCAP) for data representation, enabling automation in vulnerability management, security measurement, and compliance. NVD includes databases of security checklist references, software flaws, misconfigurations, and more.

4. Known Exploited Vulnerability (KEV) Catalog

The Cybersecurity and Infrastructure Security Agency (CISA) is responsible for maintaining the Known Exploited Vulnerability (KEV) Catalog. This catalog serves as the authoritative source for vulnerabilities that have been actively exploited. Organizations are encouraged to review and monitor the KEV catalog to prioritize the remediation of listed vulnerabilities, thereby reducing the risk of compromise.

5. Common Vulnerabilities and Exposures (CVE) Program

The Common Vulnerabilities and Exposures (CVE) Program has the mission to identify, define, and catalog publicly disclosed cybersecurity vulnerabilities. It assigns a unique CVE Record to each vulnerability, ensuring consistent descriptions for effective coordination. CVE Records are published by organizations worldwide in partnership with the CVE Program.

6. Network Vulnerability Test (NVT)

Network Vulnerability Tests (NVTs) are scripts executed on a targeted system to perform vulnerability checks, either remotely or locally. NVTs encompass vulnerabilities that have been assigned a CVE, but they can also cover vulnerabilities without a referenced CVE, broadening the scope of vulnerability assessments.

7. OpenVAS Vulnerability Scanner

OpenVAS is a comprehensive vulnerability scanner equipped with various capabilities. It supports both unauthenticated and authenticated testing, making it versatile in assessing different aspects of system security. OpenVAS covers a wide range of high-level and low-level internet and industrial protocols and offers performance tuning for large-scale scans. Additionally, it features a powerful internal programming language for customization and flexibility.

8. Greenbone and OpenVAS

Greenbone has been the driving force behind OpenVAS since 2006. OpenVAS is an integral part of the Greenbone Community Edition and is also included in the Greenbone Enterprise Appliance, providing organizations with a robust and effective solution for vulnerability management.

9. Vulnerability Management

In conclusion, vulnerability management is an essential practice in the realm of cybersecurity. The National Vulnerability Database, Known Exploited Vulnerability Catalog, Common Vulnerabilities and Exposures Program, Network Vulnerability Tests, and OpenVAS Vulnerability Scanner are valuable tools and resources that play a vital role in identifying, assessing, and mitigating vulnerabilities, ultimately strengthening an organization's security posture. The collaboration between Greenbone and OpenVAS underscores the commitment to providing robust solutions in this critical area of cybersecurity.

10. GreenBone Community Feed

GreenBone Community Feed vs Enterprise Feed

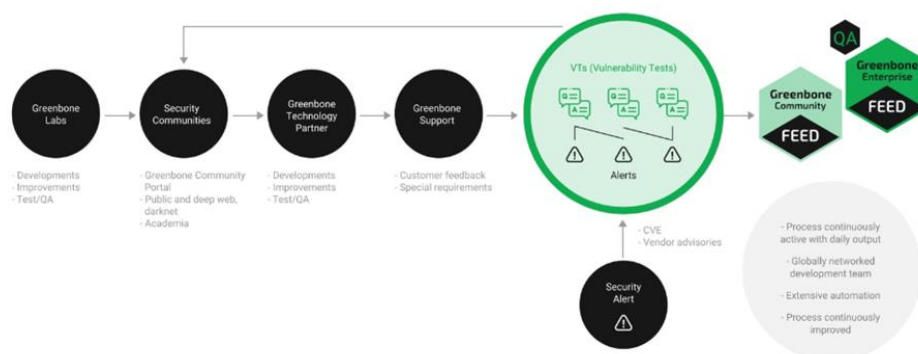
	GREENBONE COMMUNITY FEED (BASIC COVERAGE)	GREENBONE ENTERPRISE FEED (ADVANCED COVERAGE)
Home Application (only) Products (e.g., Ubuntu Linux, AVM Fritzbox, MS Office)	✓	✓
German Policy IT-Grundschutz	✓	✓
Enterprise grade products (e.g., MS Exchange, Palo Alto, Cisco, IoT/OT)	X	✓
Compliance Policies for CIS Benchmarks	X	✓
Additional Policies	X	✓
Access to Greenbone Enterprise Support	X	✓
Access to Professional Services	X	✓

Process for creating the Greenbone Enterprise Feed *(From Developer GreenBone)*

The GreenBone OpenVAS scanner incorporates a wide array of sources for security messages, including security communities, technology partners, customer input, and internal Greenbone labs, among others. For every security message, an automated process generates a corresponding ticket

within the VT (Vulnerability Test) management system. These tickets undergo thorough examination and scrutiny in the product's designated labs, where they are subjected to comprehensive investigation. Following successful implementation and rigorous quality assurance, these tickets are integrated into the feed service.

Furthermore, the product applies a consistent testing regimen to VTs that have been fully implemented by technology partners or contributed by security communities. This meticulous testing approach underscores the product's unwavering commitment to maintaining the highest standards of quality.

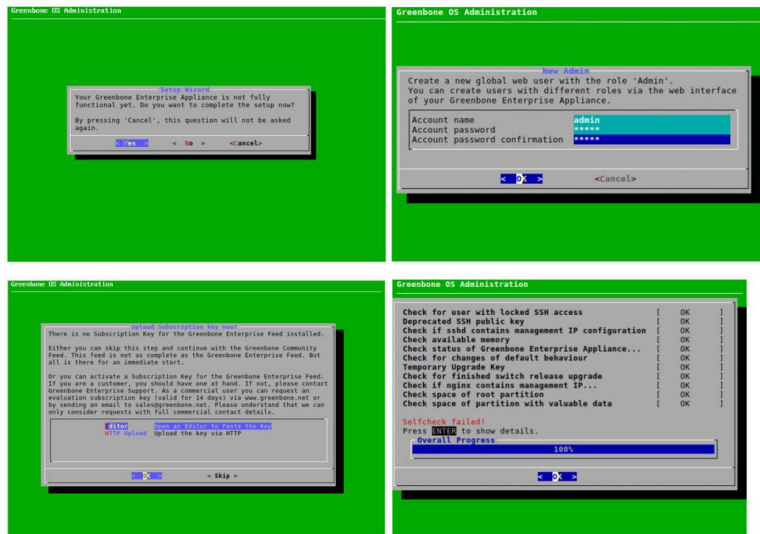


11. Installation

Through the utilization of the GreenBone Enterprise Trial, you have the option to download an OVA file and execute the installation process within Oracle VirtualBox. This installation process leads to the introduction of an imported appliance within your VirtualBox environment. This appliance streamlines the process of launching a fully equipped Virtual Machine (VM) designed for the execution of the OpenVAS scanner.

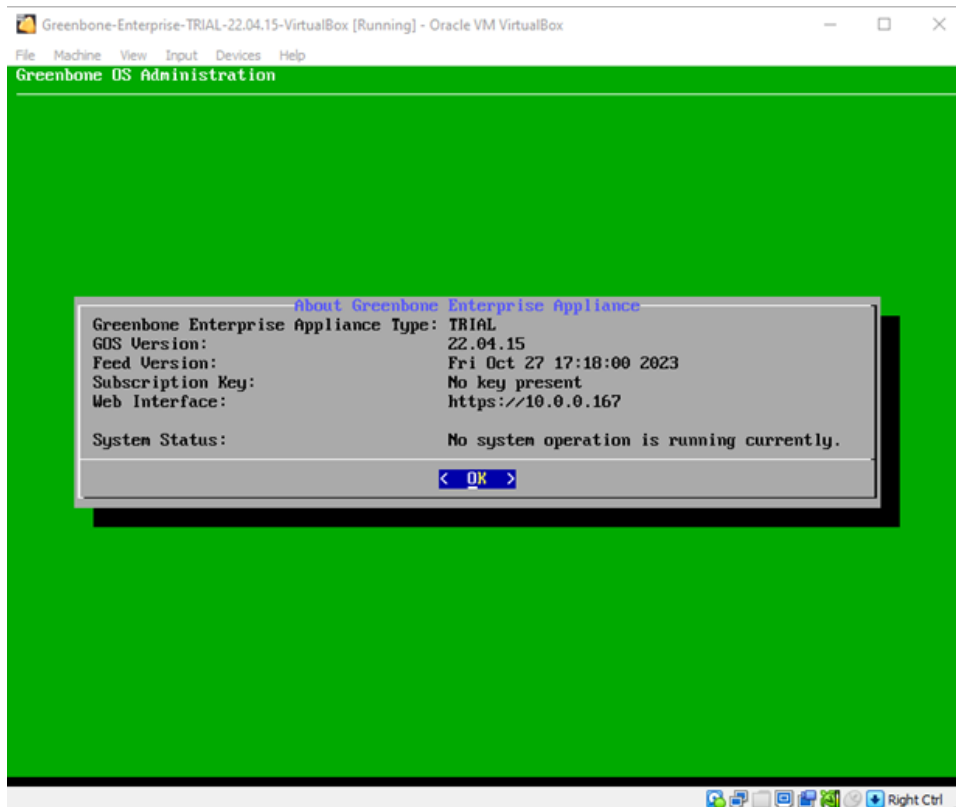
This approach stands out as the most straightforward and efficient means of locally running the OpenVAS scanner. Other methods such as Docker or running it via the command line have resulted in numerous dependency-related issues that we encountered during the initial weeks of our research into the OpenVAS scanner.

Here, you can see images illustrating a successful setup process and the subsequent scanning activities.



12. Operational

This is an active VM state without a subscription key, indicating usage of the Community Feed rather than the Enterprise Feed, including timestamps and version numbers that demonstrate recent updates.



Here is the current feed status for the VM when accessed through the web interface outside the VM.

The screenshot shows the Greenbone Enterprise Appliance web interface. The top navigation bar includes links for Dashboards, Scans, Assets, Resilience, SecInfo, Configuration, Administration, and Help. The main content area is titled 'Feed Status' and displays a table with the following data:

Type	Content	Origin	Version	Status
NVT	NVTs	Greenbone Community Feed	20231027T1718	3 days old
SCAP	CVEs, CPEs	Greenbone Community SCAP Feed	20231027T0504	3 days old
CERT	CERT-Bund Advisories, DFN-CERT Advisories	Greenbone Community CERT Feed	20231027T0407	3 days old
GVMD_DATA	Compliance Policies, Port Lists, Report Formats, Scan Configs	Greenbone Community gvmd Data Feed	20231027T0504	3 days old

13. Demonstrated Need (Log4J Vulnerability)

CISA, in collaboration with partners, is addressing an active and widespread remote code execution vulnerability (CVE-2021-44228) in Apache's Log4j software library, known as "Log4Shell." This vulnerability affects Log4j versions 2.0-beta9 to 2.14.1, posing a significant threat due to its widespread use across consumer and enterprise services, websites, and operational technology products. Unauthorized remote actors can exploit this vulnerability to take control of affected systems. To effectively address this threat, CISA advises to consider the following steps:

Key Points

Verify Scope: Assess all assets for Log4j usage, including internal software and non-internet facing technology stacks.

Comprehensive Enumeration: Conduct thorough analysis, including network and host scanning, to identify Log4J vulnerabilities. Cross-reference software with CISA's Log4j vulnerable software database.

High Fidelity Scanning: Use file system scanning scripts or vulnerability scanners that employ file scanning to detect vulnerable Log4j files.

Monitor 3rd Party Software: Continuously review CISA's log4j vulnerable software database and check for vulnerable software in third-party applications, including SaaS and cloud resources.

Patching Priority: Apply updates immediately if available. If updating is not possible, implement temporary mitigation measures, as listed in "Mitigating Log4Shell and Other Log4j-Related Vulnerabilities." However, patching is preferred over mitigation. If patching or mitigation is not possible, consider removing the vulnerable asset from the network.

CISA further advises organizations to upgrade to Log4j 2.17.1 (Java 8), 2.12.4 (Java 7) and 2.3.2 (Java 6), and review and monitor the Apache Log4j Security Vulnerabilities webpage for updates and mitigation guidance.

14. Implementation

We can make use of the OpenVAS Configuration to check for Log4J vulnerabilities, including checks for Log4j and CVE-2021-44228.

Presented below is the Scan Config outlining the NVT families responsible for detecting the Log4j vulnerability.

Family	NVTs selected	Trend
	43 of 0	→
Debian Local Security Checks	3 of 16595	→
Fedora Local Security Checks	3 of 23493	→
General	5 of 7390	→
Port scanners	2 of 9	→
Product detection	3 of 2933	→
Service detection	1 of 253	→
Settings	1 of 11	→
Ubuntu Local Security Checks	2 of 13283	→
Web application abuses	9 of 8881	→

PROJECT REVIEW

Problem

Organizations face increased cyber threats due to vulnerabilities in their systems. Without an efficient vulnerability management solution, these weaknesses expose the organization to potential threats, leading to significant risks. Our team addresses this security gap utilizing two separate solutions, Greenbone Vulnerability Management (GVM) and GitLab CI/CD.

Current System

GVM is a comprehensive solution designed to identify, assess, and manage vulnerabilities within an organization's IT infrastructure. Employing proactive scanning techniques, Greenbone systematically analyzes networks, systems, and applications to pinpoint potential security weaknesses. The platform facilitates the categorization and prioritization of vulnerabilities, enabling organizations to allocate resources efficiently and address high-risk areas promptly. GitLab DevSecOps represents the convergence of development, security, and operations, offering a holistic approach to software development by seamlessly integrating security measures throughout the entire development lifecycle. CI/CD, is an acronym for Continuous Integration/Continuous Deployment, entails a continuous cycle of building, testing, deploying, and monitoring iterative code changes, fostering an environment of constant improvement and efficiency in the software development process.

System Design

GVM

The utilization of the Greenbone Enterprise Trial provided our team with a swift and straightforward testing environment for our Linux-based appliance solutions. We efficiently configured the GVM, gaining access through VirtualBox. Additionally, we explored Docker and

Ubuntu as alternative methods for GVM setup, concluding that the Enterprise Trial offered the most user-friendly experience as it included all dependencies required for successful testing. GitLab

Within GitLab, I established a dedicated Group for our team's various repositories and subsequently created a Project to host a code repository. This repository was specifically designed to conduct comprehensive tests on different facets of DevSecOps. The tests encompassed various dimensions of application security within GitLab, evaluating aspects such as source code, dependencies, and the overall infrastructure as code. This structured approach ensured a systematic examination of the security aspects throughout our development and deployment processes.

Testing



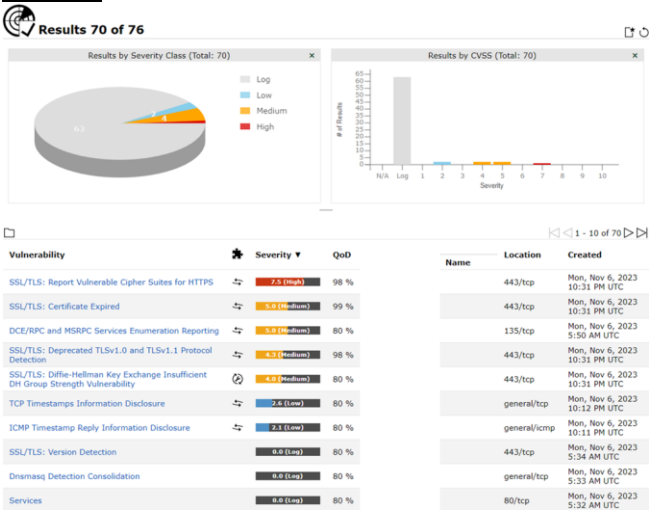
1 - 7 of 7					
Name ▲	Family		NVTs		Actions
	Total	Trend	Total	Trend	
Base (Basic configuration template with a minimum set of NVTs required for a scan. Version 20200827.)	2	→	3	→	  
Discovery (Network Discovery scan configuration. Version 20201215.)	10	→	3206	↗	  
empty (Empty and static configuration template. Version 20201215.)	0	→	0	→	  
Full and fast (Most NVT's; optimized by using previously collected information. Version 20201215.)	58	↗	133115	↗	  
Host Discovery (Network Host Discovery scan configuration. Version 20201215.)	2	→	2	→	  
Log4Shell (Configuration with checks for Log4j and CVE-2021-44228. Version 20211227.)	10	→	29	→	  
System Discovery (Network System Discovery scan configuration. Version 20201215.)	5	→	30	→	  

Here are the different scan configs available for setup on GVM through appliance setup of Enterprise Trial edition.

```
ci > security_scans.gitlab-ci.yml
1  ---
2  # All GitLab Security Scans and Reports with downloadable artifacts that expire in 16 weeks.
3
4  gemnasium-dependency_scanning:
5    stage: test
6    needs: [build-job]
7    allow_failure: true
8    interruptible: true
9    script:
10     - echo NOOP
11    artifacts:
12      reports:
13        dependency_scanning: samples/dependency-scanning.json
14        expire_in: 16 weeks
15
16  container-scanning:
17    stage: test
18    needs: [build-job]
19    allow_failure: true
20    interruptible: true
21    script:
22     - echo NOOP
23    artifacts:
24      reports:
25        container_scanning: samples/container-scanning.json
26        expire_in: 16 weeks
27
28  dast:
29    stage: test
30    needs: [build-job]
31    allow_failure: true
32    interruptible: true
33    script:
34     - echo NOOP
35    artifacts:
36      reports:
37        dast: samples/dast.json
38        expire_in: 16 weeks
39
```

Pictured above is some of the YAML code that went into setting up the GitLab security tests.

Results



Above are some of the GVM scan results showed vulnerabilities and their severity.

Scan details				Hide details
DAST	4 vulnerabilities	Download results		
SAST	60 vulnerabilities	Download results		
Container Scanning	8 vulnerabilities	Download results		
API Fuzzing	6 vulnerabilities	Download results		
Coverage Fuzzing	1 vulnerability	Download results		
Secret Detection	5 vulnerabilities	Download results		

Severity	Tool	Hide dismissed	
All severities	All tools		

☐	Severity	Vulnerability	Identifier	Tool	
☐	Critical	eval with argument of type Identifier src/html/index.html	ESLint rule ID security/detect-eval-with-expression + 1 more	SAST GitLab	
☐	Critical	Generic Object Injection Sink src/html/index.html	ESLint rule ID security/detect-object-injection + 1 more	SAST GitLab	
☐	Critical	eval with argument of type Identifier src/js/main.js	ESLint rule ID security/detect-eval-with-expression + 1 more	SAST GitLab	

In contrast here are the various successful GitLab security tests and their artifacts. When you select a specific vulnerability you can see a window that looks like this talking about the cause and solution to fix.

Deserialization of Untrusted Data

Description

A possible escalation to RCE vulnerability exists when using YAML serialized columns in Active Record < 7.0.3.1, <6.1.6.1, <6.0.5.1 and <5.2.8.1 which could allow an attacker, that can manipulate data in the database (via means like SQL injection), the ability to escalate to an RCE.

Severity: Critical

Project: [Vulnerability_Management](#) / [DevSecOps_Security_Reports](#)

Tool: Dependency Scanning

Scanner: Gemnasium

Location

File: [dependency-scanning-files/Gemfile.lock](#)

Links

- <https://discuss.rubyonrails.org/t/cve-2022-32224-possible-rce-escalation-bug-with-serialized-columns-in-active-record/81017>
- <https://github.com/advisories/GHSA-3hhc-qp5v-9p2j>
- <https://github.com/rails/rails/commit/611990f1a6c137c2d56b1ba06b27e5d2434dcd6a>
- <https://github.com/rubysec/ruby-advisory-db/blob/master/gems/activerecord/CVE-2022-32224.yml>
- <https://groups.google.com/g/rubyonrails-security/c/MmFO3LYQE8U>
- <https://nvd.nist.gov/vuln/detail/CVE-2022-32224>

Identifiers

- [GHSA-3hhc-qp5v-9p2j](#)
- [CVE-2022-32224](#)
- [Gemnasium-b60c2d6b-9083-4a97-a1b2-f7dc79bff74c](#)

Solution

Upgrade to versions 5.2.8.1, 6.0.5.1, 6.1.6.1, 7.0.3.1 or above.

Lastly, here’s a summary of all the vulnerabilities on a given project.

Vulnerability report

+ Submit vulnerability

⬆ Export

The Vulnerability Report shows results of successful scans on your project's default branch, manually added vulnerability records, and vulnerabilities found from scanning operational environments. [Learn more.](#)

Development vulnerabilities 187 Operational vulnerabilities 0

Security reports last updated 1 week ago #1070623396 ⚠ Parsing errors in pipeline • SBOMs last updated 1 week ago #1070623396

● Critical	◆ High	▼ Medium	● Low	● Info	● Unknown
17	61	83	7	0	19

Status	Severity	Tool	Activity
Needs triage +1 more	All severities	All tools	All activity

Group by: None

☐ Detected	Status	↓ Severity	Description	Identifier	Tool	Activity
☐ 2023-11-13	Needs Triage	● Critical	Deserialization of Untrusted Data dependency-scanning-files/Gemfile.lock	CVE-2022-32224 + 2 more	Dependency Scanning GitLab	

Risk Assessment

This risk assessment evaluates outcomes derived from both GVM and GitLab CI/CD scans. GVM scans, leveraging customized configurations, successfully identified critical vulnerabilities, including the Log4Shell vulnerability, with the flexibility to tailor scans predictively through the intuitive GUI of the enterprise trial edition.

In the GitLab CI/CD environment, I authored custom YAML code to orchestrate diverse scanners aimed at detecting potential vulnerabilities in a test application. These scanners encompass Static Application Security Testing (SAST) and Secret Detection for source code analysis, Dynamic Application Security Testing (DAST) and API fuzzing for assessing running web applications, and Dependency Scanning and Container Scanning for analyzing and cross-referencing the app's dependencies against a database of known vulnerabilities.

Summary

This comprehensive approach to DevSecOps ensures a thorough evaluation of security aspects, effectively addressing source code vulnerabilities, secrets, runtime application security, and potential risks associated with dependencies in the development and deployment processes.

Contributions from Theodore Atis

Wazuh

1. Wazuh Indexer:

The Wazuh indexer is an exceptionally scalable, comprehensive search, and analytics engine. This pivotal element is responsible for indexing and storing alerts generated by the Wazuh server. It offers real-time data search and analytics capabilities, allowing for configuration as a single-node or multi-node cluster to ensure scalability and high availability. The Wazuh indexer stores data in JSON documents, each correlating keys with their respective values. Through distribution across multiple shards and nodes, the Wazuh indexer ensures redundancy, safeguarding against hardware failures and enhancing query capacity.

2. Wazuh Server:

The Wazuh server analyzes data from agents, employing decoders and rules, integrating threat intelligence to identify indicators of compromise. It manages configurations and remote upgrades of agents, supporting hundreds or thousands of agents and horizontal scaling when configured as a cluster. The server enriches alert data using the MITRE ATT&CK framework and various compliance requirements. It seamlessly integrates with external tools like ServiceNow, Jira, Slack, and others, streamlining security operations.

Server Architecture:

The server houses the analysis engine, Wazuh RESTful API, agent enrollment, agent connection services, Wazuh cluster daemon, and Filebeat. It operates on Linux, whether on standalone machines, virtual machines, docker containers, or cloud instances.

3. Wazuh Dashboard:

The Wazuh dashboard serves as a web user interface for data visualization and analysis. It offers out-of-the-box dashboards for security events, regulatory compliance, vulnerable applications, file integrity monitoring, configuration assessments, and more. Beyond visualization, the dashboard facilitates Wazuh platform management, role-based access control, and single sign-on.

Dashboard Features:

Users can navigate, generate reports, and create custom visualizations. It supports auditing, policy monitoring, threat detection, regulatory compliance, and agents' monitoring and configuration. The dashboard aids in platform management, offering insights into status, logs, and statistics of Wazuh components.

4. Wazuh Agents:

Wazuh agents, deployed on various endpoints, provide threat prevention, detection, and response capabilities. They operate on diverse operating systems such as Linux, Windows, macOS, Solaris, AIX, and HP-UX.

Agent Architecture:

Agents feature a modular architecture encompassing log collection, command execution, file integrity monitoring, security configuration assessment, system inventory, malware detection, active response, container security monitoring, and cloud security monitoring. Agents communicate securely with the Wazuh server, facilitating remote configuration, monitoring, and upgrades.

Vulnerabilities

5. Vulnerability Detection:

Wazuh agents periodically collect software inventory data, sending it to the Wazuh server. The Vulnerability Detector module correlates this data with vulnerability feeds, identifying vulnerabilities and producing risk reports. The module supports various operating systems, including Windows, CentOS, Red Hat Enterprise Linux, Ubuntu, Debian, Arch Linux, and macOS. The Wazuh dashboard displays alerts grouped by severity, providing a comprehensive view of identified vulnerabilities.

Vulnerability Alerts:

Alerts contain key information such as severity, affected package details, references, CVE information, and remediation advice. The system allows tracking vulnerability remediation activities, confirming when a vulnerability has been resolved. Users can download reports to identify unresolved vulnerabilities and track remediation progress.