TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Nikita Balanenkov 201726IVSB

Managing Used Dependencies: Detecting and Mitigating Security Vulnerabilities in Java-based Web Applications

Bachelor's thesis

Supervisor:	Toomas Lepikult
	PhD
Co-supervisor:	Edmund Laugasson
	MSc

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Nikita Balanenkov 201726IVSB

Kasutatavate sõltuvuste haldamine: turvanõrkuste avaldamine ja likvideerimine Java-põhistes veebirakendustes

Bakalaureusetöö

Juhendaja:	Toomas Lepikult
	PhD
Kaasjuhendaja:	Edmund Laugasson
	MSc

Tallinn 2023

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Nikita Balanenkov

14.05.2023

Abstract

This bachelor’s thesis aims to compare different dependency scanning tools and demonstrate their practical usage in software development. The first part of the thesis introduces the concept of dependency scanning and explains its importance in modern software development. Then, several popular dependency scanning tools are compared based on different criteria.

The second part of the thesis provides step-by-step instructions on how to use selected dependency scanning tools in practice. The usage instructions cover how to set up scanners, configure them, and interpret the results of the scan. Additionally, the thesis includes a demonstration of using selected dependency scanning tools on a sample project.

The final part of the thesis presents a case study of a software development company that adopted a dependency scanning tool as part of its development process. The case study showcases how the scanner improved the company's software quality, reduced security vulnerabilities, and helped the developers to manage their dependencies more effectively.

Overall, this thesis provides a comprehensive overview of dependency scanning tools and demonstrates their practical usage in software development. It is a valuable resource for developers, software engineers, and anyone interested in improving the security and reliability of their software projects.

This thesis is written in English and is 70 pages long, including 8 chapters, 33 figures and 9 tables.

List of abbreviations and terms

AST	Abstract Syntax Tree
APK	Android Application Package
AWS	Amazon Web Services
BOM	Bill Of Materials
CI/CD	Continuous Integration/Continuous Delivery or Continuous Deployment
CLI	Command Line Interface
CSV	Comma Separated Values
CVSS	Common Vulnerability Scoring System
DOS	Denial of Service
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JAR	Java Archive
JSON	JavaScript Object Notation
NVD	National Vulnerability Database
ODC	OWASP Dependency Check
OWASP	Open Web Application Security Project
PDF	Portable Document Format
RAM	Random Access Memory
RCE	Remote Code Execution
SaaS	Software as a Service
SARIF	Static Analysis Results Interchange Format
SBOM	Software Bill Of Materials
SCA	Software Composition Analysis
SQL	Structured Query Language
WAR	Web Application Resource or Web application Archive
XML	Extensible Markup Language
XSS	Cross-Site Scripting

Table of contents

List of figures

List of tables

1 Introduction

To keep pace with the demands of feature-rich and complex web applications, software developers commonly rely on third-party libraries and frameworks to expedite development processes. However, while such external components are beneficial for accelerating work output, they give way for cybercriminals to hack vulnerabilities into a system through which sensitive information can be leaked or destructive codes introduced, disrupting web application operations. As such, overseeing dependencies has emerged as fundamental to securing robustness within contemporary web apps. The complex structure of larger projects makes staying ahead of potential security threats in Java-based web applications challenging at best.

However, thanks to recent advancements made with dependency scanning tools, it is now possible to quickly detect and remedy such vulnerabilities effectively. By analysing dependencies used throughout the coding ecosystem, scanning systems highlight common causes for concern, including known weaknesses that could expose sensitive data, alongside expert recommendations on how best to mitigate them. As part of an overall workflow strategy for better online safety practices, integrating such tools into day-to-day development protocols should be considered essential.

The primary objective of this bachelor's thesis is to examine how dependency scanning tools play a crucial role in the management of dependencies, specifically within Java-based web applications. The study will investigate various techniques and approaches used to detect and mitigate potential security vulnerabilities associated with third-party code. The research will commence by examining the significance of dependency management and the possible dangers when utilizing external sources.

This thesis will then compare different dependency scanning tools' effectiveness in identifying vulnerable areas within a test application, appraise them, and present their results. It will conclude by providing readers with guidelines on installing and running dependency scanning tools for managing dependencies and guaranteeing the safety of web applications. Through analysing prevalent literature relating to the subject matter, studying case studies, and carrying out practical experiments employing an array of dependency scanning tools, this research aims to generate vital insights concerning dependency management best practices as well as guarantee web application security. The thesis results can serve as a resource tool for stakeholders invested in web application development, such as software developers, project managers, or any other interested party engaged in software development activities.

1.1 Problem setup

Using vulnerable dependencies in the code is often a cause of security vulnerabilities. The goal of the thesis is to try to pay attention to keeping dependencies updated by proposing and analyzing different tools for the automatic detection of vulnerable dependencies in theory and practice. The thesis deals with helping to mitigate security vulnerabilities in Java-based web applications by providing possible options for detecting vulnerable dependencies that could be a source for such vulnerabilities. The thesis will not provide a direct solution to the problem, such as an actual application capable of such detection; this could be done as a continuation of the work.

1.2 Methodology

This thesis heavily relies on quantitative research involving the methods described below:

- Conducting a literature review to identify relevant studies, research papers, and articles related to the use of vulnerable dependencies in software development. Analysing the findings and summing up the key points and insights.

- Conducting a comprehensive analysis of the security risks associated with using vulnerable dependencies and exploring real-world security breaches that were caused by such dependencies. Analysing the impact of these breaches and their consequences.

- Searching for tools that can detect vulnerable dependencies automatically. Evaluating the tools based on several criteria. Analysing their features and functionalities and comparing them to other similar tools available on the market.

- Selecting the free tools identified in the previous step and conducting experimental research to test their effectiveness in detecting vulnerable dependencies. Using a sample application to evaluate the tools' performance and analysing the results.

- Based on the experimental research results, developing a list of instructions that outline the steps to be followed when using the tools to detect and mitigate vulnerable dependencies.

- Providing a practical example of how the tools can be used in a software development company to detect vulnerable dependencies in applications. Outlining the steps involved in the process and providing a detailed description of the tools used. Analysing the benefits of using the tools and how they can help the company mitigate security risks associated with vulnerable dependencies.

2 Background

The following chapter will provide basic information about dependencies and dependency vulnerability scanning tools to give readers a general understanding of the topic.

2.1 Dependencies

In programming, dependencies refer to external modules, libraries, or packages that are necessary for the proper execution of the code.

External resources are commonly produced and sustained by developers who are not affiliated with the primary software project. These resources are usually accessed via package managers such as npm (for JavaScript), pip (for Python), or Maven (for Java).

Dependencies can take many forms, including code libraries, frameworks, or even entire software systems. For a code snippet to utilize a dependency, it is essential that the said dependency is installed and configured appropriately within the project. Frequently, this process involves the explicit declaration of the interdependence within a configuration file or script, such as a pom.xml file utilized by a Maven-based Java web application or a build.gradle file used by a Gradle-based Java web application. In software development, dependencies can be classified as either direct or indirect[1].

2.1.1 Direct dependencies

Direct dependencies in Java refer to libraries or modules that are explicitly referenced in your code. These are resources that your code directly depends on to function correctly. Using Spring Boot as an example, Spring Boot would be a direct dependency of the project when creating a Java web application. Import statements that reference Spring Boot classes and interfaces would be included in the code[2].

2.1.2 Indirect dependencies

Indirect dependencies in Java are dependencies that are required by your direct dependencies. These dependencies are not referenced directly in your code but are needed by the libraries or modules that your code depends on. For example, Spring Boot may have dependencies on other modules, such as Jackson for JSON processing. Jackson would be an indirect dependency on your project because it's required by Spring Boot but not directly referenced in your code[2].

2.2 Dependency scanning tools

Dependency scanning tools are software engineering instruments utilized to detect and handle the dependencies of a given project. The aforementioned tools conduct an analysis of a given project's source code, configuration files, and other related assets in order to identify the external libraries, modules, or packages upon which it depends. Upon identification, dependency scanning tools can assist developers in verifying the currency, compatibility, and security of all dependencies.

The process of dependency scanning holds significant importance in projects that are either large or complex and possess a multitude of dependencies. Inadequate management of dependencies may result in outdated or incompatible dependencies, which can cause performance degradation, security vulnerabilities, and other related issues. Through the utilization of a dependency scanning tool, software developers are able to expeditiously detect and revise obsolete dependencies, thereby guaranteeing the stability and security of their projects.

A wide range of dependency scanning tools are currently available, ranging from rudimentary command-line utilities to intricate enterprise-grade alternatives. Several commonly used alternatives comprise Sonatype Nexus, DependencyTrack, Snyk, and OWASP Dependency Check. The aforementioned tools provide a variety of functionalities, including but not limited to dependency visualization, automated dependency resolution, and vulnerability scanning. The utilization of a dependency scanning tool enables developers to guarantee that their projects are constructed on a stable basis of secure, current, and efficiently managed dependencies[3][4].

3 Analysis

The following chapter will provide a comprehensive analysis of the security risks associated with using vulnerable dependencies and an overview of real-world security breaches that were caused by the use of such dependencies.

3.1 Analysis of risks caused by using vulnerable dependencies in web applications

In the field of software development, the use of dependencies could introduce vulnerabilities in applications.

These flaws are typically characterized by different types of threats, such as cross-site scripting (XSS), which originates from accepting unsanitized data as part of a webpage without verification or sanitization. Additionally, SQL injection can also result from failing to validate unsanitized parameters before using them to construct queries, leading to unauthorized access to or exposure of confidential information.

Apart from cross-site scripting and SQL injection, three more types of vulnerabilities are identified: remote code execution (RCE), denial of service (DoS), and information disclosure.

A critical security risk is remote code execution; attackers can use vulnerable dependencies to execute harmful code onto a victim's system, granting them considerable access rights.

The second type of hazard is DoS, where the exploitation of deficient security resources may result in the destruction or overloading of systems, starving genuine users of access to essential services.

Lastly, data breaches through susceptibility within dependent libraries are sources for information disclosure attacks carried out with specific intentions.

3.2 Analysis of previous breaches and attacks caused by using vulnerable dependencies in web applications

Vulnerable dependencies have been the source of dozens of data breaches and cyber security attacks, some of them are highlighted below:

- **Equifax, 2017.** Approximately 148 million Americans had their sensitive data exposed in a data breach sustained by Equifax in 2017, along with around fifteen million British citizens and roughly nineteen thousand Canadian residents. The flaw that enabled this compromise within the financial information giant's computer systems resulted from the exploitation of a vulnerability discovered within an open-source web application framework titled Apache Struts; more specifically, this security defect occurred due to Jakarta Multipart parser issues related to improperly handled exceptions throughout file uploads processed through HTTP headers present within Struts version numbers ranging from between two point three through two point five[5].

- **Capital One, 2019**. Capital One fell victim to a cyber-attack in 2019, which exploited a weakness in their web application firewall due to an incorrect configuration of firewall rules. Such inadequate settings paved the way for the invader to launch manipulated requests against the firewall, breaching its defenses. The attacker penetrated the resilient Apache Struts software and executed instructions that granted her access to data stored in S3 buckets[6].

- **Target, 2013.** A vulnerable point in a third-party vendor's software that had become part and parcel of Target's payment system allowed hackers to gain entrance into its payment infrastructure. The assailants took advantage of stolen third-party network credentials traced back to this same inadequacy, which furnished them direct access to Target’s network, allowing infiltration at ease. Using malware meant for RAM scraping purposes, the attackers were then able to siphon vital data being processed by Target[7][8].

- **Mariott International, 2018.** As investigations into the Marriott data breach show, an exploitable loophole existed within their third-party reservation system, which provided cybercriminals with an opportunity to gain unauthorized entry into the network. It has been established that authentication controls were at fault for failing to prevent such access from taking place, which ultimately triggered this catastrophic event for Marriott[9].

- **Heartbleed, 2014**. The Heartbleed bug (CVE-2014-0160) was discovered in OpenSSL, a popular open-source software library used to secure web communications. The theft of sensitive information, such as private keys and passwords, was enabled through the exploitation of a bug that compromised vulnerable servers; among the websites affected were Yahoo, Pinterest, and GitHub[10].

- **WannaCry, 2017**. The WannaCry ransomware attack wreaked havoc on over 200 thousand computers dispersed across 150 countries due to a critical flaw in Microsoft Windows' system (CVE-2017-0144). This flaw was reportedly discovered by National Security Agency operatives, and its details were leaked by a hacker collective known as the Shadow Brokers[11].

4 Overview of dependency scanning tools

The following chapter will provide an overview of dependency scanning tools: existing dependency scanning tools, technologies used and theoretical comparison of those tools.

4.1 Existing dependency scanning tools

Existing dependency scanning tools include the following options:

- **OWASP Dependency Check** is an open-source technology that scans application dependencies to locate any known vulnerabilities. It has the ability to support a number of file types, including WAR, JAR, and APK. The software takes advantage of a mixture of internal and public vulnerability databases to detect these known weaknesses. OWASP Dependency Check can be easily integrated into the development cycle, providing constant scanning for these particular issues[12].

- **Sonatype Nexus Lifecycle** is a commercial tool that provides in-depth analysis of Java-based application dependencies. It offers detailed information about used components, including their license information, security vulnerabilities, and potential risks. The tool also provides actionable intelligence to help developers and security teams remediate vulnerabilities and ensure compliance with industry standards[13].

- **Snyk** is a powerful dependency scanning tool that offers developers detailed information about used components, license information, potential risks, and security vulnerabilities. Its ability to provide actionable intelligence ensures compliance with industry standards while also helping developers and security teams remediate vulnerabilities proactively. Furthermore, Snyk continuously monitors applications and alerts users to newly discovered vulnerabilities[14].

- **Black Duck** offers an array of functionalities for companies seeking a comprehensive solution for managing software security risks associated with Java-based applications. Its ability to thoroughly scan application dependencies enables the identification of components used in application development, along with license details and known vulnerabilities[15].

- **DependencyTrack** is an open-source tool that helps organizations manage and control their software dependencies. It scans Java-based applications to detect known vulnerabilities in their dependencies and provides actionable intelligence to remediate the vulnerabilities. It also offers a risk scoring system that prioritizes vulnerabilities based on their severity[16].

- **Mend SCA**, as a commercial software tool, examines the dependencies of Java-based applications to pinpoint any known vulnerabilities that may compromise their integrity. License compliance information for open-source components is also provided, complemented by expert recommendations to remediate any issues that are detected. This program can be seamlessly integrated into software development processes as it continuously monitors and reports new vulnerability findings[17].

- **Aqua Security** is a commercial tool that provides vulnerability scanning and risk assessment for Java-based applications. It scans application dependencies to detect known vulnerabilities and provides detailed information about their severity and remediation recommendations. The tool can also help to ensure compliance with industry standards, such as the OWASP Top 10[18].

- **XRay**, Developed by JFrog, can be utilized as a tool for identifying known vulnerabilities in Java-based application dependencies. The scanner integrates seamlessly with popular development tools such as Jenkins and JFrog Artifactory for continuous monitoring of application dependencies. Detailed reports and dashboards are available through the use of XRay to enable teams to easily manage and remediate any detected vulnerabilities[19].

- **Veracode Software Composition Analysis** is another commercially available solution that focuses on scanning Java-based application dependencies to identify known security threats. In addition to its capability of identifying vulnerabilities, the tool also furnishes users with data concerning open-source components' licensing compliance and suggests measures for rectification. This tool's potential is emphasized through its ability to be integrated into the software development lifecycle, ensuring constant scrutiny and notifications of newly discovered problems[20].

4.2 Technologies used in dependency scanning tools

In order to detect potential security risks in software projects, dependency scanning tools employ a methodical process of scanning the source code and analyzing its dependencies.

Through parsing the code and constructing a dependency graph, relationships between components can be mapped and evaluated for any known vulnerabilities present. The scanner proceeds to individually analyze each dependency by determining its version number, provenance, and metadata.

Upon initiating the scanning process, the scanner will perform an exhaustive check against each and every dependency on a database known for its repository of vulnerabilities. The scanner vendor may maintain the database or obtain it from an external third-party like the National Vulnerability Database (NVD).

Based on various techniques such as string matching, regular expressions, and fuzzy matching, metadata information about each dependency is compared with entries in the vulnerability database. If a particular match is detected, then a detailed report highlighting specific potential risks and necessary remedial measures will be presented by the scanner. Incorporated within the report may be details about how severe a vulnerability is and what impact it has on software functionality or security.

To carry out this type of evaluation, dependency scanning tools employ numerous technologies; static analysis and dynamic analysis are potential options[21][22].

4.2.1 Static analysis

Analyzing code without running it is possible using static analysis, which is often used in software development. A key area where this technique comes into play is within dependency scanning tools, specifically when identifying the associations present among the various components that comprise a software system.

Dependencies exist on two levels: internally, meaning relations that link distinct modules within one program; or externally, denoting connections across various frameworks and libraries implemented by said program.

Through scrutinizing all aspects of a software system's coding infrastructure, static analysis enables dependency scanning tools to highlight these interdependencies, which is achieved by first parsing source code. This process entails a breakdown of the code into its various constituent parts, including functions, variables, and statements. Additionally, identification and tracing of dependencies between these segments are required through this endeavor.

The next phase of this analysis consists of constructing an abstract syntax tree (AST) through the assembly of parsed data to render an astute representation capturing both the structure and syntax of the code.

Finally, analyzing ASTs is executed using diverse approaches seeking aid in clarifying dependencies relating to individual software components. The identification of software system dependencies involves various techniques like data flow analysis, control flow analysis, and type inference.

Additionally, the dependency scanning tool produces a detailed report regarding the identified dependencies, including information about the involved components, the dependency types, and their strengths.

Static analysis is a crucial part of dependency scanning and is used by many popular dependency scanning tools in the software development industry. Nevertheless, it has its limitations since it may not recognize dependencies that arise at runtime or those that are indirectly expressed in code[23].

4.2.2 Dynamic analysis

To analyze how a program behaves during execution, developers often rely on dynamic analysis, a popular technique used in software development. Additionally, this approach can also be applied to dependency scanning tools that are tasked with identifying connections between different components within a software system.

Generally speaking, during dynamic analysis, one runs the target application while recording its specific behaviors and closely monitoring interactions between individual program sections. During program execution, the dependency scanning tool carefully observes its behavior and captures data on various runtime characteristics, including memory usage and execution time.

Upon completing the program execution, the dependency scanning tool uses different techniques, such as analyzing memory usage, tracing execution components, and identifying patterns in behavior, to scrutinize the collected data.

Finally, based on the data analysis results, it reports all identified dependencies.

Dynamic analysis, which involves identifying the components involved in a dependency, determining the type of dependency, and assessing the strength of the dependency, is a powerful technique for examining software systems. It is especially useful when capturing dependencies that only emerge at runtime. Nonetheless, conducting dynamic analysis can be resource-intensive and complicated compared to static analysis since the program needs to run while data on its behavior is collected[24].

4.3 Theoretical comparison of existing dependency scanning tools

Table 1, Table 2, Table 3, Table 4 and Table 5 provide a theoretical comparison of the nine most popular dependency scanning tools for Java-based applications, including: OWASP Dependency Check[25], Sonatype Nexus Lifecycle[26], Snyk[27], Black Duck[28], DependencyTrack[29], Mend SCA (formerly Whitesource)[30], Aqua Security[18], XRay by JFrog[31], and Veracode[32]. The comparison will be made based on 12 different criteria:

- Vulnerability database – the database of vulnerabilities that is being used by the dependency scanning tool to identify vulnerabilities.
- Supported programming languages – applications written in a programming language that can be scanned by the dependency scanning tool.
- Analysis scope – which part of the application is scanned by the dependency scanning tool.
- Analysis type – what type of analysis is used during the scan (static, dynamic).
- Integration with IDEs – which IDEs can the dependency scanning tool be integrated with.
- Integration with CI/CD – which CI/CD systems can the dependency scanning tool be integrated with.
- Integration with repositories – which repositories can the dependency scanning tool be integrated with.
- Scanning capabilities – what is being scanned by the dependency scanning tool and what is included in the final report.
- License – under which license is the dependency scanning tool operating.
- Deployment – where can the dependency scanning tool be installed (locally, in the cloud).
- Reporting – in what format is the final report presented.

- Pricing – how much does the dependency scanning tool cost.

Table 1 provides the details about OWASP Dependency Check and Snyk.

Table 1. OWASP Dependency Check and Snyk details

Parameter	OWASP Dependency Check	Snyk
Vulnerability database	OWASP Top 10, NVD, Retire.js, VulnDB, Sonatype OSS Index	Snyk Vulnerability Database
Supported programming languages	Java, .NET, Node.js, Ruby, Python, PHP, Golang	Java, .NET, Node.js, Ruby, Python, Golang, Scala, Kotlin
Analysis scope	Application dependencies	Application dependencies and container images
Analysis type	Static analysis	Static analysis
Integration with IDEs	Eclipse, IntelliJ IDEA, Visual Studio, Visual Studio Code, Jenkins, Azure DevOps, TeamCity	Visual Studio Code, IntelliJ IDEA, WebStorm, Atom, Eclipse, CLI
Integration with CI/CD	Jenkins, Azure DevOps, TeamCity, CircleCI, GitLab CI/CD, Travis CI, Bamboo, CodeShip, Codeship Pro, Drone, GitHub Actions	Jenkins, Azure DevOps, CircleCI, GitLab CI/CD, Travis CI, Bitbucket Pipelines, Codefresh, CodeShip, Codeship Pro, Drone, GitHub Actions, AWS CodeBuild

Parameter	**OWASP Dependency Check**	**Snyk**
Integration with repositories	GitHub, GitLab, Bitbucket, Artifactory, Nexus, npm, PyPI, NuGet	GitHub, GitLab, Bitbucket, JFrog Artifactory, AWS Elastic Container Registry, AWS Elastic Kubernetes Service
Scanning capabilities	Finds known vulnerabilities in dependencies	Finds known vulnerabilities in dependencies and containers, identifies license violations, detects security misconfigurations
License	Open source (Apache 2.0)	Proprietary
Deployment	On-premises, Docker, cloud	On-premises, cloud, SaaS
Reporting	HTML, JSON, XML, CSV, SARIF	HTML, JSON, XML, CLI
Pricing	Free	Free for open source projects, paid plans for enterprise use

Table 2 provides the details about Sonatype Nexus Lifecycle and Black Duck.

Table 2. Sonatype Nexus Lifecycle and Black Duck Hub details

Parameter	**Sonatype Nexus Lifecycle**	**Black Duck**
Vulnerability database	Sonatype OSS Index, NVD, OWASP Top 10, Snyk, VulnDB, Retire.js, JFrog Xray	Black Duck Knowledge Base
Supported programming languages	Java, .NET, Node.js, Ruby, Python, Go, PHP, Swift, Kotlin, Scala, JavaScript, Rust, C/C++	Java, .NET, Node.js, Ruby, Python, Go, PHP, Swift, Kotlin, Scala, JavaScript, Rust, C/C++
Analysis scope	Application dependencies, containers, and software bill of materials (SBOM)	Application dependencies, containers, and software bill of materials (SBOM)
Analysis type	Combination of static and dynamic analysis	Combination of static and dynamic analysis
Integration with IDEs	IntelliJ IDEA, Eclipse, Visual Studio, Visual Studio Code	Eclipse, IntelliJ IDEA, Visual Studio, Visual Studio Code, Atom, Sublime Text, PyCharm
Integration with CI/CD	Jenkins, Azure DevOps, Bamboo, TeamCity, CircleCI, GitLab CI/CD, Travis CI, Codefresh	Jenkins, Azure DevOps, Bamboo, TeamCity, CircleCI, GitLab CI/CD, Travis CI, Codefresh, CodeShip, AWS CodePipeline
Integration with repositories	Nexus Repository Manager, JFrog Artifactory, GitHub,	Nexus Repository Manager, JFrog Artifactory, GitHub,

Parameter	Sonatype Nexus Lifecycle	Black Duck
	GitLab, Bitbucket, npm, PyPI, Maven Central, NuGet	GitLab, Bitbucket, Docker Hub, AWS Elastic Container Registry, AWS CodeArtifact
Scanning capabilities	Finds known vulnerabilities in dependencies and provides policy enforcement capabilities based on custom rules	Finds known vulnerabilities in dependencies, identifies license compliance issues, and provides policy enforcement capabilities based on custom rules
License	Proprietary	Proprietary
Deployment	On-premises, cloud, SaaS	On-premises, cloud, SaaS
Reporting	HTML, JSON, XML, CSV, SARIF	HTML, PDF, JSON, CLI
Pricing	Paid plans for enterprise use	Paid plans for enterprise use

Table 3 provides the details about DependencyTrack and Mend SCA (formerly Whitesource).

Table 3. DependencyTrack and Mend SCA details

Parameter	DependencyTrack	Mend SCA
Vulnerability database	NVD, VulnDB, Retire.js, ODC, ODC Advanced, and WhiteSource proprietary database	WhiteSource proprietary database
Supported programming languages	Java, .NET, Node.js, Python, Ruby, Go, PHP, Scala, Swift, Kotlin, Objective-C, JavaScript	Java, .NET, Node.js, Python, Ruby, Go, PHP, Scala, Swift, Kotlin, Objective-C, JavaScript, Perl, Lua, Shell
Analysis scope	Application dependencies, containers, and software bill of materials (SBOM)	Application dependencies, containers, and software bill of materials (SBOM)
Analysis type	Combination of static and dynamic analysis	Combination of static and dynamic analysis
Integration with IDEs	Eclipse, IntelliJ IDEA, Visual Studio Code, Visual Studio	Eclipse, IntelliJ IDEA, Visual Studio Code, Visual Studio, Atom, Sublime Text, PyCharm
Integration with CI/CD	Jenkins, Bamboo, CircleCI, GitLab CI/CD, Travis CI	Jenkins, Bamboo, TeamCity, CircleCI, GitLab CI/CD, Travis CI, Azure DevOps, Bitbucket Pipelines, GitHub Actions

Parameter	**DependencyTrack**	**Mend SCA**
Integration with repositories	Nexus Repository Manager, JFrog Artifactory, GitLab, GitHub, Bitbucket, Docker Hub	Nexus Repository Manager, JFrog Artifactory, GitLab, GitHub, Bitbucket, Docker Hub, npm, PyPI, Maven Central, NuGet
Scanning capabilities	Finds known vulnerabilities in dependencies and provides policy enforcement capabilities based on custom rules	Finds known vulnerabilities in dependencies, identifies license compliance issues, and provides policy enforcement capabilities based on custom rules
License	Open source	Proprietary
Deployment	On-premises, cloud, SaaS	On-premises, cloud, SaaS
Reporting	HTML, PDF, JSON, CSV, XML	HTML, JSON, XML, CLI
Pricing	Free and open source	Paid plans for enterprise use

Table 4 provides the details about Aqua Security and XRay by JFrog.

Table 4. Aqua Security and XRay by JFrog details

Parameter	**Aqua Security**	**XRay by JFrog**
Vulnerability database	Aqua proprietary database, NVD, CVSS	NVD, VulnDB, JFrog proprietary database
Supported programming languages	Java, .NET, Node.js, Python, Ruby, Go, PHP, Scala, Swift, Kotlin, Objective-C, JavaScript, Rust	Java, .NET, Node.js, Python, Ruby, Go, PHP, Scala, Swift, Kotlin, Objective-C, JavaScript
Analysis scope	Container images, Kubernetes, cloud-native environments, and software bill of materials (SBOM)	Application dependencies, containers, and software bill of materials (SBOM)
Analysis type	Dynamic analysis	Static analysis
Integration with IDEs	None	IntelliJ IDEA, Visual Studio Code, Visual Studio
Integration with CI/CD	Jenkins, GitLab CI/CD, Travis CI, Azure DevOps, CircleCI, Tekton, Spinnaker, Codefresh	Jenkins, Bamboo, TeamCity, CircleCI, GitLab CI/CD, Travis CI, Azure DevOps, Bitbucket Pipelines, GitHub Actions
Integration with repositories	Docker Hub, Amazon ECR, Google GCR, JFrog Artifactory	Nexus Repository Manager, JFrog Artifactory, GitLab, GitHub, Bitbucket, Docker

Parameter	Aqua Security	XRay by JFrog
		Hub, npm, PyPI, Maven Central, NuGet
Scanning capabilities	Finds known vulnerabilities in container images and cloud-native environments, provides policy enforcement capabilities based on custom rules	Finds known vulnerabilities in dependencies, identifies license compliance issues, and provides policy enforcement capabilities based on custom rules
License	Proprietary	Proprietary
Deployment	On-premises, cloud, SaaS	On-premises, cloud, SaaS
Reporting	HTML, PDF, JSON, CSV	HTML, JSON, XML, CLI
Pricing	Paid plans for enterprise use	Paid plans for enterprise use

Table 5 provides the details about Veracode.

Table 5. Veracode details

Parameter	**Veracode**
Vulnerability database	Proprietary database, NVD
Supported programming languages	Java, .NET, Node.js, Python, Ruby, Go, PHP, Scala, Swift, Kotlin, Objective-C, JavaScript, TypeScript, COBOL, ABAP
Analysis scope	Dependencies and software bill of materials (SBOM)
Analysis type	Static analysis
Integration with IDEs	Eclipse, IntelliJ IDEA, Visual Studio, Visual Studio Code
Integration with CI/CD	Jenkins, Azure DevOps, TeamCity, CircleCI, GitLab CI/CD, Bamboo
Integration with repositories	JFrog Artifactory, Nexus Repository Manager, GitLab, GitHub
Scanning capabilities	Identifies known vulnerabilities and offers remediation guidance, identifies license compliance issues,

Parameter	Veracode
	provides policy enforcement capabilities based on custom rules
License	Proprietary
Deployment	On-premises, SaaS
Reporting	HTML, PDF, CSV, JSON
Pricing	Paid plans for enterprise use

5 Dependency scanning tools usage and instructions

The following chapter focuses on the practical part of the work. It will provide readers with instructions on how to properly use dependency scanning tools, demonstrate their usage on a deliberately insecure application and provide an example of dependency scanning tool's usage in the software development company.

5.1 Conduction of usage instructions for dependency scanning tools

The following chapter will provide two types of instructions on how to properly use dependency scanning tools: general and application specific.

5.1.1 General instructions

Choose a dependency scanning tool

There are several dependency scanning tools available on the market that can help you identify and fix vulnerabilities in your application. Refer to the comparison of existing dependency scanning tools to choose a tool that meets your needs and budget.

Install the dependency scanning tool

Follow the installation instructions provided by the tool vendor to integrate the scanner into your application.

Configure the dependency scanning tool

Once the scanner is installed, you need to configure it to scan your application. The configuration steps may vary depending on the scanner you choose.

Run the scan

Once the scanner is configured, you can run a scan of your application. The scanner will crawl your application and identify any vulnerabilities that exist in your application's code or dependencies.

Review the scan report

After the scan is complete, the scanner will generate a report that provides details about the vulnerabilities it has identified. Take the time to review the report carefully to understand the nature and severity of the vulnerabilities.

Resolve the vulnerabilities

Once you have reviewed the report, you can take steps to resolve the vulnerabilities. Depending on the severity of the vulnerability, the recommended actions may vary. In general, the actions you can take include:

- Updating the affected dependencies to a version that is not vulnerable
- Patching the vulnerability by applying a fix or workaround
- Removing the dependency altogether if it is not needed or cannot be patched
- Marking the vulnerability as not applicable if vulnerable functionality of the dependency is not used

To resolve the vulnerabilities, you can follow the specific instructions provided in the report. Once you have made the necessary changes, you can run the scan again to verify that the vulnerabilities have been resolved.

Monitor for new vulnerabilities

It is important to monitor your Java web application for new vulnerabilities on an ongoing basis. Different dependency scanning tools can help you stay up-to-date by providing alerts when new vulnerabilities are discovered in your application's dependencies. You can configure the scanner to send alerts to your email or another preferred communication channel or integrate it with your CI/CD pipeline to automate vulnerability scanning and resolution.

5.1.2 Specific instructions for OWASP Dependency Check

Option 1 – using a plugin

Installation

Step 1: Add the OWASP Dependency Check plugin to your project's build tool. For example, if you are using Maven, add the following plugin to your pom.xml file:

```
<plugin>
    <groupId>org.owasp</groupId>
      <artifactId>dependency-check-maven</artifactId>
    <version><version></version>
</plugin>
```

Figure 1. OWASP Dependency Check Maven plugin.

Replace <version> with the version number of the OWASP Dependency Check plugin that you want to use.

Scanning the application

Step 1: Run the OWASP Dependency Check plugin by executing the following command in your project's directory:

```
mvn org.owasp:dependency-check-maven:check
```

Figure 2. OWASP Dependency Check plugin scan execution command

Wait for the OWASP Dependency Check plugin to finish scanning your application's dependencies. The plugin will generate a report containing any identified vulnerabilities and dependencies that are out of date.

Refer to the General instructions paragraph starting with “Reviewing the report“ for further actions.

Option 2 – using the tool

Installation

Step 1: Download OWASP Dependency Check from the official website and extract the files to a directory of your choice.

Step 2: Open a terminal or command prompt and navigate to the directory where you extracted the OWASP Dependency Check files.

Step 3: Set up your Java environment by ensuring that Java is installed on your system and that the JAVA_HOME environment variable is set correctly.

Scanning the application

Step 1: Run the OWASP Dependency Check tool by typing the following command in the terminal or command prompt:

```
java -jar dependency-check-<version>-release.jar -s
<path_to_your_project_directory>
```

Figure 3. OWASP Dependency Check tool scan execution command

Replace <version> with the version number of the OWASP Dependency Check tool that you downloaded, and <path_to_your_project_directory> with the path to the directory containing your Java application.

Wait for the OWASP Dependency Check tool to finish scanning your application's dependencies. The tool will generate a report containing any identified vulnerabilities and dependencies that are out of date.

Refer to the General instructions paragraph starting with “Reviewing the report“ for further actions.

5.1.3 Specific instructions for Snyk

Installation

Step 1: Install Node.js and npm on your machine. You can download the latest version from the official Node.js website.

Step 2: Open the terminal or command prompt and run the following command to install Snyk:

```
npm install -g snyk
```

Figure 4. Snyk installation command

Scanning the application

Step 1: Create a Snyk account if you don't already have one. You can create an account for free on the Snyk website.

Step 2: Open the terminal or command prompt and run the following command to authenticate in Snyk:

```
snyk auth
```

Figure 5. Snyk authentication command

Step 2: After authenticating into the tool, run the following command in your terminal to test your application for vulnerabilities:

```
mvn snyk:test
```

Figure 6. Snyk scan execution command

This command will scan your application and identify any vulnerabilities that exist.

After running the scan execution command, Snyk will generate a report that identifies any vulnerabilities that exist in your application's dependencies.

Refer to the General instructions paragraph starting with “Reviewing the report“ for further actions.

5.1.4 Specific instructions for DependencyTrack

Installation

Step 1: Install Docker and Docker Compose on your machine. You can find the installation instructions for your platform on the Docker website and the Docker Compose website.

Step 2: Create a directory that will contain Docker Compose file for running DependencyTrack and navigate to it in your terminal or command prompt.

Step 3: Download the official Docker Compose file for running DependencyTrack from the official website:

```
curl -O https://dependencytrack.org/docker-compose.yml
```

Figure 7. Command for downloading DependencyTrack Docker Compose file

Step 4 (optional): Open the docker-compose.yml file in a text editor and modify the db_password field under the services > dependency-track > environment section to set a secure password for the database.

Step 5: Start the DependencyTrack server and the database by running the following command in the project directory:

```
docker-compose up -d
```

Figure 8. Command for running DependencyTrack Docker Compose file

Integration with the application

Step 1: Add the CycloneDX plugin to your application. If you are using Maven as a build tool, add the following plugin to your pom.xml file:

```
<plugin>
	<groupId>org.cyclonedx</groupId>
		<artifactId>cyclonedx-maven-plugin</artifactId>
	<version><version></version>
</plugin>
```

Figure 9. CycloneDX Maven plugin

Replace <version> with the version number of the CycloneDX plugin that you want to use.

If you are using Gradle as a build tool, add the following dependency to your build.gradle file:

```
id "org.cyclonedx.bom" version <version>
```

Figure 10. CycloneDX Gradle dependency

Replace <version> with the version number of the CycloneDX plugin that you want to use.

Step 2: Build your Java web application and generate a CycloneDX Bill of Materials (BOM) file by running the following commands.
If Maven is used as a build tool:

```
mvn org.cyclonedx:cyclonedx-maven-plugin:makeBom
```

Figure 11. CycloneDX Maven plugin execution command

If Gradle is used as a build tool:

```
./gradlew cyclonedxBom
```

Figure 12. CycloneDX Gradle dependency execution command

Step 3: Upload the CycloneDX BOM file to DependencyTrack by logging into the DependencyTrack web interface at http://localhost:8080 and navigating to the Projects section.

Step 4: Click on the Create Project button and enter the required information for your application.

Step 5: Open the project and navigate to the Components tab. click on the Upload BOM button and select the BOM file that was generated in Step 2.

Step 6: After refreshing the page, DependencyTrack will analyze the BOM file and generate a report of any vulnerabilities found in your application's dependencies. You can view the report under the Vulnerabilities tab.

Refer to the General instructions paragraph starting with “Reviewing the report“ for further actions.

5.1.5 Specific instructions for XRay by JFrog

The instructions will cover installing and using the XRay by JFrog plugin in IntelliJ Idea. XRay by JFrog also offers other integration options such as VS Code, Android Studio, Docker, JFrog CLI, GitHub Actions, etc. You can find detailed installation instructions for each supported product on the JFrog website.

Installation

Step 1: Go to the XRay by JFrog homepage at jfrog.com/xray and create an account to start a free trial.

Step 2: To install the XRay plugin for IntelliJ IDEA, you'll need to open the IDE, go to File > Settings > Plugins, search for "XRay", and click "Install".

Step 3: Once you've installed the XRay plugin, you'll need to configure it to connect to your JFrog instance. This will involve entering your JFrog credentials and specifying the URL for your JFrog instance.

Scanning the application

Step 1: To run a scan using XRay, you can simply click the "Run Scan" button in the XRay panel of IntelliJ IDEA. This will initiate a scan of your project's dependencies.

Once you've done the XRay scanning, you'll be able to view XRay results directly in your IDE.

Refer to the General instructions paragraph starting with “Reviewing the report“ for further actions.

5.2 Practical usage of existing dependency scanning tools

Practical usage of existing dependency scanning tools involved testing four free tools: OWASP Dependency Check, Snyk, DependencyTrack, and XRay by JFrog on the WebGoat project[33].

WebGoat is a deliberately insecure web application created for educational purposes, and as such, it presents a useful target for testing dependency scanning tools. The application includes a variety of vulnerability examples and challenges that simulate real-world web application security flaws, including those that may arise from dependencies. As a Java-based application, WebGoat relies on numerous dependencies, such as libraries and frameworks, to function. These dependencies can introduce vulnerabilities of their own, such as outdated versions with known security issues or code injection vulnerabilities.

Testing dependency scanning tools on WebGoat can provide valuable insights into the effectiveness and accuracy of these tools. By intentionally introducing vulnerabilities into the application and then scanning for them, it can be assessed how well the scanners are able to detect and report on these issues.

In addition, testing dependency scanning tools on WebGoat can also help identify potential areas for improvement in the scanners themselves. For example, if a scanner fails to detect a known vulnerability in one of WebGoat's dependencies, this may indicate a need for updates or enhancements to the scanner's vulnerability database or detection algorithms.

Overall, WebGoat is an excellent target for testing dependency scanning tools, as it presents a realistic and varied set of vulnerabilities that can be used to assess the effectiveness and accuracy of these tools. By using WebGoat in this way, users can gain a better understanding of the strengths and weaknesses of different dependency scanning tools and make more informed decisions about which tools to use in their own web application security testing efforts.

5.2.1 OWASP Dependency Check

Running OWASP Dependency Check on WebGoat project generated the report in HTML format (see Appendix 2).

Analysis of the report led to the following conclusions regarding Maven dependencies:

- The application contains 138 dependencies.
- The number of vulnerable dependencies is 16, 8 being of critical severity, 7 being of high severity, 1 being of medium severity
- The total number of vulnerabilities is 75.
- The list of vulnerable components with respective severity level is shown in Table 6.

Table 6. OWASP Dependency Check list of vulnerable components with respective severity level

Component	**Severity level**
hsqldb-2.5.2.jar	CRITICAL
snakeyaml-1.30.jar	CRITICAL
spring-web-5.3.21.jar	CRITICAL
spring-security-web-5.7.2.jar	CRITICAL
spring-security-crypto-5.7.2.jar	CRITICAL
spring-security-core-5.7.2.jar	CRITICAL
spring-security-config-5.7.2.jar	CRITICAL
xstream-1.4.5.jar	CRITICAL
spring-webmvc-5.3.21.jar	HIGH
spring-expression-5.3.21.jar	HIGH

Component	**Severity level**
spring-core-5.3.21.jar	HIGH
xnio-api-3.8.7.Final.jar	HIGH
undertow-servlet-2.2.18.Final.jar	HIGH
undertow-core-2.2.18.Final.jar	HIGH
jackson-databind-2.13.3.jar	HIGH
guava-31.1-jre.jar	MEDIUM

5.2.2 Snyk

Running Snyk on WebGoat project generated the report in the CLI (see Appendix 3).

Analysis of the report led to the following conclusions regarding Maven dependencies:

- The application contains 136 dependencies.
- The number of vulnerable dependencies is 10, 2 being of critical severity, 4 being of high severity, 4 being of medium severity.
- The total number of vulnerabilities is 55.
- The list of vulnerable components with respective severity level is shown in Table 7.

Table 7. Snyk list of vulnerable components with respective severity level

Component	**Severity level**
spring-webmvc-5.3.21.jar	CRITICAL
xstream-1.4.5.jar	CRITICAL
hsqldb-2.5.2.jar	HIGH
snakeyaml-1.30.jar	HIGH
spring-security-web-5.7.2.jar	HIGH
undertow-core-2.2.18.Final.jar	HIGH
spring-expression-5.3.21.jar	MEDIUM
spring-security-config-5.7.2.jar	MEDIUM
xnio-api-3.8.7.Final.jar	MEDIUM
jackson-databind-2.13.3.jar	MEDIUM

5.2.3 DependencyTrack

Running DependencyTrack on WebGoat project generated the report in the server run using docker (see Appendix 4).

Analysis of the report led to the following conclusions:

- The application contains 142 dependencies.
- The number of vulnerable dependencies is 12, 5 being of critical severity, 4 being of high severity, 3 being of medium severity.
- The total number of vulnerabilities is 43.
- The list of vulnerable components with respective severity level is shown in Table 8.

Table 8. DependencyTrack list of vulnerable components with respective severity level

Component	Severity level
hsqldb-2.5.2.jar	CRITICAL
snakeyaml-1.30.jar	CRITICAL
spring-web-5.3.21.jar	CRITICAL
spring-security-web-5.7.2.jar	CRITICAL
xstream-1.4.5.jar	CRITICAL
spring-webmvc-5.3.21.jar	HIGH
xnio-api-3.8.7.Final.jar	HIGH
undertow-core-2.2.18.Final.jar	HIGH
jackson-databind-2.13.3.jar	HIGH
guava-31.1-jre.jar	MEDIUM

Component	Severity level
spring-expression-5.3.21.jar	MEDIUM
spring-security-crypto-5.7.2.jar	MEDIUM

5.2.4 XRay by JFrog

Running XRay by JFrog on WebGoat project generated the scan report in the corresponding plugin (see Appendix 5). Analysis of the report led to the following conclusions:

- The number of dependencies is not known.
- The number of vulnerable dependencies is 12, 5 being of critical severity, 7 being of high severity, 2 being of medium severity.
- The total number of vulnerabilities is 60.
- The list of vulnerable components with respective severity level is shown in Table 9.

Table 9. XRay by JFrog list of vulnerable components with respective severity level

Component	Severity level
hsqldb-2.5.2.jar	CRITICAL
snakeyaml-1.30.jar	CRITICAL
spring-webmvc-5.3.21.jar	HIGH
spring-web-5.3.21.jar	CRITICAL
spring-core-5.3.21.jar	HIGH
spring-security-core-5.7.2.jar	CRITICAL
undertow-core-2.2.18.Final.jar	HIGH

Component	**Severity level**
xstream-1.4.5.jar	CRITICAL
jackson-databind-2.13.3.jar	HIGH
jetty-server:9.4.48v0220622	MEDIUM
thymeleaf-extras-springsecurity5:3.0.4.RELEASE	MEDIUM
wildfly-common:1.5.4.Final	HIGH

5.3 Example of practical usage of dependency scanning tool in software development company

The use of the dependency scanning tool will be demonstrated on the infrastructure of Bally’s Interactive.

Bally's Interactive is a gaming company that develops and maintains online casino games and sports betting applications. Managing used software dependencies is critical to their development process to ensure software quality, performance, and compliance.

The methodology used by Bally's Interactive to manage software dependencies involves using Nexus Lifecycle as a dependency scanning tool at the build stage of each application to avoid deploying applications containing vulnerable dependencies even in test environments. Nexus Lifecycle is integrated into the build stage of GoCD pipelines, and policies are defined for acceptable components, versions, and licenses.

Nexus Lifecycle is used in the software development process by scanning all software dependencies used in the applications. The results of the scan are then reviewed by the development team, and any vulnerabilities or policy violations are addressed.

From my experience of as part of the development team at Bally’s Interactive, it is safe to state that approximately 1 out of 10 builds fail because of Nexus reports containing 1 to 10 reported vulnerabilities in the code, which, if ignored, might lead to highly vulnerable applications.

An example scenario indicating the usage of Nexus Lifecycle is described below.

The build stage of the pipeline has failed (see Appendix 6, Figure 23).

The reason for the failed build is Nexus finding two components with critical vulnerabilities in applications' dependencies (see Appendix 6, Figure 24).

Opening the link for the report provided by Nexus opens up the page, where we can see that the components with critical vulnerabilities are transitive dependencies com.fasterxml.jackson.core:jackson-core:2.14.1 and org.yaml:snakeyaml:1.33 (see Appendix 6, Figure 25).

When opening an overview of the com.fasterxml.jackson.core:jackson-core:2.14.1 component, we can see that it is a transitive dependency of the component org.springframework.boot:spring-boot-starter-webflux:3.0.2 and that the recommended way of mitigating the vulnerability is upgrading the dependency to version 2.15.0-rc1. (see Appendix 6, Figure 26).

Apart from that, we can also navigate to the "Policy violations" tab, which contains information about policies that have been violated by the component (see Appendix 6, Figure 27).

On the next tab "Security" we can see information about security policies that have been violated and the exact vulnerability that the dependency contains (see Appendix 6, Figure 28).

When opening the details of the vulnerability, we can see a lot of important information, such as the vulnerability score, explanation, vulnerable files and functions, recommendations, versions affected, etc. (See Appendix 6, Figure 29, Figure 30, Figure 31).

The next tab "Legal" contains information about the licenses that have been detected in the component and also outlines legal policy violations, if such are present (see Appendix 6, Figure 32).

The last tab "Audit log" includes actions such as when a scan was initiated, when policies were defined or updated, when vulnerabilities were identified, and when remediation actions were taken. In our case, the remediation action that has been taken is marking the vulnerability as “Not applicable” since vulnerable functionality is not used in the application (see Appendix 6, Figure 33).

The use of Nexus Lifecycle has led to a significant reduction in the number of vulnerabilities in the software dependencies used in the applications and a greater awareness among developers about the existence of such vulnerabilities. Additionally, the development team has been able to identify and address policy violations early in the development process, leading to better compliance with organizational policies. This has resulted in improved software quality and performance.

6 Analysis of practical part

Demonstration of theoretical knowledge is a crucial element in academic writing, and it is achieved through the practical segment of the thesis.

Consequently, the practical part of the thesis concentrates on providing an illustration via various dependency scanning tools that are useful for software development. In such scenarios, step-by-step instructions simplify usage procedures and demonstrate how different dependency scanning tools help in feasible software development.

Furthermore, a case study of a well-known software development company provides additional insight into scanner applications. Detailing the experiences of a software development company in using dependency scanning tools, the second section of this article outlines specific challenges relating to dependency management that were faced by said organization. With the efficient scanning capabilities provided by these powerful tools, however, those hurdles were successfully surmounted, thereby illustrating both the current potential problems facing developers today and how powerful detection solutions could prove quite useful in addressing them specifically.

Completing our understanding of these vital utilities appears later in this piece with step-by-step directions on properly installing and leveraging dependency scanning tools. The practical aspect of this thesis is crucial, as it equips the reader with the necessary skills to apply dependency scanning tools in their own projects. The application of different dependency scanning tools in software development is well demonstrated, making the practical section an excellent resource for readers.

Indeed, the instructions are concise and straightforward, which allows readers to easily comprehend and deploy these sophisticated tools in their respective projects. Moreover, through the example of a software development company, this section emphasizes the pragmatic benefits associated with utilizing dependency scanning tools.

7 Future research

Possible future research directions for the topic might include:

- Investigation of new techniques and tools for dependency management. As the landscape of software development evolves, new tools and techniques for managing dependencies may emerge. Future research could explore and evaluate these new approaches and assess their effectiveness in detecting and mitigating security vulnerabilities.

- Analysis of the effectiveness of different vulnerability detection methods. Dependency scanning tools use various methods for detecting security vulnerabilities, such as static analysis and dynamic analysis. Future research could compare the effectiveness of these different methods in detecting and mitigating security vulnerabilities in Java-based web applications.

- Examination of the impact of vulnerability mitigation on application performance. Updating dependencies and mitigating security vulnerabilities can sometimes have an impact on the performance of web applications. Future research could investigate the relationship between vulnerability mitigation and application performance and explore ways to optimize this balance.

- Study of the human factors in dependency management. While dependency scanning tools are valuable tools for detecting and mitigating security vulnerabilities, human factors also play a critical role in effective dependency management. Future research could examine the role of developers, project managers, and other stakeholders in ensuring the security of web applications through effective dependency management practices.

- Investigation of the relationship between open-source software and security vulnerabilities. Open-source software plays a significant role in software development and is often used in web applications. However, the use of open-source code can also introduce security vulnerabilities. Future research could explore the relationship between open-source software and security vulnerabilities and investigate ways to mitigate these risks in Java-based web applications.

8 Summary

The thesis aimed to address the issue of security vulnerabilities in Java-based web applications caused by used dependencies. The thesis proposed the use of dependency scanning tools as a solution to identify and manage vulnerabilities associated with used dependencies in web applications.

The thesis provided a comprehensive analysis of the current state-of-the-art tools and techniques for managing used dependencies in Java-based web applications. It highlighted the advantages of automated dependency scanning tools.

The thesis also presented a detailed evaluation of several popular dependency scanning tools, including a detailed comparison. The evaluation criteria included different important parameters such as supported programming languages, used vulnerability database, analysis scope, pricing etc.

In conclusion, the thesis provided valuable insights into the issue of managing used dependencies in Java-based web applications and proposed a practical solution using dependency scanning tools. The evaluation of different scanners can guide developers in selecting the most appropriate tool for their specific needs. The recommendations provided in the thesis can help improve the security of Java-based web applications and prevent vulnerabilities caused by used dependencies.

References

[1] "What are software dependencies." Sonatype. https://www.sonatype.com/launchpad/what-are-software-dependencies (accessed May 14, 2023).

[2] "What are direct and indirect dependencies?" Snyk Support. https://support.snyk.io/hc/en-us/articles/360000905138-What-are-direct-and-indirect-dependencies- (accessed May 13, 2023).

[3] S. Koussa. "13 tools for checking the security risk of open-source dependencies." TechBeacon. https://techbeacon.com/app-dev-testing/13-tools-checking-security-risk-open-source-dependencies (accessed May 14, 2023).

[4] "Free for Open Source Application Security Tools," OWASP, https://owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools (accessed May 14, 2023).

[5] J. Fruhlinger, "Equifax Data Breach FAQ: What happened, who was affected, what was the impact?," CSO Online, https://www.csoonline.com/article/3444488/equifax-data-breach-faq-what-happened-who-was-affected-what-was-the-impact.html (accessed May 14, 2023).

[6] "2019 capital one cyber incident: What happened," Capital One, https://www.capitalone.com/digital/facts2019/ (accessed May 14, 2023).

[7] D. Lukic. "Target Data Breach: How was target hacked?," IDStrong, https://www.idstrong.com/sentinel/that-one-time-target-lost-everything/ (accessed May 14, 2023).

[8] C. Jones. "Warnings (& lessons) of the 2013 Target Data Breach." Red River. https://redriver.com/security/target-data-breach (accessed May 14, 2023).

[9] J. Fruhlinger. "Marriott Data Breach FAQ: How did it happen and what was the impact?" CSO Online. https://www.csoonline.com/article/3441220/marriott-data-breach-faq-how-did-it-happen-and-what-was-the-impact.html (accessed May 14, 2023).

[10] S. Koussa. "13 tools for checking the security risk of open-source dependencies." TechBeacon. https://techbeacon.com/app-dev-testing/13-tools-checking-security-risk-open-source-dependencies (accessed May 14, 2023).

[11] "What was Wannacry?: Wannacry ransomware," Malwarebytes, https://www.malware bytes.com/wannacry (accessed May 14, 2023).

[12] "OWASP Dependency-Check," OWASP, https://owasp.org/www-project-dependency-check/ (accessed May 14, 2023).

[13] "Sonatype lifecycle - Application Security Testing & Management," Sonatype, https://www.sonatype.com/products/open-source-security-dependency-management (accessed May 14, 2023).

[14] "Open Source Security Management: SCA Tool," Snyk, https://snyk.io/product/open-source-security-management/ (accessed May 14, 2023).

[15] "Black Duck Software Composition Analysis (SCA)," Synopsys, https://www.synopsys.com/software-integrity/security-testing/software-composition-analysis.html (accessed May 14, 2023).

[16] Track: Software bill of materials (SBOM) analysis," Dependency, https://dependencytrack.org/ (accessed May 14, 2023).

[17] "MEND SCA: Open Source Software Management made simple," Mend, https://www.mend.io/sca/ (accessed May 14, 2023).

[18] "Container vulnerability scanning for cloud native applications," Aqua, https://www.aquasec.com/products/container-vulnerability-scanning/ (accessed May 14, 2023).

[19] "Software composition analysis tool - jfrog security," JFrog, https://jfrog.com/xray/ (accessed May 14, 2023).

[20] "Software composition analysis (SCA)," Veracode, https://www.veracode.com/products/software-composition-analysis (accessed May 14, 2023).

[21] A. Sharma, "Software composition analysis explained, and how it identifies open-source software risks," CSO Online, https://www.csoonline.com/article/3640808/software-composition-analysis-explained-and-how-it-identifies-open-source-software-risks.html (accessed May 14, 2023).

[22] "Component analysis," OWASP, https://owasp.org/www-community/Component_Analysis (accessed May 14, 2023).

[23] "What is SAST and how does static code analysis work?," Synopsys, https://www.synopsys.com/glossary/what-is-sast.html (accessed May 14, 2023).

[24] "What is Dynamic Application Security Testing (DAST): Micro focus," Microfocus, https://www.microfocus.com/en-us/what-is/dast (accessed May 14, 2023).

[25] Jeremylong, "OWASP Dependency Check repository," GitHub, https://github.com/jeremylong/DependencyCheck (accessed May 14, 2023).

[26] "Sonatype IQ Server," Sonatype Documentation, https://help.sonatype.com/iqserver (accessed May 14, 2023).

[27] "User docs," User Docs - Snyk User Docs, https://docs.snyk.io/ (accessed May 14, 2023).

[28] "User Guide BlackDuck," Synopsys, https://testing.blackduck.synopsys.com/doc/pdfs/user_guide.pdf (accessed May 13, 2023).

[29] S. Springett, "Dependency-Track Introduction," dependency track, https://docs.dependencytrack.org/ (accessed May 14, 2023).

[30] "MEND documentation," Mend Documentation, https://docs.mend.io/ (accessed May 14, 2023).

[31] "JFrog Xray," JFrog Help Center, https://jfrog.com/help/r/get-started-with-the-jfrog-platform/jfrog-xray (accessed May 14, 2023).

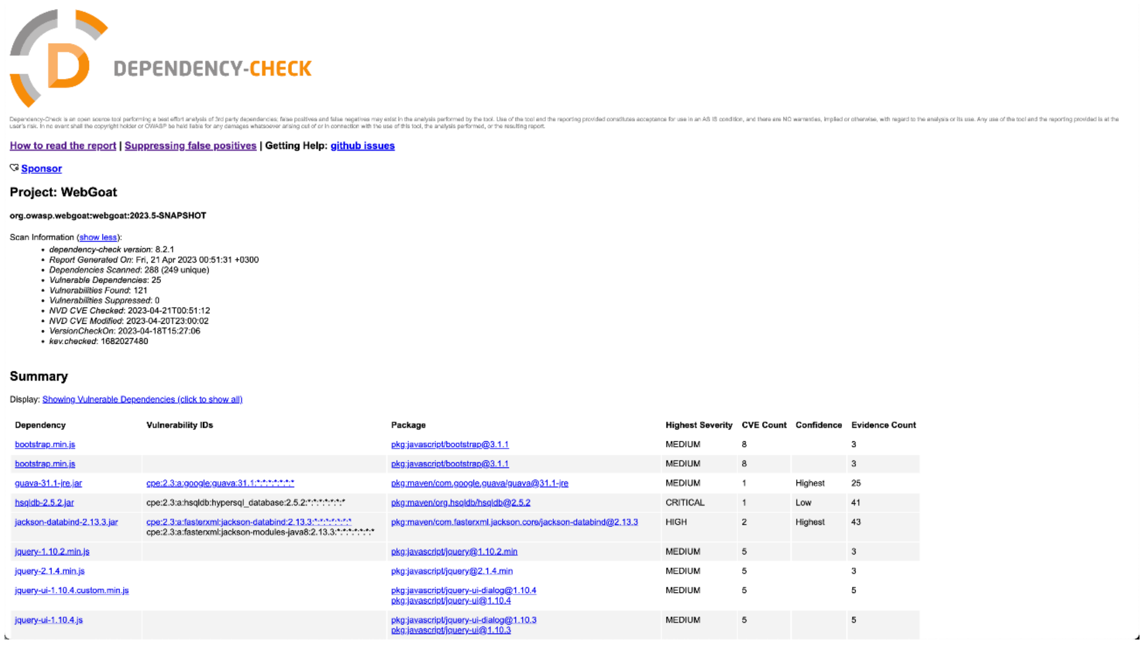
[32] "Veracode Docs," Veracode Docs RSS, https://docs.veracode.com/ (accessed May 14, 2023).

[33] WebGoat, "WebGoat repository," GitHub, https://github.com/WebGoat/WebGoat (accessed May 14, 2023).

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – OWASP Dependency Check report



Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

How to read the report | Suppressing false positives | Getting Help: github issues

Sponsor

Project: WebGoat

org.owasp.webgoat:webgoat:2023.5-SNAPSHOT

Scan Information (show less):

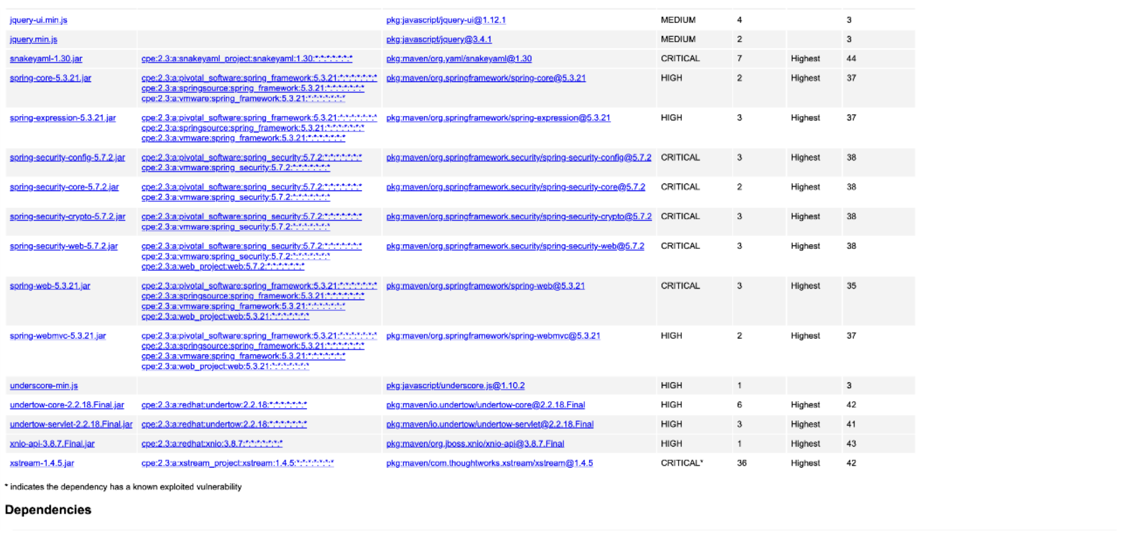
- *dependency-check version*: 8.2.1
- *Report Generated On*: Fri, 21 Apr 2023 00:51:31 +0300
- *Dependencies Scanned*: 288 (249 unique)
- *Vulnerable Dependencies*: 25
- *Vulnerabilities Found*: 121
- *Vulnerabilities Suppressed*: 0
- *NVD CVE Checked*: 2023-04-21T00:51:12
- *NVD CVE Modified*: 2023-04-20T23:00:02
- *VersionCheckOn*: 2023-04-18T15:27:06
- *kev.checked*: 1682027480

Summary

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
bootstrap.min.js		pkg:javascript/bootstrap@3.1.1	MEDIUM	8		3
bootstrap.min.js		pkg:javascript/bootstrap@3.1.1	MEDIUM	8		3
guava-31.1-jre.jar	cpe:2.3:a:google:guava:31.1:*:*:*:*:*:*:*	pkg:maven/com.google.guava/guava@31.1-jre	MEDIUM	1	Highest	25
hsqldb-2.5.2.jar	cpe:2.3:a:hsqldb:hypersql_database:2.5.2:*:*:*:*:*:*:*	pkg:maven/org.hsqldb/hsqldb@2.5.2	CRITICAL	1	Low	41
jackson-databind-2.13.3.jar	cpe:2.3:a:fasterxml:jackson-databind:2.13.3:*:*:*:*:*:*:* cpe:2.3:a:fasterxml:jackson-modules-java8:2.13.3:*:*:*:*:*:*:*	pkg:maven/com.fasterxml.jackson.core/jackson-databind@2.13.3	HIGH	2	Highest	43
jquery-1.10.2.min.js		pkg:javascript/jquery@1.10.2.min	MEDIUM	5		3
jquery-2.1.4.min.js		pkg:javascript/jquery@2.1.4.min	MEDIUM	5		3
jquery-ui-1.10.4.custom.min.js		pkg:javascript/jquery-ui-dialog@1.10.4 pkg:javascript/jquery-ui@1.10.4	MEDIUM	5		5
jquery-ui-1.10.4.js		pkg:javascript/jquery-ui-dialog@1.10.3 pkg:javascript/jquery-ui@1.10.3	MEDIUM	5		5

Figure 13. OWASP Dependency Check report, part 1



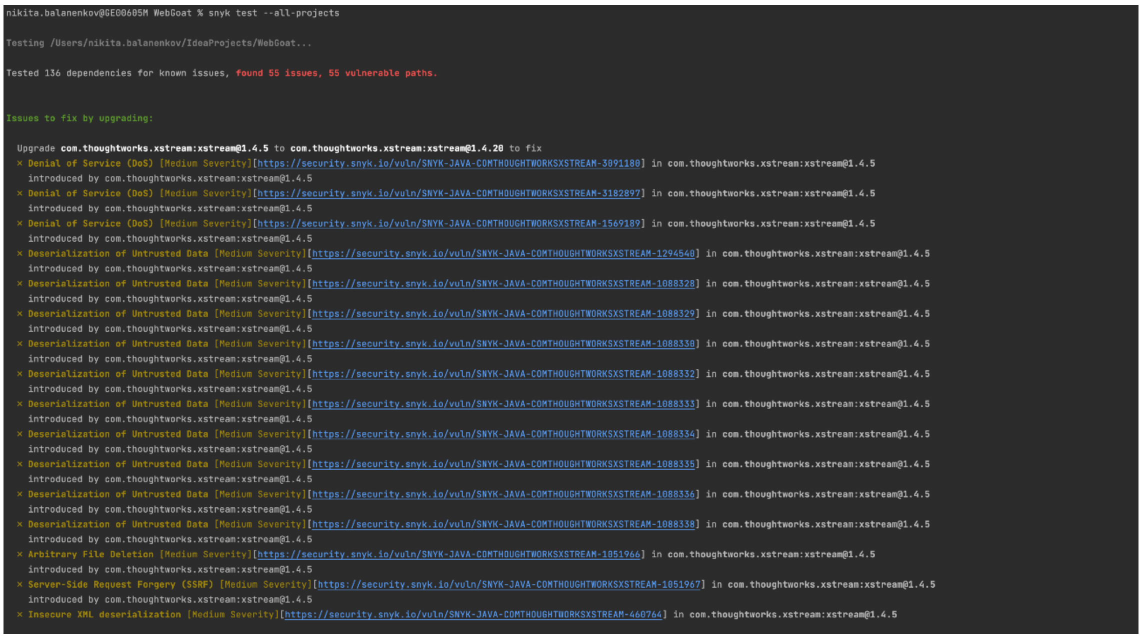
Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
jquery-ui.min.js		pkg:javascript/jquery-ui@1.12.1	MEDIUM	4		3
jquery.min.js		pkg:javascript/jquery@3.4.1	MEDIUM	2		3
snakeyaml-1.30.jar	cpe:2.3:a:snakeyaml_project:snakeyaml:1.30:*:*:*:*:*:*:*	pkg:maven/org.yaml/snakeyaml@1.30	CRITICAL	7	Highest	44
spring-core-5.3.21.jar	cpe:2.3:a:pivotal_software:spring_framework:5.3.21:*:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:5.3.21:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_framework:5.3.21:*:*:*:*:*:*:*	pkg:maven/org.springframework/spring-core@5.3.21	HIGH	2	Highest	37
spring-expression-5.3.21.jar	cpe:2.3:a:pivotal_software:spring_framework:5.3.21:*:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:5.3.21:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_framework:5.3.21:*:*:*:*:*:*:*	pkg:maven/org.springframework/spring-expression@5.3.21	HIGH	3	Highest	37
spring-security-config-5.7.2.jar	cpe:2.3:a:pivotal_software:spring_security:5.7.2:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_security:5.7.2:*:*:*:*:*:*:*	pkg:maven/org.springframework.security/spring-security-config@5.7.2	CRITICAL	3	Highest	38
spring-security-core-5.7.2.jar	cpe:2.3:a:pivotal_software:spring_security:5.7.2:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_security:5.7.2:*:*:*:*:*:*:*	pkg:maven/org.springframework.security/spring-security-core@5.7.2	CRITICAL	2	Highest	38
spring-security-crypto-5.7.2.jar	cpe:2.3:a:pivotal_software:spring_security:5.7.2:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_security:5.7.2:*:*:*:*:*:*:*	pkg:maven/org.springframework.security/spring-security-crypto@5.7.2	CRITICAL	3	Highest	38
spring-security-web-5.7.2.jar	cpe:2.3:a:pivotal_software:spring_security:5.7.2:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_security:5.7.2:*:*:*:*:*:*:* cpe:2.3:a:web_project:web:5.7.2:*:*:*:*:*:*:*	pkg:maven/org.springframework.security/spring-security-web@5.7.2	CRITICAL	3	Highest	38
spring-web-5.3.21.jar	cpe:2.3:a:pivotal_software:spring_framework:5.3.21:*:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:5.3.21:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_framework:5.3.21:*:*:*:*:*:*:* cpe:2.3:a:web_project:web:5.3.21:*:*:*:*:*:*:*	pkg:maven/org.springframework/spring-web@5.3.21	CRITICAL	3	Highest	35
spring-webmvc-5.3.21.jar	cpe:2.3:a:pivotal_software:spring_framework:5.3.21:*:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:5.3.21:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_framework:5.3.21:*:*:*:*:*:*:* cpe:2.3:a:web_project:web:5.3.21:*:*:*:*:*:*:*	pkg:maven/org.springframework/spring-webmvc@5.3.21	HIGH	2	Highest	37
underscore-min.js		pkg:javascript/underscore.js@1.10.2	HIGH	1		3
undertow-core-2.2.18.Final.jar	cpe:2.3:a:redhat:undertow:2.2.18:*:*:*:*:*:*:*	pkg:maven/io.undertow/undertow-core@2.2.18.Final	HIGH	6	Highest	42
undertow-servlet-2.2.18.Final.jar	cpe:2.3:a:redhat:undertow:2.2.18:*:*:*:*:*:*:*	pkg:maven/io.undertow/undertow-servlet@2.2.18.Final	HIGH	3	Highest	41
xnio-api-3.8.7.Final.jar	cpe:2.3:a:redhat:xnio:3.8.7:*:*:*:*:*:*:*	pkg:maven/org.jboss.xnio/xnio-api@3.8.7.Final	HIGH	1	Highest	43
xstream-1.4.5.jar	cpe:2.3:a:xstream_project:xstream:1.4.5:*:*:*:*:*:*:*	pkg:maven/com.thoughtworks.xstream/xstream@1.4.5	CRITICAL*	36	Highest	42

* indicates the dependency has a known exploited vulnerability

Dependencies

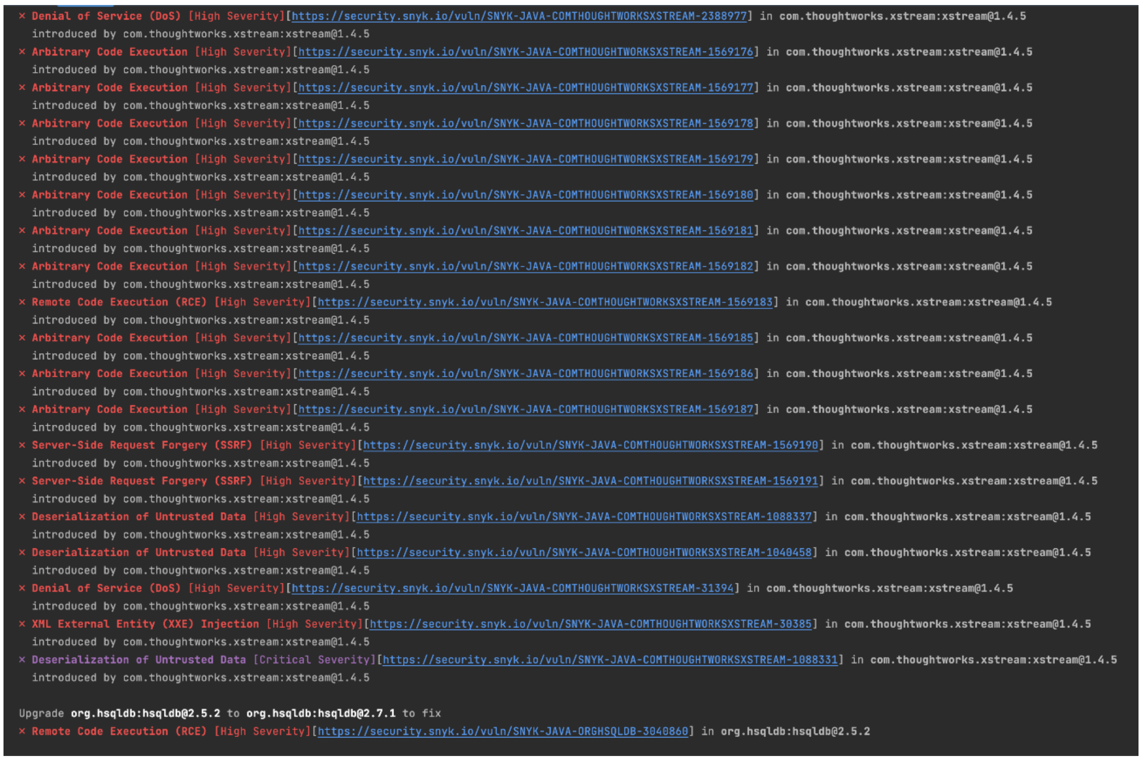
Figure 14. OWASP Dependency Check report, part 2

Appendix 3 – Snyk report



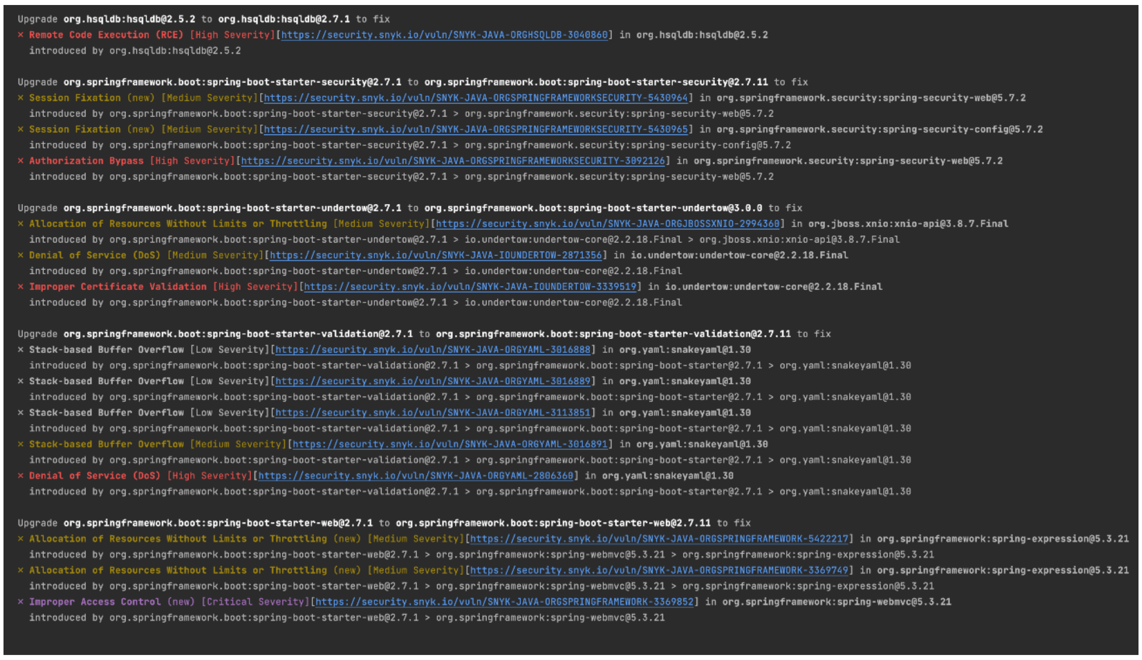
```
nikita.balanenkov@GE00605M WebGoat % snyk test --all-projects

Testing /Users/nikita.balanenkov/IdeaProjects/WebGoat...

Tested 136 dependencies for known issues, found 55 issues, 55 vulnerable paths.


Issues to fix by upgrading:

  Upgrade com.thoughtworks.xstream:xstream@1.4.5 to com.thoughtworks.xstream:xstream@1.4.20 to fix
  × Denial of Service (DoS) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-3091180] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Denial of Service (DoS) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-3182897] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Denial of Service (DoS) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569189] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Deserialization of Untrusted Data [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1294540] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Deserialization of Untrusted Data [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1088328] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Deserialization of Untrusted Data [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1088329] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Deserialization of Untrusted Data [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1088330] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Deserialization of Untrusted Data [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1088332] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Deserialization of Untrusted Data [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1088333] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Deserialization of Untrusted Data [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1088334] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Deserialization of Untrusted Data [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1088335] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Deserialization of Untrusted Data [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1088336] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Deserialization of Untrusted Data [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1088338] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Arbitrary File Deletion [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1051966] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Server-Side Request Forgery (SSRF) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1051967] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Insecure XML deserialization [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-460764] in com.thoughtworks.xstream:xstream@1.4.5
```

Figure 15. Snyk report, part 1



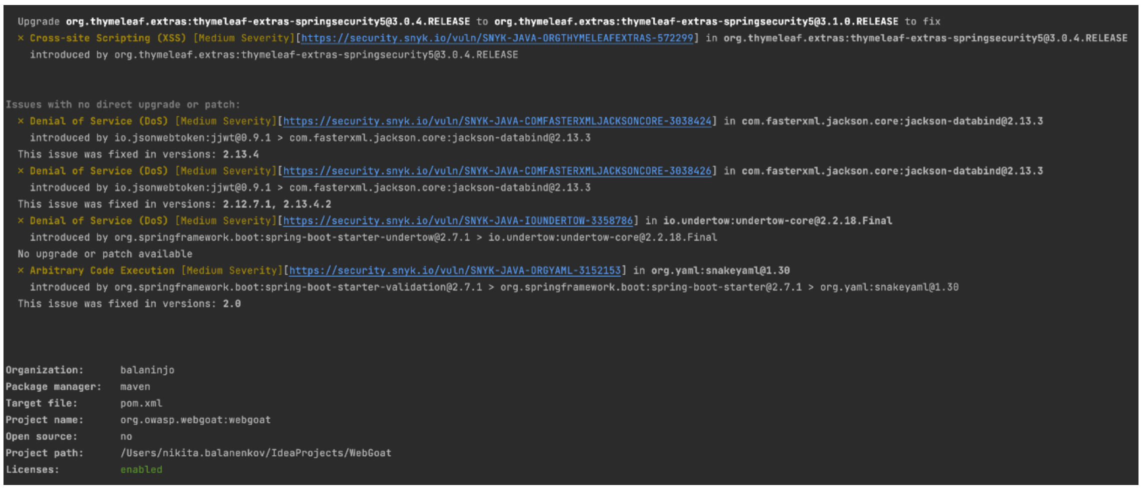
```
  × Denial of Service (DoS) [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-2388977] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Arbitrary Code Execution [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569176] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Arbitrary Code Execution [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569177] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Arbitrary Code Execution [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569178] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Arbitrary Code Execution [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569179] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Arbitrary Code Execution [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569180] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Arbitrary Code Execution [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569181] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Arbitrary Code Execution [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569182] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Remote Code Execution (RCE) [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569183] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Arbitrary Code Execution [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569185] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Arbitrary Code Execution [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569186] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Arbitrary Code Execution [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569187] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Server-Side Request Forgery (SSRF) [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569190] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Server-Side Request Forgery (SSRF) [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1569191] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Deserialization of Untrusted Data [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1088337] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Deserialization of Untrusted Data [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1040458] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Denial of Service (DoS) [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-31394] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × XML External Entity (XXE) Injection [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-30385] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5
  × Deserialization of Untrusted Data [Critical Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMTHOUGHTWORKSXSTREAM-1088331] in com.thoughtworks.xstream:xstream@1.4.5
    introduced by com.thoughtworks.xstream:xstream@1.4.5

  Upgrade org.hsqldb:hsqldb@2.5.2 to org.hsqldb:hsqldb@2.7.1 to fix
  × Remote Code Execution (RCE) [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGHSQLDB-3040860] in org.hsqldb:hsqldb@2.5.2
```

Figure 16. Snyk report, part 2



```
Upgrade org.hsqldb:hsqldb@2.5.2 to org.hsqldb:hsqldb@2.7.1 to fix
× Remote Code Execution (RCE) [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGHSQLDB-3040860] in org.hsqldb:hsqldb@2.5.2
  introduced by org.hsqldb:hsqldb@2.5.2

Upgrade org.springframework.boot:spring-boot-starter-security@2.7.1 to org.springframework.boot:spring-boot-starter-security@2.7.11 to fix
× Session Fixation (new) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGSPRINGFRAMEWORKSECURITY-5430964] in org.springframework.security:spring-security-web@5.7.2
  introduced by org.springframework.boot:spring-boot-starter-security@2.7.1 > org.springframework.security:spring-security-web@5.7.2
× Session Fixation (new) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGSPRINGFRAMEWORKSECURITY-5430965] in org.springframework.security:spring-security-config@5.7.2
  introduced by org.springframework.boot:spring-boot-starter-security@2.7.1 > org.springframework.security:spring-security-config@5.7.2
× Authorization Bypass [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGSPRINGFRAMEWORKSECURITY-3092126] in org.springframework.security:spring-security-web@5.7.2
  introduced by org.springframework.boot:spring-boot-starter-security@2.7.1 > org.springframework.security:spring-security-web@5.7.2

Upgrade org.springframework.boot:spring-boot-starter-undertow@2.7.1 to org.springframework.boot:spring-boot-starter-undertow@3.0.0 to fix
× Allocation of Resources Without Limits or Throttling [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGJBOSSXNIO-2994360] in org.jboss.xnio:xnio-api@3.8.7.Final
  introduced by org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final > org.jboss.xnio:xnio-api@3.8.7.Final
× Denial of Service (DoS) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-IOUNDERTOW-2871356] in io.undertow:undertow-core@2.2.18.Final
  introduced by org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final
× Improper Certificate Validation [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-IOUNDERTOW-3339519] in io.undertow:undertow-core@2.2.18.Final
  introduced by org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final

Upgrade org.springframework.boot:spring-boot-starter-validation@2.7.1 to org.springframework.boot:spring-boot-starter-validation@2.7.11 to fix
× Stack-based Buffer Overflow [Low Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGYAML-3016888] in org.yaml:snakeyaml@1.30
  introduced by org.springframework.boot:spring-boot-starter-validation@2.7.1 > org.springframework.boot:spring-boot-starter@2.7.1 > org.yaml:snakeyaml@1.30
× Stack-based Buffer Overflow [Low Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGYAML-3016889] in org.yaml:snakeyaml@1.30
  introduced by org.springframework.boot:spring-boot-starter-validation@2.7.1 > org.springframework.boot:spring-boot-starter@2.7.1 > org.yaml:snakeyaml@1.30
× Stack-based Buffer Overflow [Low Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGYAML-3113851] in org.yaml:snakeyaml@1.30
  introduced by org.springframework.boot:spring-boot-starter-validation@2.7.1 > org.springframework.boot:spring-boot-starter@2.7.1 > org.yaml:snakeyaml@1.30
× Stack-based Buffer Overflow [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGYAML-3016891] in org.yaml:snakeyaml@1.30
  introduced by org.springframework.boot:spring-boot-starter-validation@2.7.1 > org.springframework.boot:spring-boot-starter@2.7.1 > org.yaml:snakeyaml@1.30
× Denial of Service (DoS) [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGYAML-2806360] in org.yaml:snakeyaml@1.30
  introduced by org.springframework.boot:spring-boot-starter-validation@2.7.1 > org.springframework.boot:spring-boot-starter@2.7.1 > org.yaml:snakeyaml@1.30

Upgrade org.springframework.boot:spring-boot-starter-web@2.7.1 to org.springframework.boot:spring-boot-starter-web@2.7.11 to fix
× Allocation of Resources Without Limits or Throttling (new) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGSPRINGFRAMEWORK-5422217] in org.springframework:spring-expression@5.3.21
  introduced by org.springframework.boot:spring-boot-starter-web@2.7.1 > org.springframework:spring-webmvc@5.3.21 > org.springframework:spring-expression@5.3.21
× Allocation of Resources Without Limits or Throttling (new) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGSPRINGFRAMEWORK-3369749] in org.springframework:spring-expression@5.3.21
  introduced by org.springframework.boot:spring-boot-starter-web@2.7.1 > org.springframework:spring-webmvc@5.3.21 > org.springframework:spring-expression@5.3.21
× Improper Access Control (new) [Critical Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGSPRINGFRAMEWORK-3369852] in org.springframework:spring-webmvc@5.3.21
  introduced by org.springframework.boot:spring-boot-starter-web@2.7.1 > org.springframework:spring-webmvc@5.3.21
```

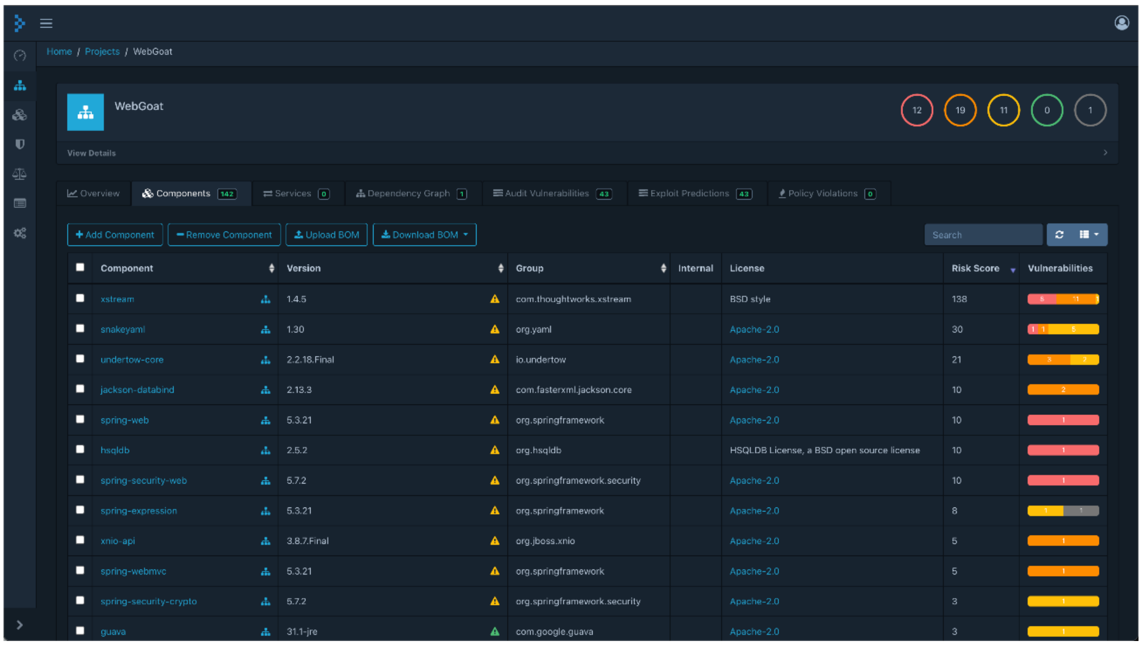
Figure 17. Snyk report, part 3



```
 Upgrade org.thymeleaf.extras:thymeleaf-extras-springsecurity5@3.0.4.RELEASE to org.thymeleaf.extras:thymeleaf-extras-springsecurity5@3.1.0.RELEASE to fix
 × Cross-site Scripting (XSS) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGTHYMELEAFEXTRAS-572299] in org.thymeleaf.extras:thymeleaf-extras-springsecurity5@3.0.4.RELEASE
   introduced by org.thymeleaf.extras:thymeleaf-extras-springsecurity5@3.0.4.RELEASE


Issues with no direct upgrade or patch:
 × Denial of Service (DoS) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMFASTERXMLJACKSONCORE-3038424] in com.fasterxml.jackson.core:jackson-databind@2.13.3
   introduced by io.jsonwebtoken:jjwt@0.9.1 > com.fasterxml.jackson.core:jackson-databind@2.13.3
 This issue was fixed in versions: 2.13.4
 × Denial of Service (DoS) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMFASTERXMLJACKSONCORE-3038426] in com.fasterxml.jackson.core:jackson-databind@2.13.3
   introduced by io.jsonwebtoken:jjwt@0.9.1 > com.fasterxml.jackson.core:jackson-databind@2.13.3
 This issue was fixed in versions: 2.12.7.1, 2.13.4.2
 × Denial of Service (DoS) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-IOUNDERTOW-3358786] in io.undertow:undertow-core@2.2.18.Final
   introduced by org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final
 No upgrade or patch available
 × Arbitrary Code Execution [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGYAML-3152153] in org.yaml:snakeyaml@1.30
   introduced by org.springframework.boot:spring-boot-starter-validation@2.7.1 > org.springframework.boot:spring-boot-starter@2.7.1 > org.yaml:snakeyaml@1.30
 This issue was fixed in versions: 2.0



Organization:      balaninjo
Package manager:   maven
Target file:       pom.xml
Project name:      org.owasp.webgoat:webgoat
Open source:       no
Project path:      /Users/nikita.balanenkov/IdeaProjects/WebGoat
Licenses:          enabled
```

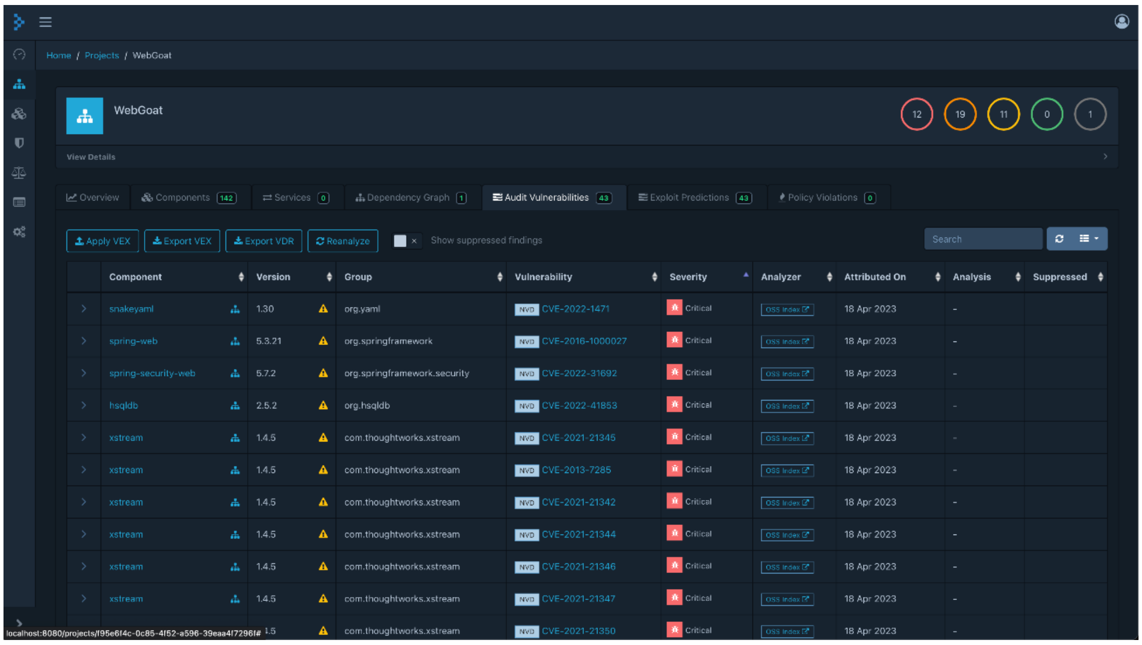
Figure 18. Snyk report, part 4

Appendix 4 – DependencyTrack report



Component	Version	Group	Internal	License	Risk Score	Vulnerabilities
xstream	1.4.5	com.thoughtworks.xstream		BSD style	138	8 11 1
snakeyaml	1.30	org.yaml		Apache-2.0	30	1 1 5
undertow-core	2.2.18.Final	io.undertow		Apache-2.0	21	3 2
jackson-databind	2.13.3	com.fasterxml.jackson.core		Apache-2.0	10	2
spring-web	5.3.21	org.springframework		Apache-2.0	10	1
hsqldb	2.5.2	org.hsqldb		HSQLDB License, a BSD open source license	10	1
spring-security-web	5.7.2	org.springframework.security		Apache-2.0	10	1
spring-expression	5.3.21	org.springframework		Apache-2.0	8	1 1
xnio-api	3.8.7.Final	org.jboss.xnio		Apache-2.0	5	1
spring-webmvc	5.3.21	org.springframework		Apache-2.0	5	1
spring-security-crypto	5.7.2	org.springframework.security		Apache-2.0	3	1
guava	31.1-jre	com.google.guava		Apache-2.0	3	1

Figure 19. DependencyTrack report, part 1

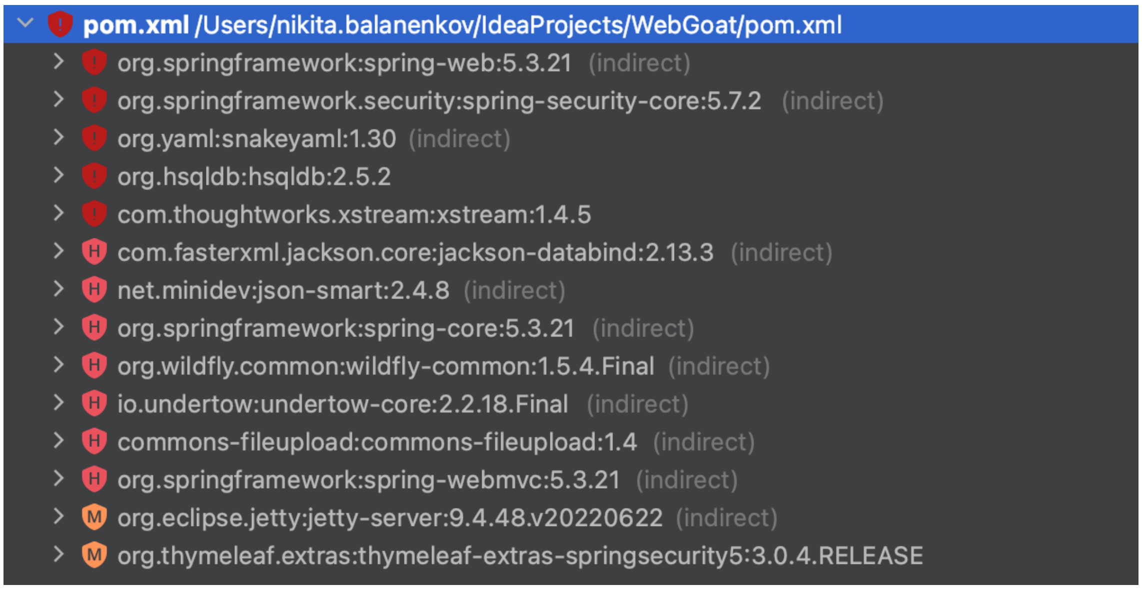


Component	Version	Group	Vulnerability	Severity	Analyzer	Attributed On	Analysis	Suppressed
snakeyaml	1.30	org.yaml	NVD CVE-2022-1471	Critical	OSS Index	18 Apr 2023	-	
spring-web	5.3.21	org.springframework	NVD CVE-2016-1000027	Critical	OSS Index	18 Apr 2023	-	
spring-security-web	5.7.2	org.springframework.security	NVD CVE-2022-31692	Critical	OSS Index	18 Apr 2023	-	
hsqldb	2.5.2	org.hsqldb	NVD CVE-2022-41853	Critical	OSS Index	18 Apr 2023	-	
xstream	1.4.5	com.thoughtworks.xstream	NVD CVE-2021-21345	Critical	OSS Index	18 Apr 2023	-	
xstream	1.4.5	com.thoughtworks.xstream	NVD CVE-2013-7285	Critical	OSS Index	18 Apr 2023	-	
xstream	1.4.5	com.thoughtworks.xstream	NVD CVE-2021-21342	Critical	OSS Index	18 Apr 2023	-	
xstream	1.4.5	com.thoughtworks.xstream	NVD CVE-2021-21344	Critical	OSS Index	18 Apr 2023	-	
xstream	1.4.5	com.thoughtworks.xstream	NVD CVE-2021-21346	Critical	OSS Index	18 Apr 2023	-	
xstream	1.4.5	com.thoughtworks.xstream	NVD CVE-2021-21347	Critical	OSS Index	18 Apr 2023	-	
	.5	com.thoughtworks.xstream	NVD CVE-2021-21350	Critical	OSS Index	18 Apr 2023	-	

localhost:8080/projects/f95e6f4c-0c85-4f52-a596-39eaa4f7296f#

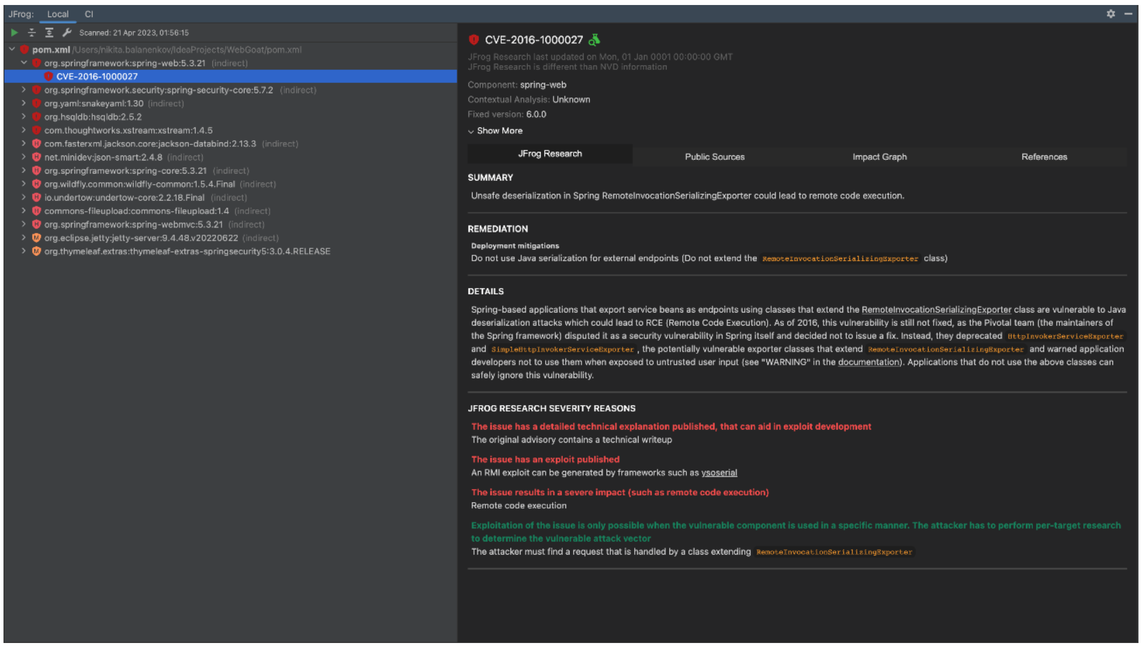
Figure 20. DependencyTrack report, part 2

Appendix 5 – XRay by JFrog report



```
pom.xml /Users/nikita.balanenkov/IdeaProjects/WebGoat/pom.xml
  org.springframework:spring-web:5.3.21  (indirect)
  org.springframework.security:spring-security-core:5.7.2  (indirect)
  org.yaml:snakeyaml:1.30  (indirect)
  org.hsqldb:hsqldb:2.5.2
  com.thoughtworks.xstream:xstream:1.4.5
  H com.fasterxml.jackson.core:jackson-databind:2.13.3  (indirect)
  H net.minidev:json-smart:2.4.8  (indirect)
  H org.springframework:spring-core:5.3.21  (indirect)
  H org.wildfly.common:wildfly-common:1.5.4.Final  (indirect)
  H io.undertow:undertow-core:2.2.18.Final  (indirect)
  H commons-fileupload:commons-fileupload:1.4  (indirect)
  H org.springframework:spring-webmvc:5.3.21  (indirect)
  M org.eclipse.jetty:jetty-server:9.4.48.v20220622  (indirect)
  M org.thymeleaf.extras:thymeleaf-extras-springsecurity5:3.0.4.RELEASE
```

Figure 21. XRay by JFrog report, part 1



```
Scanned: 21 Apr 2023, 01:56:15
pom.xml /Users/nikita.balanenkov/IdeaProjects/WebGoat/pom.xml
  org.springframework:spring-web:5.3.21  (indirect)
    CVE-2016-1000027
  org.springframework.security:spring-security-core:5.7.2  (indirect)
  org.yaml:snakeyaml:1.30  (indirect)
  org.hsqldb:hsqldb:2.5.2
  com.thoughtworks.xstream:xstream:1.4.5
  H com.fasterxml.jackson.core:jackson-databind:2.13.3  (indirect)
  H net.minidev:json-smart:2.4.8  (indirect)
  H org.springframework:spring-core:5.3.21  (indirect)
  H org.wildfly.common:wildfly-common:1.5.4.Final  (indirect)
  H io.undertow:undertow-core:2.2.18.Final  (indirect)
  H commons-fileupload:commons-fileupload:1.4  (indirect)
  H org.springframework:spring-webmvc:5.3.21  (indirect)
  M org.eclipse.jetty:jetty-server:9.4.48.v20220622  (indirect)
  M org.thymeleaf.extras:thymeleaf-extras-springsecurity5:3.0.4.RELEASE
```

CVE-2016-1000027

JFrog Research last updated on Mon, 01 Jan 0001 00:00:00 GMT
JFrog Research is different than NVD information

Component: spring-web
Contextual Analysis: Unknown
Fixed version: 6.0.0
Show More

SUMMARY

Unsafe deserialization in Spring RemoteInvocationSerializingExporter could lead to remote code execution.

REMEDIATION

Deployment mitigations
Do not use Java serialization for external endpoints (Do not extend the `RemoteInvocationSerializingExporter` class)

DETAILS

Spring-based applications that export service beans as endpoints using classes that extend the RemoteInvocationSerializingExporter class are vulnerable to Java deserialization attacks which could lead to RCE (Remote Code Execution). As of 2016, this vulnerability is still not fixed, as the Pivotal team (the maintainers of the Spring framework) disputed it as a security vulnerability in Spring itself and decided not to issue a fix. Instead, they deprecated `HttpInvokerServiceExporter` and `SimpleHttpInvokerServiceExporter`, the potentially vulnerable exporter classes that extend `RemoteInvocationSerializingExporter` and warned application developers not to use them when exposed to untrusted user input (see "WARNING" in the documentation). Applications that do not use the above classes can safely ignore this vulnerability.

JFROG RESEARCH SEVERITY REASONS

The issue has a detailed technical explanation published, that can aid in exploit development
The original advisory contains a technical writeup

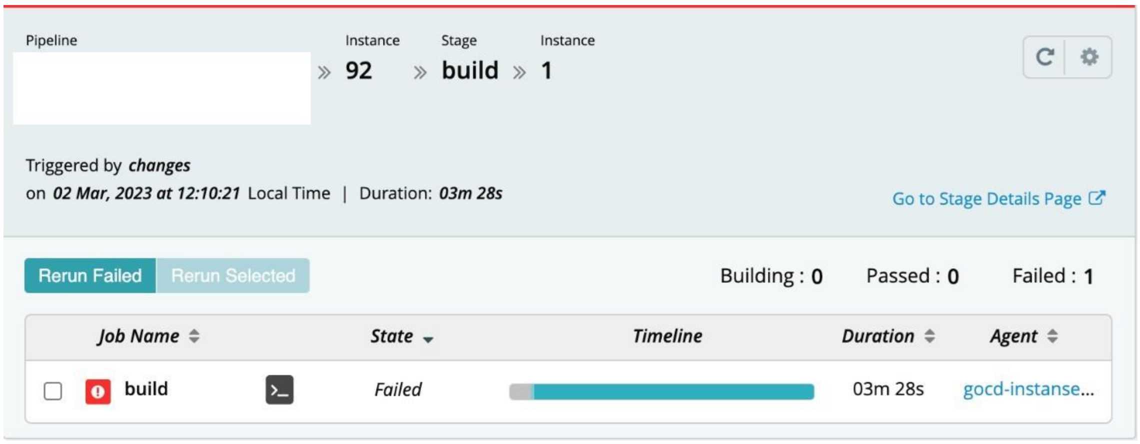
The issue has an exploit published
An RMI exploit can be generated by frameworks such as ysoserial

The issue results in a severe impact (such as remote code execution)
Remote code execution

Exploitation of the issue is only possible when the vulnerable component is used in a specific manner. The attacker has to perform per-target research to determine the vulnerable attack vector
The attacker must find a request that is handled by a class extending `RemoteInvocationSerializingExporter`

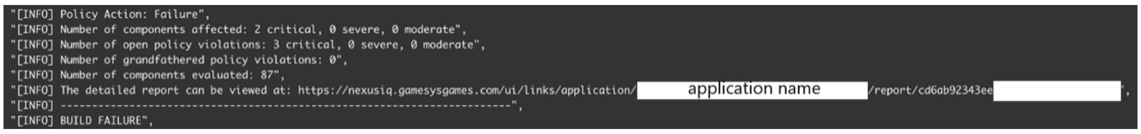
Figure 22. XRay by JFrog report, part 2

Appendix 6 – Sonatype Nexus Lifecycle usage example



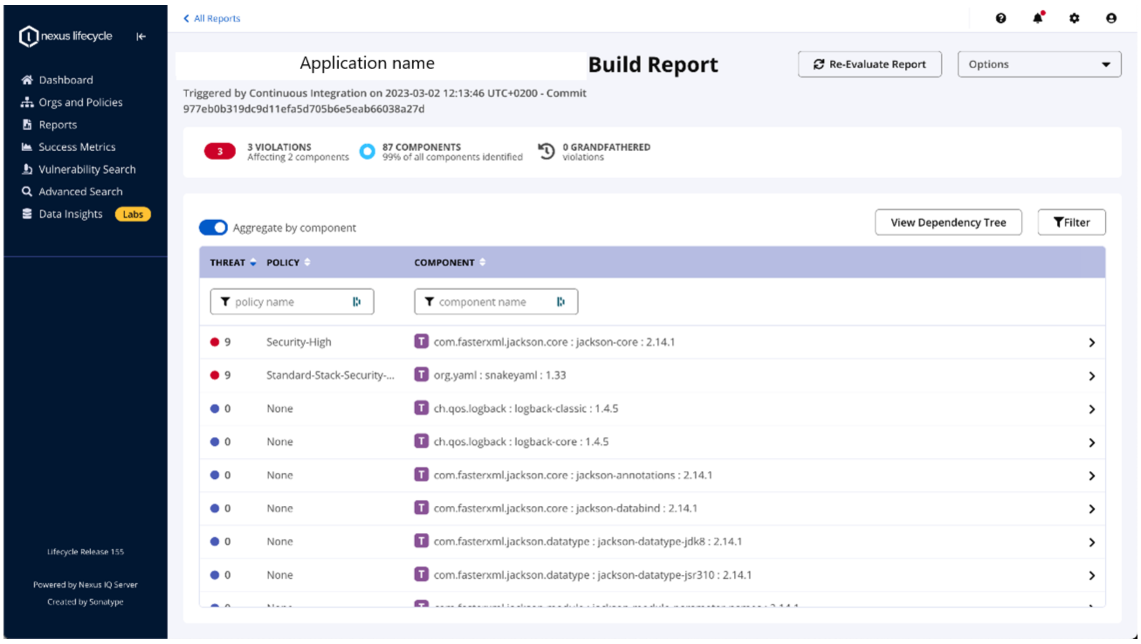
	Job Name		State	Timeline	Duration	Agent
☐	build		*Failed*		03m 28s	gocd-instanse...

Figure 23. Failed build stage



```
"[INFO] Policy Action: Failure",
"[INFO] Number of components affected: 2 critical, 0 severe, 0 moderate",
"[INFO] Number of open policy violations: 3 critical, 0 severe, 0 moderate",
"[INFO] Number of grandfathered policy violations: 0",
"[INFO] Number of components evaluated: 87",
"[INFO] The detailed report can be viewed at: https://nexusiq.gamesysgames.com/ui/links/application/ application name /report/cd6ab92343ee[redacted]",
"[INFO] ------------------------------------------------------------------------",
"[INFO] BUILD FAILURE",
```

Figure 24. Build logs



Application name **Build Report**

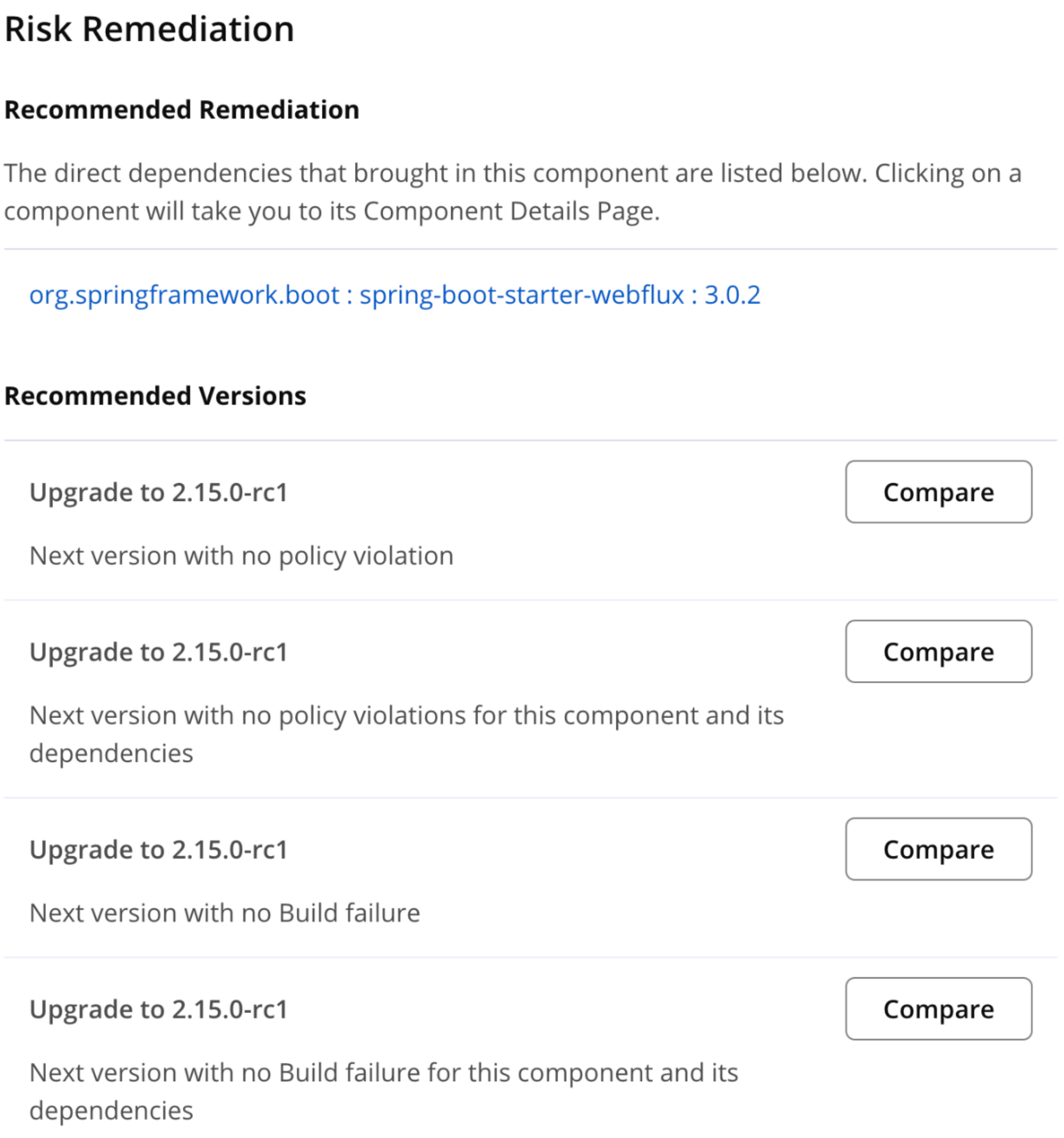
Triggered by Continuous Integration on 2023-03-02 12:13:46 UTC+0200 - Commit 977eb0b319dc9d11efa5d705b6e5eab66038a27d

3 | 3 VIOLATIONS Affecting 2 components | 87 COMPONENTS 99% of all components identified | 0 GRANDFATHERED violations

Aggregate by component

THREAT	POLICY	COMPONENT
	policy name	component name
🔴 9	Security-High	com.fasterxml.jackson.core : jackson-core : 2.14.1
🔴 9	Standard-Stack-Security-...	org.yaml : snakeyaml : 1.33
🔵 0	None	ch.qos.logback : logback-classic : 1.4.5
🔵 0	None	ch.qos.logback : logback-core : 1.4.5
🔵 0	None	com.fasterxml.jackson.core : jackson-annotations : 2.14.1
🔵 0	None	com.fasterxml.jackson.core : jackson-databind : 2.14.1
🔵 0	None	com.fasterxml.jackson.datatype : jackson-datatype-jdk8 : 2.14.1
🔵 0	None	com.fasterxml.jackson.datatype : jackson-datatype-jsr310 : 2.14.1

Figure 25. Application's build report in Sonatype Nexus Lifecycle


Risk Remediation

Recommended Remediation

The direct dependencies that brought in this component are listed below. Clicking on a component will take you to its Component Details Page.

org.springframework.boot : spring-boot-starter-webflux : 3.0.2

Recommended Versions

Upgrade to 2.15.0-rc1

Next version with no policy violation

Upgrade to 2.15.0-rc1

Next version with no policy violations for this component and its dependencies

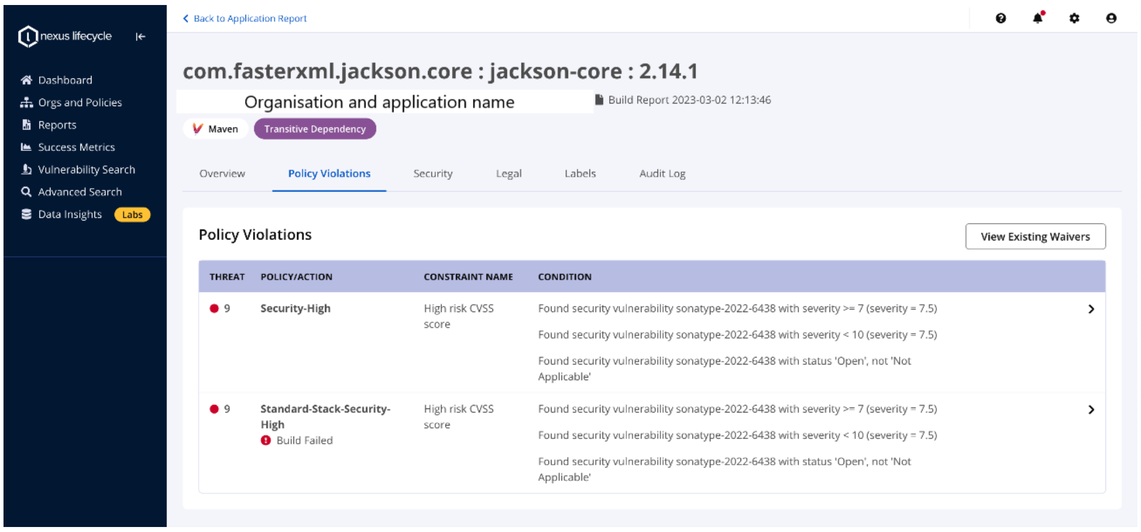
Upgrade to 2.15.0-rc1

Next version with no Build failure

Upgrade to 2.15.0-rc1

Next version with no Build failure for this component and its dependencies

Figure 26. Risk remediation options



com.fasterxml.jackson.core : jackson-core : 2.14.1

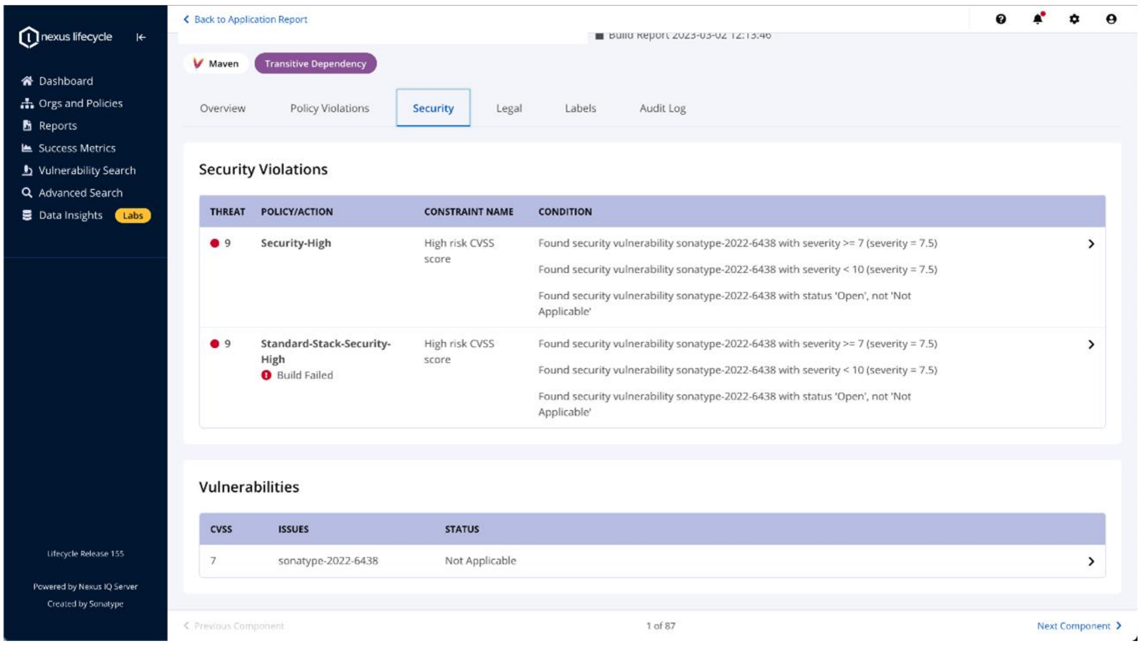
Organisation and application name

Build Report 2023-03-02 12:13:46

Policy Violations

THREAT	POLICY/ACTION	CONSTRAINT NAME	CONDITION
● 9	Security-High	High risk CVSS score	Found security vulnerability sonatype-2022-6438 with severity >= 7 (severity = 7.5) Found security vulnerability sonatype-2022-6438 with severity < 10 (severity = 7.5) Found security vulnerability sonatype-2022-6438 with status 'Open', not 'Not Applicable'
● 9	Standard-Stack-Security-High Build Failed	High risk CVSS score	Found security vulnerability sonatype-2022-6438 with severity >= 7 (severity = 7.5) Found security vulnerability sonatype-2022-6438 with severity < 10 (severity = 7.5) Found security vulnerability sonatype-2022-6438 with status 'Open', not 'Not Applicable'

Figure 27. Policy violation details



Build Report 2023-03-02 12:13:46

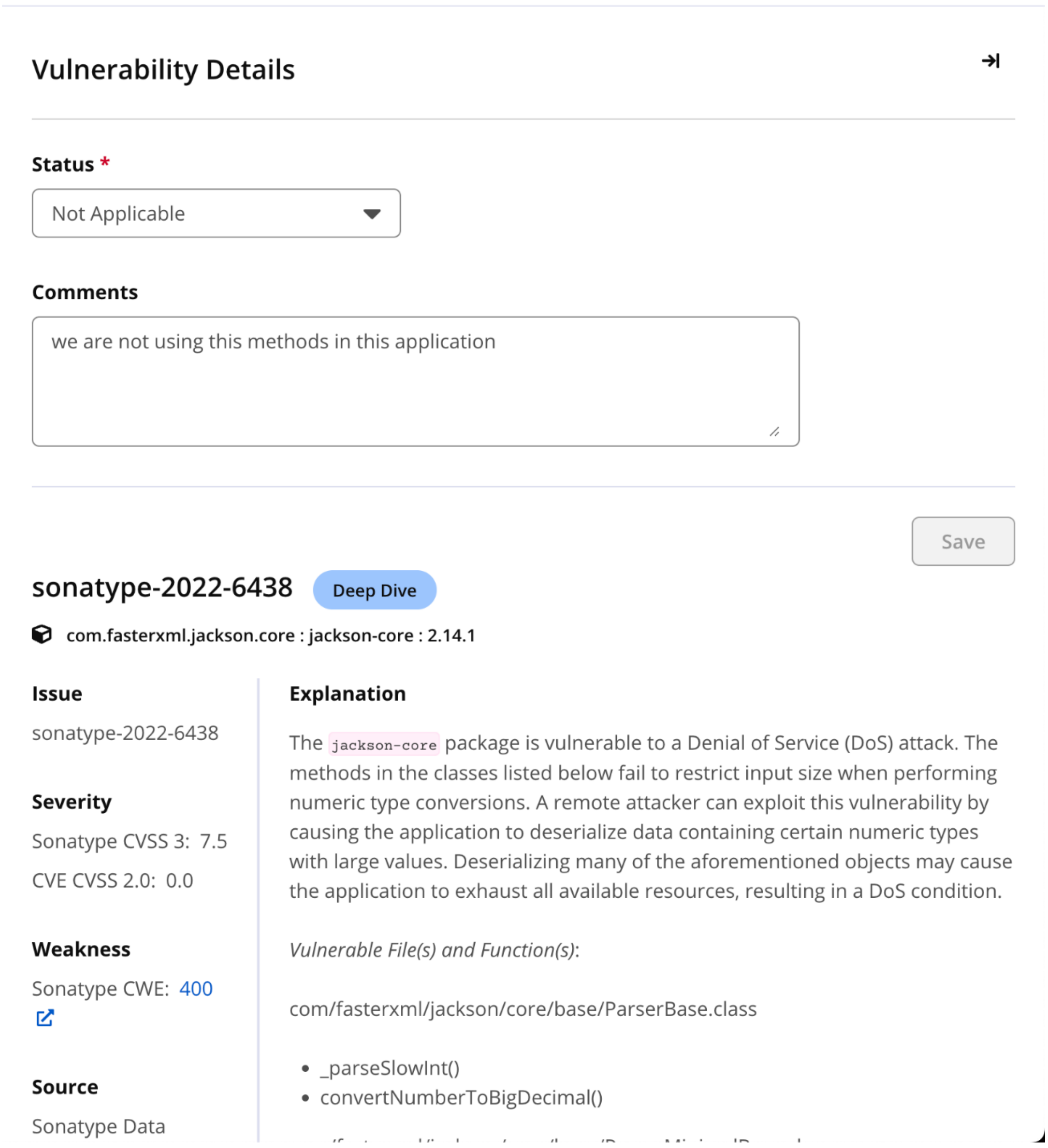
Security Violations

THREAT	POLICY/ACTION	CONSTRAINT NAME	CONDITION
● 9	Security-High	High risk CVSS score	Found security vulnerability sonatype-2022-6438 with severity >= 7 (severity = 7.5) Found security vulnerability sonatype-2022-6438 with severity < 10 (severity = 7.5) Found security vulnerability sonatype-2022-6438 with status 'Open', not 'Not Applicable'
● 9	Standard-Stack-Security-High Build Failed	High risk CVSS score	Found security vulnerability sonatype-2022-6438 with severity >= 7 (severity = 7.5) Found security vulnerability sonatype-2022-6438 with severity < 10 (severity = 7.5) Found security vulnerability sonatype-2022-6438 with status 'Open', not 'Not Applicable'

Vulnerabilities

CVSS	ISSUES	STATUS
7	sonatype-2022-6438	Not Applicable

Figure 28. Security violation details


Vulnerability Details

Status *

Not Applicable

Comments

we are not using this methods in this application

sonatype-2022-6438

com.fasterxml.jackson.core : jackson-core : 2.14.1

Issue

sonatype-2022-6438

Severity

Sonatype CVSS 3: 7.5

CVE CVSS 2.0: 0.0

Weakness

Sonatype CWE: 400

Source

Sonatype Data

Explanation

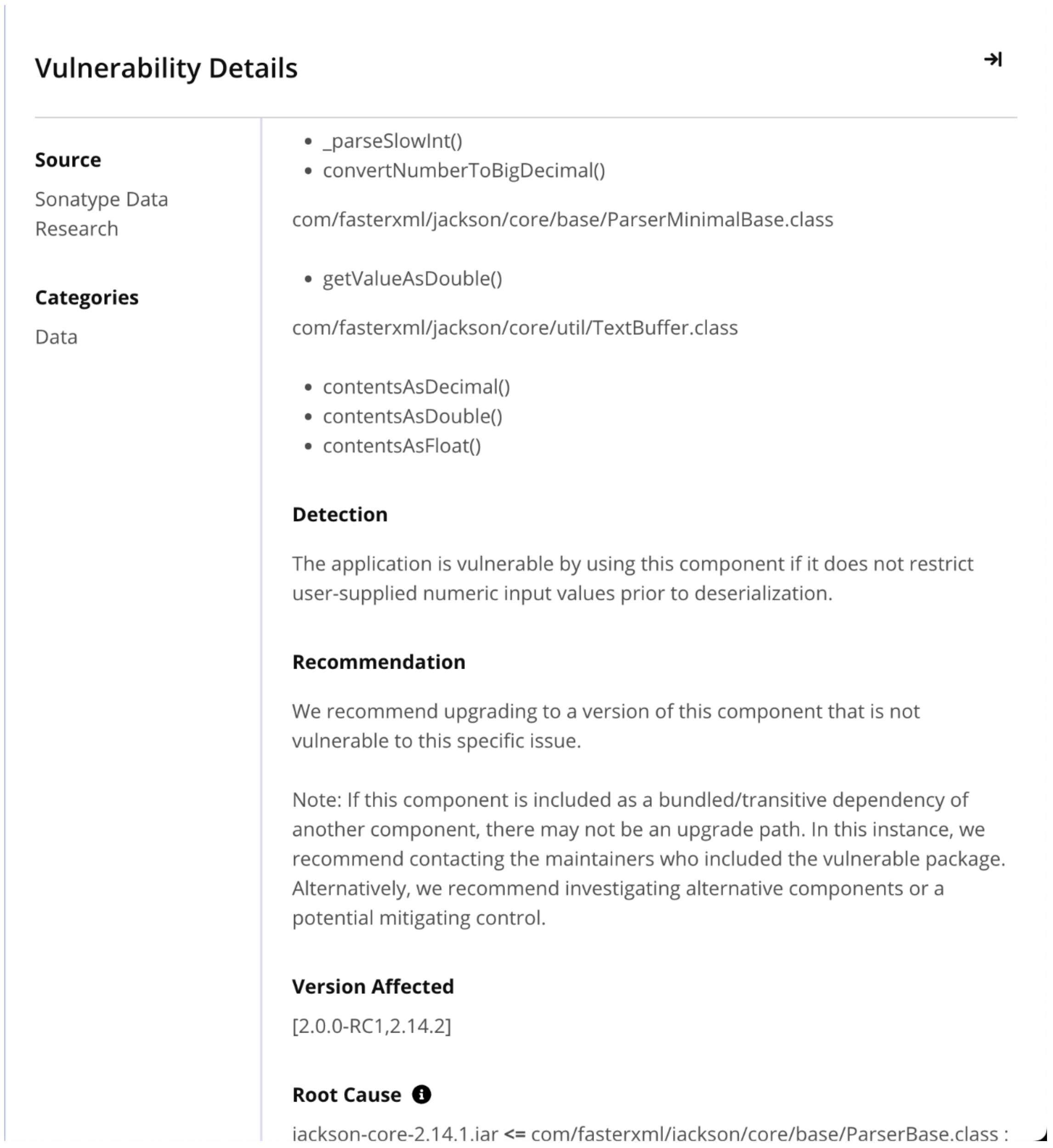
The `jackson-core` package is vulnerable to a Denial of Service (DoS) attack. The methods in the classes listed below fail to restrict input size when performing numeric type conversions. A remote attacker can exploit this vulnerability by causing the application to deserialize data containing certain numeric types with large values. Deserializing many of the aforementioned objects may cause the application to exhaust all available resources, resulting in a DoS condition.

Vulnerable File(s) and Function(s):

com/fasterxml/jackson/core/base/ParserBase.class

- _parseSlowInt()
- convertNumberToBigDecimal()

Figure 29. Vulnerability details, part 1



Vulnerability Details

Source

Sonatype Data Research

Categories

Data

- _parseSlowInt()
- convertNumberToBigDecimal()

com/fasterxml/jackson/core/base/ParserMinimalBase.class

- getValueAsDouble()

com/fasterxml/jackson/core/util/TextBuffer.class

- contentsAsDecimal()
- contentsAsDouble()
- contentsAsFloat()

Detection

The application is vulnerable by using this component if it does not restrict user-supplied numeric input values prior to deserialization.

Recommendation

We recommend upgrading to a version of this component that is not vulnerable to this specific issue.

Note: If this component is included as a bundled/transitive dependency of another component, there may not be an upgrade path. In this instance, we recommend contacting the maintainers who included the vulnerable package. Alternatively, we recommend investigating alternative components or a potential mitigating control.

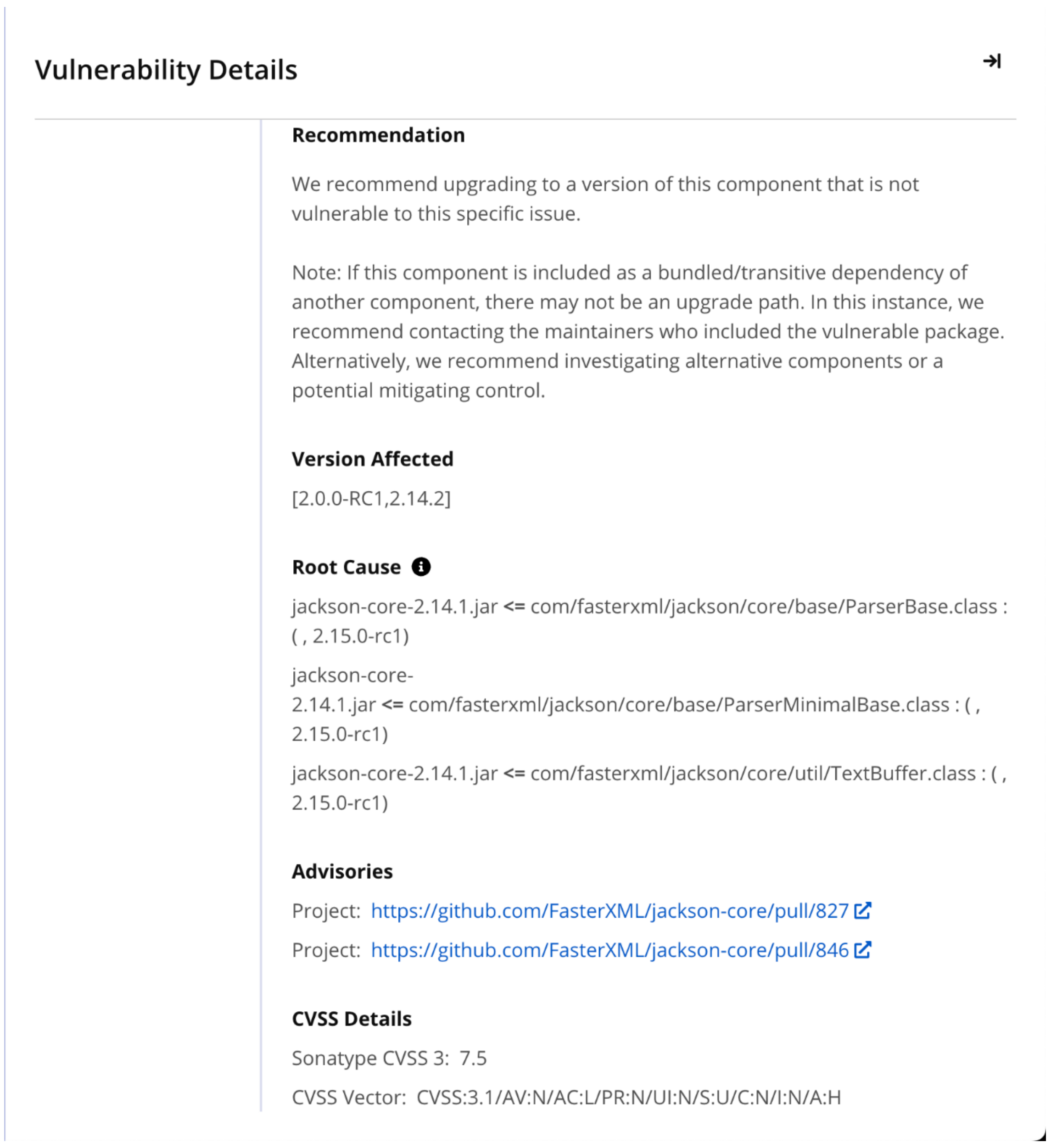
Version Affected

[2.0.0-RC1,2.14.2]

Root Cause

jackson-core-2.14.1.jar **<=** com/fasterxml/jackson/core/base/ParserBase.class :

Figure 30. Vulnerability details, part 2



Vulnerability Details

Recommendation

We recommend upgrading to a version of this component that is not vulnerable to this specific issue.

Note: If this component is included as a bundled/transitive dependency of another component, there may not be an upgrade path. In this instance, we recommend contacting the maintainers who included the vulnerable package. Alternatively, we recommend investigating alternative components or a potential mitigating control.

Version Affected

[2.0.0-RC1,2.14.2]

Root Cause

jackson-core-2.14.1.jar **<=** com/fasterxml/jackson/core/base/ParserBase.class : (, 2.15.0-rc1)

jackson-core-2.14.1.jar **<=** com/fasterxml/jackson/core/base/ParserMinimalBase.class : (, 2.15.0-rc1)

jackson-core-2.14.1.jar **<=** com/fasterxml/jackson/core/util/TextBuffer.class : (, 2.15.0-rc1)

Advisories

Project: https://github.com/FasterXML/jackson-core/pull/827

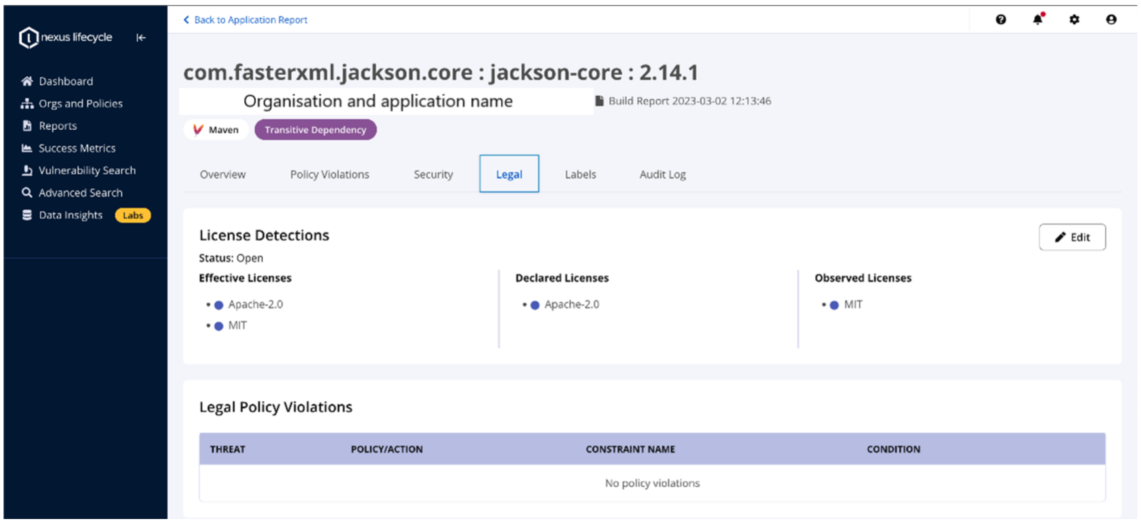
Project: https://github.com/FasterXML/jackson-core/pull/846

CVSS Details

Sonatype CVSS 3: 7.5

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

Figure 31. Vulnerability details, part 3



com.fasterxml.jackson.core : jackson-core : 2.14.1

Organisation and application name

Build Report 2023-03-02 12:13:46

License Detections

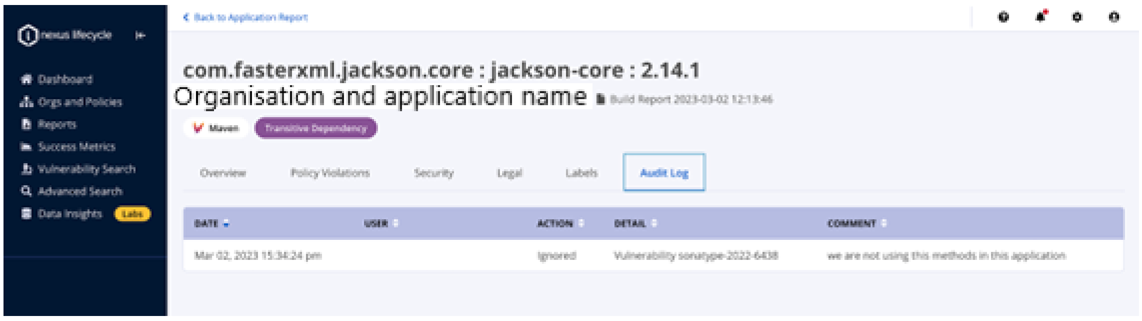
Status: Open

Effective Licenses	Declared Licenses	Observed Licenses
● Apache-2.0 ● MIT	● Apache-2.0	● MIT

Legal Policy Violations

THREAT	POLICY/ACTION	CONSTRAINT NAME	CONDITION
No policy violations			

Figure 32. Legal details



com.fasterxml.jackson.core : jackson-core : 2.14.1

Organisation and application name

Build Report 2023-03-02 12:13:46

DATE	USER	ACTION	DETAIL	COMMENT
Mar 02, 2023 15:34:24 pm		Ignored	Vulnerability sonatype-2022-6438	we are not using this methods in this application

Figure 33. Audit log