JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 18, No. 4, December 2024, pp. 214-223

Expanding a PMD Ruleset for Mitigating Java Deserialization Vulnerabilities

Jisun Lee and Dongsu Kang*

Department of Computer Engineering, Korea National Defense University, Nonsan, Republic of Korea

yasminn0727@gmail.com, dasekang@korea.kr

Abstract

CWE-502 vulnerabilities have been reported over 100 times each year since 2018, comprising more than 1% of all documented vulnerabilities in 2021. However, domestic research on this topic remains scarce. This study applied expanded rules to 4 out of the 6 Rules and Recommendations in the Software Engineering Institute's Computer Emergency Response Team (SEI CERT) Oracle Coding Standard for Java. To mitigate this vulnerability, the PMD ruleset was expanded by referencing the SEI CERT Coding Standard as a static analysis solution. The extended ruleset can be used by utilizing the OWASP Top 10 attack scenarios and the OWASP Deserialization Cheat Sheet. This study emphasizes the significance of deserialization vulnerabilities and aims to enhance the reliability testing and evaluation of system software with Java.

Category: Privacy and Security

Keywords: PMD rule set; Static analysis; Insecure deserialization; CWE-502; Java vulnerability

I. INTRODUCTION

Although coding standards are established, they may sometimes be overlooked to meet specific functional requirements [1]. Additionally, the rapid pace of technological advancement underscores the growing need for comprehensive reliability testing in software. According to the 2024 Cyber Threat Trends Report for the first half of the year, reported cybersecurity incidents reached 899, a 35% rise from 664 in the same period of 2023. This underscores the need to update reliability testing of weapon system software to meet current security requirements.

In this study, we analyze vulnerabilities in open software, with a focus on the most frequently occurring issue, CWE-502. CWE-502 can result in remote code execution (RCE) and denial-of-service (DoS) attacks. Despite its severity and frequency, there is no domestic research on this vulnerability, nor is CWE-502 included in vulnerability checklists for static analysis of weapon system software. This paper proposes a static analysis technique to mitigate CWE-502 using non-compliant and compliant code examples provided by the Software Engineering Institute's Computer Emergency Response Team (SEI CERT). The static analysis tool used for this purpose is PMD, an open-source software.

The structure of this paper is as follows: Section II reviews the static analysis tool PMD and CWE-502. Section III examines the vulnerabilities of open software. Section IV analyzes the SEI CERT Oracle Coding Standard for Java and presents an extension of its PMD ruleset. Section V includes an attack scenario of deserialization. Section VI concludes.

Open Access http://dx.doi.org/10.5626/JCSE.2024.18.4.214 http://jcse.kiise.org

Received 03 December 24; **Accepted** 12 December 24

*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

II. RELATED RESEARCH

A. PMD

PMD is an abbreviation for "programming mistake detector" or "programming malicious code," but it lacks an official designation. PMD is a source code quality analysis tool that identifies potential errors in the code, such as coding style violations, unused variables, and duplicated code, in a static environment. Developed in 2002 with support from organizations like the United States Defense Advanced Research Projects Agency, PMD has undergone regular updates to its diagnostic functions and rules since release. As of 2024, it supports diagnostic rules for 16 programming languages. For Java, PMD offers over 283 diagnostic rules in eight categories.

PMD does not directly parse Java source code; instead, it utilizes Java compiler compiler (JavaCC) to convert source files into an abstract syntax tree (AST). The AST is then used to analyze the code structure and perform static analysis. Additionally, PMD provides an application programming interface (API) for creating custom rules using Java or XPath queries. XPath rules operate on the AST by treating it as a document object model (DOM), which is similar to extensible markup language (XML). The DOM is a hierarchical model that represents the structure of XML or HTML documents. By structuring the AST as XML, XPath can be used to navigate it.

Software for the e-Government information system is mainly developed using the e-Government framework. FindBugs and PMD for source code analysis is advised when implementing software with the Java-based e-Government standard framework [2]. While FindBugs has more diagnostic rules than PMD, it does not allow users to customize or create their own rules. In contrast, PMD enables users to create or extend rules for diagnosing vulnerabilities. As this paper aims to enhance the tool for detecting specific vulnerabilities, PMD was chosen.

B. CWE-502 (Deserialization of Untrusted Data)

1) Vulnerability Classification

As software security becomes increasingly important, so does the need for effective vulnerability management and classification systems. Databases such as common vulnerabilities and exposures (CVE), common weakness enumeration (CWE), and national vulnerability database (NVD) are vital for managing vulnerabilities by unifying and analyzing them, as shown in Fig. 1. The NVD assigns common vulnerability scoring system (CVSS) scores for CVEs. Additionally, the OWASP Top 10 and SANS Top 25 (SysAdmin, Audit, Network, Security) use CWE to classify the most frequent or critical vulnerabilities. There are secure coding guidelines that provide recommendations by leveraging the structured frameworks of these systems, including the SEI CERT guidelines and, domestically, the

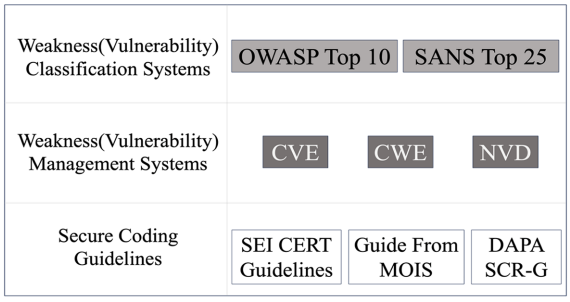

Fig. 1. International and domestic security weakness (vulnerability) sharing systems.

Software Development Security Guide from the Ministry of the Interior and Safety [3], along with the DAPA SCR-G from the Defense Acquisition Program Administration (DAPA) [4].

2) Importance of CWE-502

CWE-502 is one of the critical vulnerabilities that has surged in frequency in recent years. Since 2017, the total number of reported vulnerabilities has consistently exceeded 10,000 annually, with approximately 29,000 cases reported by October 2024. Since 2018, more than 100 cases of CWE-502 have been reported each year, and in 2021, CWE-502 accounted for over 1% of all vulnerabilities. Table 1 shows the CWE-502 statistic results from the NVD, presenting the total number of vulnerabilities and those related to CWE-502 from 2007 to the present.

CWE-502 was first discovered in 2006 but gained prominence in 2015 due to a vulnerability in the Apache Commons Collections library. It was entered the OWASP Top 10 in 2017 as the 8th ranked vulnerability, retaining this rank in the 2021 edition.

CWE-502 was ranked 15th in the 2023 SANS Top 25. This ranking has significantly contributed to advancing active research, focusing on various techniques and approaches for preventing and mitigation of this vulnerability. It has played a key role in fostering extensive studies. The studies primarily address the deserialization overviews, methods attackers use to exploit it and mitigation strategies [1, 5-7], proposed methods for gadget chain detection and linking [8-12], and tools or systems to detect deserialization vulnerabilities [13]. Research in this area remains active in 2024, further advancing the understanding and improving countermeasures for this critical issue.

3) CWE-502

CWE-502 refers to a vulnerability caused by insecure deserialization. Serialization packages internal program object data for external storage or transmission, typically converting it into a binary file or byte stream. This process enables data to be stored in a database or hard drive or transmitted over a network. Conversely, deserialization reconstructs the object structure from a binary file or byte

Table 1. Raw data of CWE-502

Year	Matches	Total	Percentage (%)
2007	1	6,516	0.02
2008	0	5,632	0.00
2009	0	5,732	0.00
2010	2	4,639	0.04
2011	2	4,150	0.05
2012	3	5,288	0.06
2013	2	5,187	0.04
2014	0	7,937	0.00
2015	3	6,487	0.05
2016	9	6,447	0.14
2017	69	14,643	0.47
2018	126	16,509	0.76
2019	134	17,305	0.77
2020	157	18,349	0.86
2021	215	20,155	1.07
2022	170	25,043	0.68
2023	224	28,822	0.78
2024	170	29,318	0.58
Total	1,287	228,159	0.56

stream, extracting serialized data from sequences (arrays).

The deserialization vulnerability occurs when untrusted input source is deserialized without proper verification. Fig. 2 shows how a serialized stream is deserialized. During the deserialization process, an attacker can manipulate the stream, leading to integrity compromise, RCE, or denial of service attacks.

As the significance of CWE-502 has increased, it was included in the 2021 version of the Software Development Security Guide published by the Ministry of the Interior and Safety. However, domestic research on this issue remains insufficient. CWE-502 is also not included in the

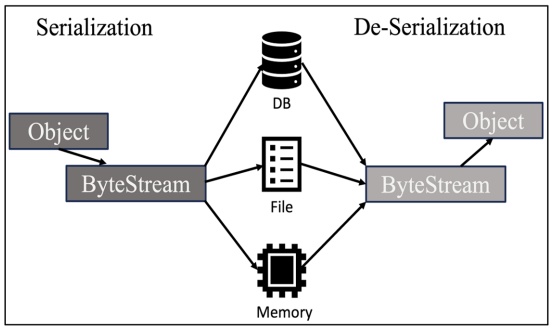

Fig. 2. Serialized object lifecycle.

vulnerability checklists for static tests of weapon system software conducted by the Defense Acquisition Program Administration. Prior to 2022, only CWE-658, CWE-659, and CWE-660 were part of the vulnerability checks. The updated security weakness checklist from 2022 still does not include CWE-502. Thus, there is an urgent need for coding rules and tools that can prevent CWE-502 vulnerabilities.

III. SOFTWARE VULNERABILITY

A. Open-Source Software

To determine how many vulnerabilities could be identified through static analysis, issues reported in the open-source software (OSS) utilized within the United States Department of Defense weapon systems were analyzed. Two significant OSS projects, specifically tailored for use and developed in Java, were selected for analysis. The distributed common ground system (DCGS) is used by the United States military for collecting and producing military intelligence. The distributed data framework (DDF) was specifically created for DCGS as an advanced data integration framework, processing and analyzing large scale data in Java.

The software defined radio (SDR) allows multiple wireless communication services to operate on a single device via software manipulation.

REDHAWK SDR, developed by the United States Department of Defense, is an OSS framework for developing SDR featuring a modular architecture, real-time processing, and a graphical interface.

Out of 782 reported defect issues officially registered in the open-source community for DDF and REDHAWK SDR, a total of 13 remain unresolved and are classified as "Open" issues, while 769 are resolved as "Closed" issues. Upon analyzing the 769 Closed issues, it was discovered that 17 of them were related to CWE, CVE, or the OWASP Top 10.

A total of 47 CVEs were associated with these 17 issues. After eliminating duplicates, 42 unique CVEs were identified. Of these, 28 were related to Java, and the corresponding CWEs are listed in Table 2: CWE-400 (1 issue), CWE-20 (2 issues), CWE-502 (25 issues), CWE-184 (4 issues), CWE-78 (1 issue), and CWE-22 (1 issue). It was evident that CWE-502 occurred most frequently.

For the six CWEs listed in Table 3, examined whether they were included in the essential vulnerability inspection items for weapon system software.

As a result, CWE-22, CWE-78, and CWE-400 were found to be included. In addition, a review of the Software Development Security Guide [3] indicated that secure coding guidelines exist for CWE-22, CWE-78, and CWE-502. Specifically, CWE-22 and CWE-78 have been addressed since 2013 edition. Park [14] expanded the

Table 2. List of Java related CVEs and associated CWEs

No.	CVE list	Associated CWE
1	CVE-2021-44228	CWE-20, CWE-400, CWE-502
2	CVE-2019-17531	CWE-502
3	CVE-2019-17267	CWE-502
4	CVE-2019-16943	CWE-502
5	CVE-2019-16942	CWE-502
6	CVE-2019-16335	CWE-502
7	CVE-2019-12814	CWE-502
8	CVE-2019-12384	CWE-502
9	CVE-2019-12086	CWE-502
10	CVE-2019-10086	CWE-502
11	CVE-2019-0232	CWE-78
12	CVE-2018-19362	CWE-502
13	CVE-2018-19361	CWE-502
14	CVE-2018-19360	CWE-502
15	CVE-2018-14718	CWE-502
16	CVE-2018-12023	CWE-502
17	CVE-2018-12022	CWE-502
18	CVE-2018-11307	CWE-502
19	CVE-2018-7489	CWE-502, CWE-184
20	CVE-2018-5968	CWE-502, CWE-184
21	CVE-2018-133	CWE-22
22	CVE-2017-17485	CWE-502
23	CVE-2017-16026	CWE-20
24	CVE-2017-15095	CWE-502, CWE-184
25	CVE-2016-8749	CWE-502
26	CVE-2017-7525	CWE-502, CWE-184
27	CVE-2016-6809	CWE-502
28	CVE-2015-6420	CWE-502

Table 3. CWE list of previous research

CWE list	Description
CWE-400	Uncontrolled resource consumption
CWE-20	Improper input validation
CWE-502	Deserialization of untrusted data
CWE-184	Incomplete list of disallowed inputs
CWE-78	Improper neutralization of special elements used in an OS command
CWE-22	Improper limitation of a pathname to a restricted directory

PMD ruleset to mitigate CWE-22 and CWE-78. However, CWE-502 has only recently gained recognition for its significance, being added in the 2021 revision, while domestic research remains limited.

B. CWE-502 Analysis

The analysis of 25 CVEs related to CWE-502 revealed that 80% of the vulnerabilities were associated with FasterXML libraries, while 20% were related to Apache libraries. Jackson-databind, developed by FasterXML, is primarily used for serializing and deserializing data in JSON format. And the libraries developed by the Apache Software Foundation (ASF) are widely utilized Java-based open-source programs globally as shown in Fig. 3.

As shown in Table 4, the majority of vulnerabilities posed severe security risks, with a high CVSS (v3.x) score of 9 or even higher. Although the vulnerabilities that most frequently occurred were linked to FasterXML libraries, the Apache vulnerability CVE-2021-44228 received an official Critical rating, along with a maximum CVSS score of 10, indicating extreme severity.

The 2022 Top Routinely Exploited Vulnerabilities report published by Cybersecurity & Infrastructure Security Agency (CISA), CVE-2021-44228 was listed among the top 12 most exploited by attackers and remains a significant concern with various ongoing problems. ASF advises updating to the latest software versions as a mitigation. If immediate updates are not feasible or practical, the removal of the JndiLookup class is suggested. CISA [15] also advises disconnecting affected systems from the network in cases where patching or mitigation remains challenging or unachievable.

Other Apache libraries have also experienced issues due to a lack of input validation during deserialization. CVE-2016-8749 is a vulnerability when deserializing with Jackson and JacksonXML in Apache Camel. CVE-2016-6809 occurs when deserializing MATLAB files using the JMatIO library in Apache Tika. CVE-2015-

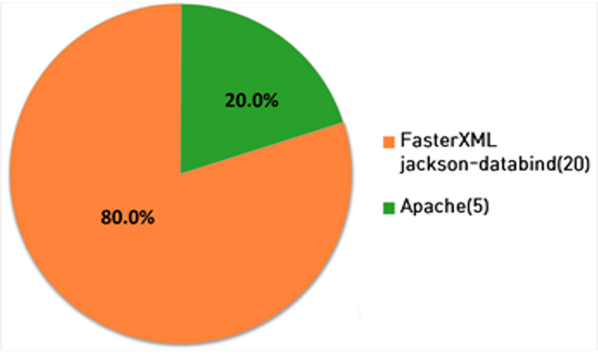

Fig. 3. Common vulnerabilities and exposures related to CWE-502 in distributed data framework.

Table 4. CVE classification and CVSS for CWE-502

Sort	CVE list	CVSS
Apache		
Log4Shell vulnerability	CVE-2021-44228	10
Commons BeanUtils	CVE-2019-10086	7.3
Camel	CVE-2016-8749	9.8
Tika	CVE-2016-6809	9.8
FasterXML: jackson-databind	CVE-2019-17531	9.8
	CVE-2019-17267	9.8
	CVE-2019-16943	9.8
	CVE-2019-16942	9.8
	CVE-2019-16335	9.8
	CVE-2019-12814	5.9
	CVE-2019-12384	5.9
	CVE-2019-12086	7.5
	CVE-2019-10086	7.3
	CVE-2018-19362	9.8
	CVE-2018-19361	9.8
	CVE-2018-19360	9.8
	CVE-2018-14718	9.8
	CVE-2018-12023	7.5
	CVE-2018-12022	7.5
	CVE-2018-11307	9.8
	CVE-2018-7489	9.8
	CVE-2018-5968	8.1
	CVE-2017-17485	9.8
	CVE-2017-15095	9.8
	CVE-2017-7525	9.8

Table 5. FasterXML vulnerability components

FasterXML related CVE list	Associated component
CVE-2019-17531	apache-log4j-extra
CVE-2019-17267	net.sf.ehcache.hibernate.EhcacheJta TransactionManagerLookup
CVE-2019-16943	p6spy(3.8.6) jar
CVE-2019-16942	commons-dbcp(1.4) jar
CVE-2019-16335	com.zaxxer.hikari.HikariDataSource
CVE-2019-12814	JDOM 1.x or 2.x jar
CVE-2019-12384	logback-core gadget
CVE-2019-12086	mysql-connector-java jar(≤8.0.14)
CVE-2019-10086	BeanIntrospector Class
CVE-2018-19362	jboss-common-core Class
CVE-2018-19361	openjpa Class
CVE-2018-19360	axis2-transport-jms Class
CVE-2018-14718	slf4j-ext Class
CVE-2018-12023	Oracle JDBC jar
CVE-2018-12022	Jodd-db jar
CVE-2018-11307	Gadget class of iBatis
CVE-2018-7489	c3p0 library, Incomplete fixes for CVE-2017-7525
CVE-2018-5968	Limitations of the blacklist approach, Incomplete fixes for CVE-2017-7525 and CVE-2017-17485
CVE-2017-17485	Spring library, Incomplete fixes for CVE-2017-7525
CVE-2017-15095	Limitations of the blacklist approach, Incomplete fixes for CVE-2017-7525
CVE-2017-7525	Sent to the readValue method of ObjectMapper

6420 occurs when Apache Commons Collections is used for deserialization in CISCO products. Mitigation measures for these vulnerabilities include updating Apache Camel (to 2.16.5, 2.17.5, 2.18.2, or later), updating Apache Tika (to 1.18-1 or later), and updating Commons Collections (to 3.2.2 or later).

Vulnerabilities related to the Jackson-databind library from FasterXML mainly arise from processing polymorphic types. This library deserializes specific classes when default typing is enabled. Converting JSON or XML data into objects, there is a risk of arbitrary code execution if certain modules or components are present in the classpath.

For example, components related to CVE-2019-17531 is apache-log4j-extra, an Apache Log4j extension providing additional logging features and various logging patterns. Including apache-log4j-extra in the classpath, it poses a potential risk of RCE during Jackson deserialization.

Mitigating FasterXML vulnerabilities requires maintaining the latest Jackson version. Keeping software update is crucial for security. Although various studies exist, this paper does not detail them as they are not directly relevant.

However, a review of related research highlights challenges in the immediate application of patches and in identifying changes. Many companies either do not provide the relevant source code or offer security updates without annotations, complicating response measures. Therefore, fundamental coding rules and countermeasures for CWE-502 are necessary and develop countermeasures. Despite this urgency, research remains insufficient. Consequently, this paper proposes a static analysis method using PMD to mitigate CWE-502 vulnerabilities. FasterXML vulnerability components are shown in Table 5.

IV. PMD RULESET EXPANSION

A. SEI Coding Standard

The CERT from the SEI of Carnegie Mellon University has developed secure coding standards. These standards provide guidelines for programmers for enhancing the safety and security of software systems.

The SEI CERT Oracle Coding Standard for Java defines Java coding rules to reduce security vulnerabilities from programmer errors. It consists of Rules and Recommendations.

- Rules: Violating these rules may cause defects impacting system safety, reliability, or security.
- Recommendations: Implementing recommendations can enhance the safety, reliability, or security of the software system.

Six rules and recommendations related to CWE-502 are listed in Table 6.

SER01-J specifies that readObject(), writeObject(), and readObjectNoData() must be declared as private and non-static. Declaring these methods as public allows invocation by untrusted code, while declaring them as private and non-static restricts access, preventing malicious overriding. These methods are essential during serialization and deserialization processes; their private and non-static declaration ensures that they cannot be accessed or overridden externally, preserving object integrity and enhancing security.

The SER03-J rule addresses the security risks associated with sensitive data. Even if declared as private fields, sensitive data (e.g., encryption keys, digital certificates, or confidential information serialized into byte streams) may still be exposed during serialization. Attackers can reconstruct this data, posing significant risks. To mitigate this risk, the transient keyword should be used to exclude these fields from the serialization process.

SER06-J addresses the risk of manipulation when a class contains private mutable objects. Mutable objects include types such as Date, Calendar, StringBuilder, ArrayList, HashMap, and HashSet, which can change their internal state. If these objects are modified externally, malicious data may be added or existing data removed. To mitigate this risk, defensive copying should be employed by returning a copy of the mutable object instead of the original, thus preserving the integrity of the original data.

SER07-J advises against using the default serialization format for classes that are intended to be immutable, such as Singleton classes. A singleton class is designed to have only one instance; however, the default serialization format can create new instances, violating the class's immutability and compromising stability. The readResolve() method is invoked immediately after deserialization to ensure that any newly created object during the process is replaced with the original instance.

SER12-J requires verifying object types before invoking the readObject() method. The resolveClass() method validates class information. A whitelist of allowed classes should be set in resolveClass() to block unauthorized class deserialization.

SEC58-J, part of Rec. 15 Platform Security (SEC), relates to CWE-502. It advises against performing dangerous operations (e.g., file manipulation, network connections, or command execution) in the readObject() method. If fields managing external resources are defined, proper exception handling should be implemented to minimize security risks.

B. PMD Ruleset Expansion

This study expanded the PMD ruleset by selecting four out of five Rules and one Recommendation from the SEI CERT Oracle Java Coding Standard. SER06-J was excluded from this expansion due to its reliance on the use of mutable objects, which varies depending on implementation and design, making it unsuitable for detection using PMD. Additionally, Recommendations were excluded as they have a lower priority compared to Rules. Consequently, SER01-J, SER03-J, SER07-J, and SER12-J were selected and incorporated into the extended ruleset.

Table 6. CWE-502 related rules and recommendations

Rules and recommendations	Details
Rule 14. Serialization	
SER01-J	Do not deviate from the proper signatures of serialization methods.
SER03-J	Do not serialize unencrypted sensitive data.
SER06-J	Make defensive copies of private mutable components during deserialization.
SER07-J	Do not use the default serialized form for classes with implementation-defined invariants.
SER12-J	Prevent deserialization of untrusted data.
Rec. 15. Platform Security	
SEC58-J	Deserialization methods should not perform potentially dangerous operations.

Table 7. SER01-J XML rule implementation

XML element	Description
name	SerializationAccessCheck
class	net.sourceforge.pmd.lang.rule.XPathRule
language	java
message	Serialization/deserialization methods lack private and non-static declarations.
description	To prevent access and overrides, readObject(), readObjectNoData(), writeObject(), and readResolve() must be private, non-static.
properties	xpath
value	//ClassOrInterfaceDeclaration[ImplementsList/ClassOrInterfaceType[@Image='Serializable']]// MethodDeclaration[MethodDeclarator[@Image='readObject' or @Image='writeObject' or @Image='readObjectNoData' or @Image='writeReplace' or @Image='readResolve'] and (@Private='false' or @Static='true')]
priority	1

1) SER01-J Ruleset Expansion

To generate a query, the AST of the Compliant Solution must be analyzed. This can be visually explored using PMD's Rule Designer. The 'class' field represents the class in which the Java rule is defined, and for XPath. The rule implementation that matches the XML elements can be found in Table 7.

The ‘language’ field specifies the target language as Java. And the ‘message’ field contains the error message displayed if the rule is violated. The ‘description’ explains the rule, and ‘properties’ is set to xpath if XPath is used. The ‘value’ contains the XPath query defining the diagnostic criteria.

The 'priority' field indicates the rule's importance, ranging from 1 (high) to 5 (low). This is based on the priority value from the SEI CERT Oracle Coding Standard for Java's Risk Assessment. SER01-J, with the highest score of P27, is assigned a PMD priority of 1 (high).

2) SER03-J Ruleset Expansion Method

The value contains a query that identifying non-serializable classes and detecting input/output stream classes without proper exception handling.

The table omits the class, language, and properties elements that overlap with the SER01-J XML rule implementation. The rule is to indicate its purpose of preventing unintended serialization. The priority reflects the value provided in the SER03-J Risk Assessment. Since the assigned priority is P6, the priority is set to 4 (Medium_Low). Table 8 demonstrates the implementation of the SER03-J rule.

3) SER07-J Ruleset Expansion Method

The rule detects instances where readResolve() is absent and the deserialized instance is not replaced with the current valid Singleton instance.

The rule, “CheckReadResolveInSingleton,” verifies readResolve() in Singleton classes. The value contains a query based on the diagnostic steps detecting cases where INSTANCE or instance exists without the declaration of readResolve(). The priority of PMD, derived from the SER07-J Risk Assessment, reflects a P4 assessment, resulting in a low priority 5 (low). Table 9 presents the SER-07 rule.

4) SER12-J Ruleset Expansion Method

Building a whitelist is challenging as it limits allowed classes to those approved by developers. The provided

Table 8. SER03-J XML rule implementation

XML element	Description
name	BlockUnintendedSerialization
message	Non-serializable class should throw NotSerializableException in writeObject(), readObject(), or readObjectNoData().
description	Non-serializable classes must throw NotSerializableException to prevent unintended serialization of subclasses when using ObjectOutputStream or ByteArrayOutputStream.
value	//ClassOrInterfaceDeclaration[@Public='true' and not(ImplementsList/ClassOrInterfaceType[@Image='Serializable']) and not(.//NameList/Name[@Image='NotSerializableException']) and //ClassOrInterfaceBody//ClassOrInterfaceType[@Image='ObjectOutputStream' or @Image='ByteArrayOutputStream']]
priority	4

Table 9. SER07-J XML rule implementation

XML element	Description
name	CheckReadResolveInSingleton
message	readResolve() is not implemented in a singleton class with an INSTANCE or instance field.
description	Classes with singleton fields like instance or INSTANCE must implement readResolve() to ensure singleton pattern safety.
value	//ClassOrInterfaceDeclaration[.//VariableDeclarator/VariableDeclaratorId[@Name='INSTANCE' or @Name='instance'] and not(.//MethodDeclaration/MethodDeclarat or [@Image='readResolve'])]
priority	5

Table 10. SER12-J XML rule implementation

XML element	Description
name	WhitelistCheckInDeserialization
message	No whitelist exists in deserialization process.
description	When declaring deserialization methods like ObjectInputStream, readObject, readResolve, and readObjectNoData, include a whitelist. Example whitelist: "java.lang.String", //Allow String class "java.util.ArrayList", //Allow ArrayList class "java.lang.Integer", //Allow Integer class "java.util.HashMap", //Allow HashMap class "java.util.HashSet", //Allow HashSet class "com.example.AllowedClass1" //Add user-defined class AllowedClass1
value	//ClassOrInterfaceDeclaration[(.//MethodCall[MethodName='readObject' or MethodName='readObjectNoData' or MethodName='readResolve'] or .//ClassOrInterfaceType[@Image='ObjectInputStream']) and not(.//VariableDeclaratorId[contains(@Image, 'whitelist')]) and not(.//MethodDeclaration[@Name='resolveClass'])]
priority	4

Table 11. Attack scenario

Order	Description
1	A react application (frontend) interacts with a Spring Boot (backend) microservice.
2	Developers aimed to maintain immutability but designed the system to serialize user state for data exchange with each request.
3	The attacker discovers repeatedly occurring serialized data. Upon finding the "rO0" Java object signature encoded in Base64, they initiate an attack.
4	They use tools like Java Serial Killer and Ysoserial to attempt remote code execution on the application server.

description is for reference only; the actual rule requires careful review. The rule detects specific terms but not their context. Tools like SWAT (Serial Whitelist Application Trainer) aid in creating effective whitelists. The PMD priority reflects the SER12-J Risk Assessment of P9, assigning a priority 4 (medium-low). Table 10 shows the SER12-J rule implementation

V. ATTACK SCENARIO

The OWASP Top 10 provides attack scenarios for each rank. Under A08:2021 – Software and Data Integrity Failures, Table 11 outlines "Scenario: Insecure Deserialization," linked to CWE-502. Java serialized objects share three traits: (1) Begin with "AC ED 00 05" in hex, (2) Begin with "rO0" in Base64, and (3) Content-Type is set to application/x-java-serialized-object in HTTP response headers.

Attackers identify Java serialized objects by recognizing the "rO0" pattern, which indicates the beginning of serialized data in Java. This pattern helps attackers confirm that the data is serialized. And Java serialized data in hexadecimal format always begins with "AC ED 00 05." Attackers can use this signature to identify serialized data.

Another approach is to examine the HTTP response headers. If the Content-Type is set to application/x-java-serialized-object, it indicates that the response contains a serialized Java object. Burp Suite can be employed to analyze HTTP response headers. Burp Suite is a toolkit designed to detect security vulnerabilities in web applications by analyzing requests and responses.

According to Attack Scenario, vulnerabilities can arise from issues occurring during the deserialization process. These vulnerabilities can be mitigated by rules in the expanded PMD ruleset, which validate during deserialization. The SER07-J rule detects cases where deserialized instances are not replaced with appropriate objects. In addition, the SER12-J rule identifies instances where deserialized classes are not included in the whitelist.

The OWASP Cheat Sheet series was developed to provide information on specific security topics. The Deserialization Cheat Sheet outlines three key guidelines: declare sensitive data using the transient keyword, prevent deserialization of domain objects by throwing exceptions in the readObject method, especially in applications implementing serializable due to hierarchical structures, and restrict allowed classes by utilizing the resolveClass method.

While the recommendation to use the transient keyword was presented in the first example of SER03-J, it was excluded from the PMD ruleset extension due to the broad scope of sensitive data. The remaining two guidelines are addressed by the extended PMD ruleset through SER03-J and SER12-J. The SER03-J rule detects non-compliant code in serializable classes that fail to handle exceptions to prevent improper serialization of subclasses. Meanwhile, SER12-J detects cases where deserialized classes are not included in the whitelist. The expanded PMD ruleset improves CWE-502 vulnerability validation and is expected to lower attack risks.

VI. CONCLUSION

This study examined software vulnerabilities in Java, specifically identifying CWE-502 as the most commonly occurring vulnerability. In response, we expanded the PMD ruleset for Java to address and mitigate this vulnerability.

We proposed the extended PMD ruleset to effectively mitigate risks associated with CWE-502. The proposed four rules are designed to detect patterns resembling non-compliant code examples in the SEI CERT Oracle Coding Standard for Java. These rules aim to encourage adherence to secure coding practices in real-world development environments, thereby improving both code quality and security.

CONFLICT OF INTEREST

The authors have declared that no competing interests exist.

REFERENCES

1. A. Sabatini, "Evaluating the testability of insecure deserialization vulnerabilities via static analysis," 2020 [Online]. Available: https://www.politesi.polimi.it/handle/10589/187947.
2. J. Nam, "Analysis and extension of the PMD rule-set for the source code security strengthening of IT systems," M.S. thesis, Korea University, Seoul, Korea, 2015.
3. Korea Internet & Security Agency, "Software Development Security Guide," 2021 [Online]. Available: https://www.kisa.or.kr/2060204/form?postSeq=5&lang_type=KO&page=1.
4. Defense Acquisition Program Administration, "Weapon systems software development and management manual," 2020 [Online]. Available: https://www.dapa.go.kr/dapa/na/ntt/selectNttInfo.do?bbsId=462&nttSn=33009&menuId=335.
5. I. Sayar, A. Bartel, E. Bodden, and Y. Le Traon, "An in-depth study of java deserialization remote-code execution exploits and vulnerabilities," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 1, article no. 25, 2023. https://doi.org/10.1145/3554732
6. S. F. Fingann, "Java deserialization vulnerabilities," M.S. thesis, University of Oslo, Oslo, Norway, 2020.
7. S. Cristalli, "Static and dynamic analyses for protecting the java software execution environment," Ph.D. dissertation, University of Milano, Milano, Italy, 2020.
8. B. Chen, L. Zhang, X. Huang, Y. Cao, K. Lian, Y. Zhang, and M. Yang, "Efficient detection of Java deserialization gadget chains via bottom-up gadget search and dataflow-aided payload construction," in *Proceedings of 2024 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2024, pp. 3961-3978. https://doi.org/10.1109/SP54263.2024.00150
9. S. Cao, X. Sun, X. Wu, L. Bo, B. Li, R. Wu, et al., "Improving Java deserialization gadget chain mining via overriding-guided object generation," in *Proceedings of 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, Melbourne, Australia, 2023, pp. 397-409. https://doi.org/10.1109/ICSE48619.2023.00044
10. S. Cao, B. He, X. Sun, Y. Ouyang, C. Zhang, X. Wu, et al., "ODDFuzz: discovering Java deserialization vulnerabilities via structure-aware directed greybox fuzzing," in *Proceedings of 2023 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2023, pp. 2726-2743. https://doi.org/10.1109/SP46215.2023.10179377
11. S. Rasheed and J. Dietrich, "A hybrid analysis to detect java serialisation vulnerabilities," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, Virtual Event, Australia, 2020, pp. 1209-1213. https://doi.org/10.1145/3324884.3418931
12. J. C. Santos, M. Mirakhorli, and A. Shokri, "Seneca: taint-based call graph construction for Java object deserialization," *Proceedings of the ACM on Programming Languages*, vol. 8(OOPSLA1), pp. 1125-1153, 2024. https://doi.org/10.1145/3649851
13. Q. Zhang, Y. Xu, Z. Yin, C. Zhou, and Y. Jiang, "Automatic policy synthesis and enforcement for protecting untrusted deserialization," in *Proceedings of the 31st* Annual *Network and Distributed System Security (NDSS) Symposium*, San Diego, CA, USA, 2024, pp. 1-18.
14. J. H. Park, "A study of security rules for checking software weaknesses related to input validation," M.S. thesis, Dankook University, Yongin, Korea, 2015.
15. Cybersecurity & Infrastructure Security Agency, "Apache

Log4j Vulnerability Guidance," 2022 [Online]. Available: https://www.cisa.gov/news-events/news/apache-log4j-vulnerability-guidance.

Jisun Lee https://orcid.org/0009-0000-6237-6220

Jisun Lee is currently pursuing the M.S. degree in computer engineering at Korea National Defense University, South Korea. She is also a signal officer with the Ministry of Defense, specializing in military communications. Her current research interests include SW security testing, SW reliability, and coding rules.

Dongsu Kang https://orcid.org/0000-0001-6481-5071

Dongsu Kang is currently a professor of Computer Science and Engineering and director of the Department of Defense Science, Korea National Defense University. His main area of expertise is software security testing, penetration testing, AI-based systems testing, naval cyber security, weapon system software, North Korea software, interoperability of defense system, machine learning, defense modeling and simulation, and defense acquisition. He was the director of Defense Science Center in Research Institute for National Security Affairs.