2017 IEEE Symposium on Security and Privacy Workshops

IDE Plugins for Detecting Input-Validation Vulnerabilities

Aniqua Z. Baset
University of Utah
aniqua@cs.utah.edu

Tamara Denning
University of Utah
tdenning@cs.utah.edu

***Abstract*—Many vulnerabilities in products and systems could be avoided if better secure coding practices were in place. There exist a number of Integrated Development Environment (IDE) plugins which help developers check for security flaws while they code. In this work, we present a review of these plugins. We specifically focus on the plugins that detect input-validation-related vulnerabilities. We list salient features such as their supported IDEs, applicable languages and specific types of vulnerability checks. We believe this work synthesizes information useful for future research on IDE plugins for detecting input-validation-related vulnerabilities.**

I. Introduction

Many vulnerabilities in today's systems and applications result from common, well-documented coding errors. The most common form of such vulnerabilities arise when developers do not validate inputs from external entities (e.g., human user, computer or network component) before use, allowing adversaries to construct malicious inputs to compromise the system or application. It is possible to detect these types of vulnerabilities during software development.

Both static and dynamic analysis tools have been developed to detect security flaws in code (e.g., [1], [2], [3], [4]). These tools normally come with their own command-line or graphical interfaces to run analyses and display results. This requires developers to move back and forth between their coding environment (e.g., IDE) where they program, and the tool's interface, where they separately check for security problems. This overhead oftentimes contributes to lower adoption of security tools [5]. In recent years static analysis for security has become available via IDE plugins, providing a more seamless experience. These plugins allow developers to check security flaws in their code from within their IDE, since they present their results in the IDE like regular compiler errors. This in-situ security analysis and feedback can help developers detect flaws in the earlier stages of software development.

In this work we synthesize information on IDE plugins that provide security functionality. We specifically focus on plugins that provide support for input-validation-related vulnerabilities, namely: Improper input validation (CWE[1] 20), Command injection (CWE 77), OS Command injection (CWE 78), Cross-site scripting (CWE 79), SQL injection (CWE 89), LDAP injection (CWE 90), XML injection (CWE 91), Unsafe reflection (CWE 470), and XPath injection (CWE 643). We intentionally approach our data collection from the perspective of a developer, which we believe allows us to better understand the obstacles faced by a security-conscious developer. We find that there is a lack of information on these plugins about specific vulnerability checks and detection accuracy, which may contribute to lower adoption among developers. We believe that this paper lays the groundwork for future research in this area by synthesizing the information necessary to orient researchers choosing to tackle this underexplored space.

[1]Common Weakness Enumeration (CWE) is a listing of software weaknesses and vulnerability types [6].

II. IDE Plugins for Input Validation

We gathered security plugin information in four ways. First, we searched the plugin lists and marketplaces for four of the most prominent IDEs: Eclipse, IntelliJ IDEA, Visual Studio, and Netbeans IDEs. Second, we looked for plugins in forum discussions like StackExchange. Third, we checked lists of static security analysis tools (e.g., [29], [30]) to determine whether any of them have support for IDE integration. Fourth, we searched for security plugins developed in the academic literature. Finally, we checked the vulnerability documentation of each found plugin to determine whether it checks for input-validation-related vulnerabilities in code.

We list the available IDE security plugins in Table I. We exclude some IDE plugins from our list that do not present results within the IDE. For example, the Eclipse plugin for Coverity uploads the code to a server; once the server-side analysis is complete the result is presented via the developer's online account. In contrast, we *do* include Checkmarx CxSAST, Fortify, and Veracode: while the analysis is performed on a server, the results are presented in the IDE similar to the other listed plugins. We also exclude Cppcheclipse [31] from our list since it does not support the input-validation-related vulnerabilities that we are interested in for this work. However, we include some plugins (e.g., Codepro AnalytiX, SensioLabsInsight, SSVChecker) that do not have a full list of vulnerability checks publicly available, as they might have support for input vulnerabilities. We also exclude Contrast since the Eclipse plugin version of it has been discontinued [32].

Supported IDEs and Platforms. As evident from Table I, security plugins are available for most mainstream IDEs and

DOI 10.1109/SPW.2017.37

Table I: IDE plugins available for security checks

Plugin	IDE	Language and/or Platform	Availability	Source	Introduced	Last update
Android Lint [7]	AS, Eclipse	Java, XML, Android	Free	Open	—	—
ASIDE [8], [9]	Eclipse	Java, PHP	Free	Open	Feb'13	Sept'14
CodeDX* [10]	Eclipse, VS	Java, .NET, Android	Commercial	Closed	Jan'15	Feb/Mar'16
Codepro AnalytiX [11]	Eclipse	Java, JSP, XML	Free	—	Feb'05	Oct'10
Checkmarx CxSAST§ [12]	Eclipse, VS, IntelliJ	Java, .NET, Python, Ruby, C/C++, C#, JS	Commercial	Closed	—	—
ESVD [13], [14]	Eclipse	Java	Free	Closed	July14	Nov'16
Findbugs [15]	Eclipse, NB, IntelliJ, AS	Java, Android	Free	Open	—	—
Fortify [16]	Eclipse, VS	C/C++, Java, .NET, PHP, JS, Python	Commercial	Closed	—	Feb/Mar'17
FxCop [17]	VS	.NET	Free	Closed	—	—
Goanna Studio [18]	Eclipse, VS	C/C++	Commercial	Closed	—	—
Klocwork Insight‡ [19]	Eclipse, IntelliJ, VS	Java, C/C++, C#	Commercial	Closed	—	—
LAPSE+ [20], [21]	Eclipse	Java	Free	Open	Mar'11	Mar'11
SecureAssist [22]	Eclipse, VS, IntelliJ	Java, PHP, .NET	Commercial	—	—	—
SensioLabsInsight [23]	PHPStorm	PHP	Both	Closed	Oct'14	Jan'17
SonarLint [24]	Eclipse, VS, IntelliJ	Java, JS, PHP, .NET, Python	Free	Open	Oct'15	Feb'17
SSVChecker* [25], [26]	Eclipse	C/C++, Python, PHP	Free	Closed	May'10	Nov'16
Veracode [27]	Eclipse, VS, IntelliJ	Java, C/C++, C#, .NET, Python, Ruby, JS, PHP, Android	Commercial	Closed	—	Feb'17

VS = Visual Studio, IntelliJ = IntelliJ IDEA, NB = NetBeans, AS = Android Studio, JS = JavaScript
**Runs multiple analysis tools and present the combined results, §Previous version: CxSuite, ‡Previous version: Klocwork Solo*
ASIDE, ESVD, LAPSE+, and SSVChecker are academic. The standalone version of Findbugs is also from academic work [28].

languages/platforms, with the partial exception of Ruby and Android. We find only two plugins for Ruby (Checkmarx CxSAST, Veracode) and among all the plugins only Lint and FindBugs are available for Android Studio. We failed to find plugins for text-based editors such as Vim and Sublime.

Developer Experience. After installing a plugin, developers can initiate the security analysis by clicking on the compile (or similar) button. Much like regular compiler errors and warnings, these plugins display a list of identified flaws in an informational pane as well as indicate problematic code lines with markers in the code editor pane. This *just-in-place* reporting style is familiar to developers and allows them to make required changes in the code while viewing their result.

We have observed differences in quality and thoroughness in analysis reporting among plugins (as presented in Table II). Some plugins provide details in their report such as possible attacks, how the problem in code can lead to those attacks, examples of vulnerable and secure code, and risk ratings. Other plugins only mention the name of the possible attack or provide brief description of the attack. Besides pointing out the problem areas, some plugins also suggest possible mitigation strategies. However, in most cases these detailed reporting techniques serve to educate the developer on the identified attack and are not quick fixes specific to the code. To provide flexibility, some plugins also allow users to temporarily turn off particular warnings or to select/unselect specific vulnerability checks. Prior research suggests that such customization options make plugins more usable [5].

Documentation and Available Information. In Table III we include references to documentation on the supported vulnerability checks, categorized by the level of available information. As can be seen from the table, the full lists of vulnerability checks are not publicly available for several plugins, making it harder for interested developers to compare and choose which plugin to use. We find that none of the plugins provides information about detection accuracy[2].

Table II: Feedback styles

Plugin	Vuln. description	Mitigation	Other options
Android Lint	Short	—	Auto run when build, Select/unselect checks, Suppress warnings
ASIDE	Detailed	Quick fixes	—
CodeDX	Short	—	Suppress warnings
Codepro AnalytiX	Detailed	Quick fixes	—
Checkmarx	Detailed	—	Data flow viewer
ESVD	Just vuln. name	—	—
FxCop	Short	General	Suppress warnings
Findbugs	Detailed	—	—
Fortify	Detailed	General	—
Goanna Studio	Detailed	General	Select/unselect checks, Suppress warnings
Klocwork Insight	Detailed	General	—
LAPSE+	Just vuln. name	—	—
SecureAssist	Detailed	General	—
SensioLabsInsight	Short	—	—
SonarLint	Short	—	—
SSVChecker	Short	—	Suppress warnings
Veracode	Detailed	General	—

Note: all the plugins show risk rating for detected vulnerabilities except Android Lint and LAPSE+

[2]Contrast reports its results against OWASP benchmarks [33] but we excluded it from our study since the plugin version of it has been discontinued.

Table III: Vulnerability check documentation

Documentation type	Plugins
List of the checked vulnerability names or potential attack names, e.g., SQL injection, Cross-site scripting, etc.	ASIDE [9], ESVD [13], Klocwork Insight [34], LAPSE+ [20], Checkmarx‡ [35], SecureAssist‡ [36]
List of checked rules, i.e., the patterns that the plugins check for to determine potential vulnerabilities§	Android Lint [37], SonarLint* [38] (categorized by supported languages), SensioLabsInsight [39]‡, FxCop [40], Findbugs* [41], Goanna Studio* [42]
List of supported analysis tools that are used for the checks. No combined vulnerability checklist available	SSVChecker [26], CodeDX [43]
No list of checked vulnerabilities or rules is available	Codepro AnalytiX, Fortify, Veracode

‡*Full list is not available, only some examples*

**With mappings to CWE, CVE entries*

§*For example: Android Lint lists 'addJavascriptInterface called' as a rule to flag potential unsafe reflection. FxCop uses the 'CA2100: ReviewSqlQueriesForSecurityVulnerabilities' rule to flag possible SQL injection attacks when a method sets the IDbCommand.CommandText property using a string that is built from a string argument to the method.*

Table IV: Input-validation related vulnerability checks

Vulnerability checks	CWE	Android Lint	ASIDE	Checkmarx	ESVD	Findbugs	Fortify	FxCop	Goanna Studio	Klocwork Insight	LAPSE+	SecureAssist	SonarLint	Veracode
Improper input validation	20	-	✓	-	-	✓	-	-	-	-	-	-	-	-
Command injection	77	-	-	✓	✓	✓	-	-	✓	-	✓	-	-	-
OS Command Injection	78	-	-	✓	-	✓	-	-	✓	-	-	-	-	-
Cross-site Scripting	79	-	-	✓	✓	✓	✓	✓	-	✓	✓	-	-	-
SQL Injection	89	-	-	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	✓
LDAP injection	90	-	-	✓	✓	✓	-	-	-	-	✓	✓	-	-
XML injection	91	-	-	-	-	-	-	-	-	-	✓	-	-	-
Unsafe Reflection	470	✓	-	-	✓	-	-	-	-	-	-	-	-	-
XPath injection	643	-	-	✓	✓	✓	-	-	-	-	✓	✓	-	-

Vulnerability Checks. Table IV presents the input-validation-related vulnerabilities and corresponding plugins that support such checks. Some plugins are not present in Table IV because their vulnerability checking list is not publicly available (e.g., Codepro AnalytiX, SensioLabsInsight).

As evident from Table IV, most of the plugins check for the most common input validation vulnerabilities for web applications: SQL injection and cross-site scripting attacks. No plugin appears to check for all the 9 vulnerabilities that we were interested in. Only three plugins (ESVD, Findbugs, and LAPSE+) check for 6 or more of the vulnerabilities. Our analysis of the vulnerability coverage offered by plugins is based on their own descriptions; we did not run an independent verification. While there are disadvantages to not independently verifying the plugins, *the information that we are using is the same information that a developer or member of the public would have when deciding whether to use a plugin.*

Plugin Uptake. Among the plugin listings we encountered for different IDEs, only the Eclipse marketplace reports on number of installations. Table V shows a summary of Eclipse's install statistics sorted by rank (as collected on 3/30/2017).[3] It is to be noted that there are total 1285 ranked plugins in the Eclipse marketplace.

Table V: Installation statistics of Eclipse security plugins

Plugins	Findbugs	SonarLint	ESVD	SSVChecker	Codepro
Installs	443600	114551	1355	923	69
Rank	13	28	577	652	1038

As can be seen from Table V, Findbugs has very high installation numbers and is ranked #13 in the Eclipse marketplace in terms of installation numbers. We posit that this high popularity is due to the variety of features it offers beyond input-related vulnerability checking. Considering the installation numbers and their ranks in the marketplace, other plugins do not seem as popular as security researchers might hope.

Although the IDEs list available plugins under different categories, only Visual Studio has a security-specific category (primarily intended for code obfuscation and managing group code access in an organizational setting). As for other IDEs, the absence of any *security* category is not due to having too few categories (Eclipse=51, IntelliJ IDEA=50, Netbeans=24); they have more categories than that of Visual Studio (22).

III. Discussion

As evident from previous studies on security tools, it is cumbersome and demotivating for developers to evaluate different tools on their own [44]. This problem should also hold for the security plugins: not all plugins provide much detail about their checking capabilities (see Table III). It is important that more information be made available about these plugins (be that by manufacturer or independent researchers) such as: the accuracy of their vulnerability checks—what percentage of the detected vulnerabilities are true positives, what percentage of vulnerabilities are false negatives—incurred overheads, initial setup complexity, and similar. One place to start with collecting such statistics in a consistent manner would be to make use of existing benchmark suits (e.g, OWASP benchmark[4]).

As described earlier, different plugins take different approaches to feedback. It is inevitable that these varying styles will impact developers differently. Further research is needed on the effectiveness of different feedback styles

[3]Some of the plugins we found that support Eclipse are not available via the marketplace, and thus installation numbers are not available.

[4]The Open Web Application Security Project (OWASP) is a non-profit organization that works for the improvement of web application security. It has developed a "benchmark" project for evaluating the speed, coverage, and accuracy of tools that check web application-related vulnerabilities [33].

among different developer populations. It is also unclear if and how security feedback should be different from other forms of source code-related feedback.

IV. Conclusion

IDE plugins that check for input-validation vulnerabilities can help increase the security of code. Overall, there is a low adoption rate of security plugins. We have generally found a lack of information available about the checks performed by these plugins. In addition to more complete information, we would like for security benchmarking information to be made available for each plugin so that the developer and security communities at large can better evaluate such plugins.

References

[1] "Coverity: Static code analysis. https://www.synopsys.com/software-integrity/products/static-code-analysis.html#."

[2] "Cppcheck. http://cppcheck.sourceforge.net/."

[3] "Purifyplus. http://teamblue.unicomsi.com/products/purifyplus/."

[4] "Valgrind. http://valgrind.org/."

[5] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 2013, pp. 672–681.

[6] "Cwe. https://cwe.mitre.org/."

[7] "Lint. https://developer.android.com/studio/write/lint.html."

[8] "Aside. https://www.owasp.org/index.php/OWASP_ASIDE_Project."

[9] J. Xie, B. Chu, H. R. Lipford, and J. T. Melton, "Aside: Ide support for web application security," in *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 2011, pp. 267–276.

[10] "Codedx. http://codedx.com/ide-integration-helps-developers-adopt-application-security-testing-tools/."

[11] "Codeproanalytix. https://developers.google.com/java-dev-tools/codepro/doc/."

[12] "Checkmark cxsast. https://www.checkmarx.com/technology/static-code-analysis-sca/."

[13] "Esvd. https://marketplace.eclipse.org/content/early-security-vulnerability-detector-esvd."

[14] L. S. M. de Souza, "Early vulnerability detection for supporting secure programming," Master's thesis, Departamento de Informtica, Pontifcia Universidade Catlica do Rio de Janeiro, 2015. [Online]. Available: http://thecodemaster.net/wp-content/uploads/2015/06/early-vulnerability-detection-for-supporting-secure-programming.pdf

[15] "Findbugs. https://androidbycode.wordpress.com/2015/02/13/static-code-analysis-automation-using-findbugs-android-studio/."

[16] "Fortify. https://marketplace.eclipse.org/content/hpe-security-fortify-demand-plugin."

[17] "Fxcop. https://msdn.microsoft.com/en-us/library/bb429476(v=vs.80).aspx."

[18] "Goanna studio. https://marketplace.eclipse.org/content/goanna-studio-static-analysis-cc."

[19] "Klockwork. http://www.klocwork.com/products-services/klocwork/static-code-analysis."

[20] "Lapse+. https://code.google.com/p/lapse-plus/."

[21] P. M. Pérez, J. Filipiak, and J. M. Sierra, "Lapse+ static analysis security software: Vulnerabilities detection in java ee applications," in *Future Information Technology*. Springer, 2011, pp. 148–156.

[22] "Secureassist. https://www.cigital.com/resources/datasheets/secureassist-datasheet/."

[23] "Sensiolabsinsight. https://plugins.jetbrains.com/plugin/7589?pr=."

[24] "sonarlint. http://www.sonarlint.org/eclipse/index.html."

[25] "Ssv checker. https://marketplace.eclipse.org/content/ssvchecker."

[26] J. Dehlinger, Q. Feng, and L. Hu, "Ssvchecker: unifying static security vulnerability detection tools in an eclipse plug-in," in *Proceedings of the 2006 OOPSLA workshop on eclipse technology eXchange*. ACM, 2006, pp. 30–34.

[27] "Veracode. https://www.veracode.com/."

[28] "Findbugs. http://findbugs.sourceforge.net/."

[29] "List of tools for static code analysis. https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysi."

[30] "Owasp: Static code analysis. https://www.owasp.org/index.php/Static_Code_Analysis."

[31] "Cppcheclipse. https://marketplace.eclipse.org/content/cppcheclipse."

[32] "Contrast. https://marketplace.eclipse.org/content/contrast-eclipse."

[33] "Owasp benchmark project. https://www.owasp.org/index.php/Benchmark."

[34] "Klockwork: Application security. http://www.klocwork.com/products-services/klocwork/application-security."

[35] "Checkmarx: Vulnerability coverage. https://www.checkmarx.com/technology/vulnerability-coverage/."

[36] "Secureassist 3.0. https://codiscope.com/deeper-reporting-broader-compatibility-with-secureassist-3-0/."

[37] "Android lint checks. http://tools.android.com/tips/lint-checks."

[38] "Sonarlint: List of rules. http://www.sonarlint.org/intellij/rules/index.html#version=2.8."

[39] "Sensiolabsinsight: What we analyze. https://insight.sensiolabs.com/what-we-analyse."

[40] "Security warnings. https://msdn.microsoft.com/en-us/library/ms182296%28v=vs.140%29.aspx."

[41] "Find security bugs: Bug patterns. http://find-sec-bugs.github.io/bugs.htm."

[42] "Goanna 3.6.4 standards data sheet for cwe. http://archive.redlizards.com/docs/cwe-datasheet.pdf."

[43] "Supported application security testing tools and languages. http://codedx.com/supported-tools/."

[44] J. Witschey, S. Xiao, and E. Murphy-Hill, "Technical and personal factors influencing developers' adoption of security tools," in *Proceedings of the 2014 ACM Workshop on Security Information Workers*. ACM, 2014, pp. 23–26.