ISSN 2664-4150 (Print) & ISSN 2664-794X (Online)
South Asian Research Journal of Engineering and Technology
Abbreviated Key Title: South Asian Res J Eng Tech

| Volume-2 | Issue-6 | Nov-Dec -2020 | **DOI: 10.36346/sarjet.2020.v02i06.001**

Original Research Article

Towards Secure Software Engineering

L M Jayalath*, K A C Dharshana, R M T P Rathnayake

"Vijaya Niwasa", Ananda Mw, Kithulampitiya, Galle.
No: 40, Bodiraja Mawatha, Buwalikada, Kandy.
Paliyakotuwa, Hettipola

***Corresponding Author**
L M Jayalath

Article History
Received: 10.11.2020
Accepted: 22.11.2020
Published: 25.11.2020

Abstract: Software plays a major role in the present context. Therefore more and more software solutions are developed. These software solutions are hacked due to vulnerabilities available in the software. Most of these vulnerabilities are security-related. One of the key reasons behind this is the lack of security-related knowledge among the developers. Therefore it is important to assist them during the implementation stage. There are static analysis tools to assist them but these tools have a high concern on code quality and the architecture while having less attention for code security. Due to all these, it cost a lot during the maintenance phase to overcome the security-related issues. The suggested solution is a combination of three modules. Module one will rank the threats identified by the Microsoft Threat Modeling Tool in the design phase. Module two is an IntelliJ plugin which will assist the Java developers to maintain software security with respect to fifteen selected CERT guidelines. Module three will compare the design and the implementation while considering the inputs from the other two modules.

Keywords: Microsoft Threat Modeling Tool, STRIDE, threat ranking, CERT security guidelines, CWE, data flow diagram.

INTRODUCTION

Software solutions have positively impacted the living standard of the people. As a result of that more and more software solutions are developed. These software solutions are hacked by exploring the vulnerabilities or the loop holes available [10]. Most of the vulnerabilities are security related. Major reason behind this is the lack of security concerns during the software development life cycle. Most of the software developers do not have much knowledge about software security aspects. Hence security concerns are neglected during the implementation stage where most of the software security violations could be mitigated. As a consequence of all these it cost a lot during the maintenance phase to overcome the security issues [11]. Therefore it is really important to pay attention to the security aspect during early stages of the software development life cycle. There are static analysis tools to assist the developers during the implementation stage. But these tools pay high attention to code quality and architecture while having less concern about security aspects [7].

Secure Development Life Cycle (SDLC) is a model which is developed to focus on the security aspect at each and every phase of the Software Development Life Cycle [15]. According to this model there are some basic actions to be performed during each phase as illustrated in figure 1.

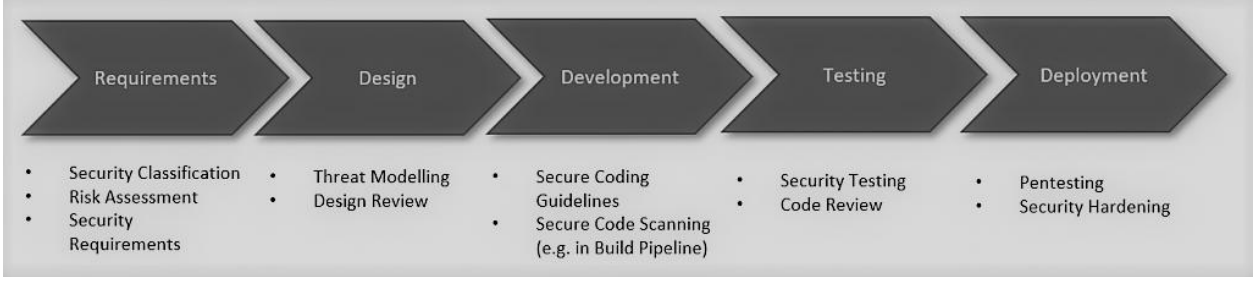

Fig-1: Secure Development Life Cycle (SDLC)

As the mistakes which happen during the design and the implementation stages result high degree of security vulnerabilities, this research is focusing on those two stages [22]. It is identified that there are several tools available in the market to SDLC approach during software design and development. Microsoft Threat Modeling Tool is one such tool which is used during the design stage. It is used to analyze data flow diagrams and will give a report containing possible security threats which could occur. Here the threats are categorized according to STRIDE model [23]. For a complex data flow diagram this tool gives a lot of possible threats. Practically it is not possible to consider all the threats there for it is really essential to rank the threats. With the ranking it is possible to prioritize the threats. Then it will reduce a high amount of possible threat by resolving the highest prioritized threats. In order to aid the software development stage there are static analysis tools such as SonarQube. These Static analysis tools are unable to catch the security violations in the code at real time.

Implemented solution will help to practically implement the SDLC for software design and development stages with three separate modules which are integrated together with the help of a common database. First module will rank the threats identified by the Microsoft Threat Modeling Tool. As a result of that the developer will get to know the highest prioritized threats that he should consider during the development stage in order to reduce the damage. Second module is a plugin which is designed to identify security guideline violations on the fly. And the third module will compare the results given by the two modules which represent two phases of the software development life cycle. This comparison will help to identify the phase (design or implementation) responsible for the identified security guideline violations with the second module. When considering about the industrial level development of software, each phase of the software development lifecycle is carried out by different teams of individuals. There for it is highly essential to compare the stages and to identify the responsible stage for the violations.

EXPERIMENTAL SECTION

Design and the implementation of the implemented solution will be discussed in three modules. First module is the threat ranking module and figure 2 demonstrates architecture of that module. Data flow diagrams are the input of this module. Only the web applications are being considered on this module. Microsoft Threat Modeling Tool helps to find the threats in the design phase of the Software projects. Data Flow Diagrams (DFD) are drawn through the Microsoft threat modeling tool. And this tool provides the threat report according to the drawn DFD. Threat report includes all the threats that can happen according to the drawn DFD. The threats are separated with respect to the interactions. And all the threats are categorized under specific STRIDE. Threat report is generated as an HTML page. Therefore Natural language processing is used to read the HTML page and to extract the threats. Mainly the threats are extracted in two ways. With respect to the interactions and under specific STRIDE. Mainly there are three modules in this project. These three modules are connected to the database. Extracted threats according to the interactions are sent to the database. That details are used as inputs to module three. By analyzing multiple threat reports for the web applications, identified 27 main different consequences that can happen according to the drawn data flow diagrams. Each type of web application was used for this analysis except static and animated web applications. And there can't be any threat which is out of the identified 27 threats. DREAD is a threat modeling and risk assessing methodology which is used to rate, compare and prioritize the severity of risk. By using the DREAD model, the risk rating can be done for a given threat by asking the following questions:

- Damage potential: How great is the damage if the vulnerability is exploited?
- Reproducibility: How easy is it to reproduce the attack?
- Exploitability: How easy is it to launch an attack?
- Affected users: As a rough percentage, how many users are affected?
- Discoverability: How easy is it to find the vulnerability?

There is a scoring table that used to calculate the severity of the risks in the DREAD model. After asking the above questions, count the values (1–3) for a given threat. The result can fall in the range of 5–15. Then it can treat threats with overall ratings of 12–15 as High risk, 8–11 as Medium risk, and 5–7 as Low risk. This DREAD model is not

a technical threat ranking method. It is a manual approach. lecturers, professors, and specialists in this field rank the threats by answering the aforementioned questions according to their experience and knowledge. There is a certain scoring system that is used to ranking the threats. Otherwise, each individual can rate the threats in their own way. If the individuals ranking the threats their own way, the variance of the rating values can differ for the same threat. Here the identified 27 major threats have been ranked according to High, Medium and Lower rates. Therefore multiple research papers are used to rank the threats by DREAD model. Keywords have been created to identify the main 27 threats separately. Identified keywords are applied to the extracted threats. According to the keywords, the threats are ranked as High, Medium and Low. And finally, identify the highest priority threats.

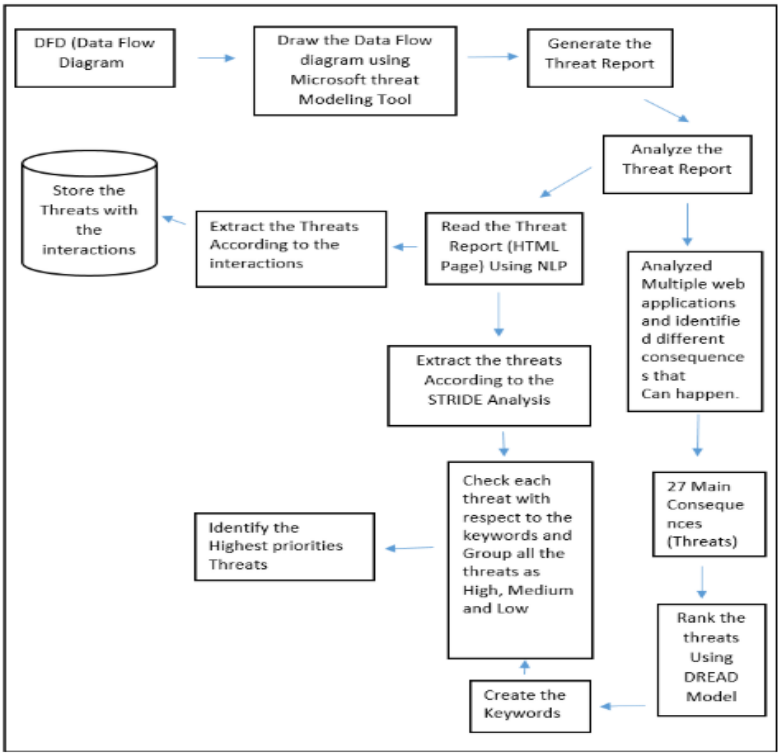

Fig-2: Architecture of Module 1

For the implementation of ranking the threats, first it is needed to draw the DFD (Data Flow Diagram) through the Microsoft threat modeling tool. Therefore different data flow diagrams were drawn to analyze the threat reports. After that threat reports were generated for each Data flow diagrams. Then threat report has been read by using natural language processing (NLP).The Microsoft threat modeling tool provides the html type threat report. Therefore python libraries had to be used to read the html type threat report. By reading the threat report, all the threats were extracted from the HTML page. And Spyder IDE was used to extract the threats. NLTK, re (Regular expression), urllib and beautiful soup python libraries were used to extract the threats from the threat report. Threats are extracted in two ways from the threat reports. The extracted threats under interactions are sent to the database. These details are used in module three. The extracted threats as STRIDE are used to rank the Threats. MySQL is used as a database to store the data. And interface have been created to insert the threat report for any user. That user can insert the file location of the threat report by using the save file dialog box. After inserting the threat report, the report is read by NLP and extracted all the threats from the report. And each threat is checked with the keywords and separated as High, Medium and Low. After clicking the rank button user can view the threats ranked according to high, medium and Low.

With module 2, it is expected to develop a plugin to IntelliJ IDE and the source code is read from the IntelliJ plugin in real time while the developer is coding. For reading the source code, Java Parser was selected as the language support needed is Java and the accuracy is high instead of creating a custom parser. The source code is passed through the Java Parser library and it is converted into an Abstract Syntax Tree (AST) using Java Parser. A logic is written for each SEI CERT secure coding violations out of selected 15, in order to identify the relevant violations in the code. The AST from the Java Parser is read and relevant data structure, class, method or code fragment relevant to the given logic is read and compared with each other. If there exists an equivalence between them, a security violation is identified and it is visually shown to the developer by highlighting the code syntax. At the end of the project completion, the final results containing all the violations of the project is stored in a MySQL database to be used for the Module 3 in order to do further tasks in the project. Above mentioned steps are clearly demonstrated with figure 3.

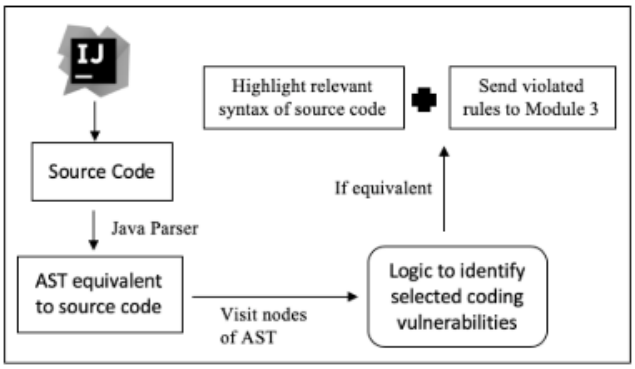

Fig-3: Architecture of Module 2

To assist the developer, IntelliJ plugin was built by using Gradle approach and Java Parser was used to parse the source code for identifying security violations. The violations of the code are identified mainly by using method level, class level and package level violation detectors based on the source code fragment which leads to the violation. The source code which is parsed by the Java Parser is converted to AST and is read by "ClassLevelCode.java", "MethodLevelCode.java" and "PackageLevelCode.java" classes which are implemented in the plugin project. From these classes, all the classes, methods, data structures and code fragments of the source code are categorized and can be accessed at any time as they are stored in separate data structures such as Array Lists and Hash Maps to be used when necessary. Logics to identify secure coding violations are written as separate methods inside "ClassLevelViolation.java", "MethodLevelViolation.java" and "PackageLevelViolation.java" classes. These classes use data from the data structures from the previous steps to check the equality with the logic and then the security violations are detected. The "RealtimeParser.java" class of the framework is used to capture source code fragments the user types in IntelliJ IDE in an on the fly manner. At each time the user types a source code, an AST is generated by the Java Parser library and the relevant Java Parser methods are used to traverse the AST with the support of the Visitor design pattern found in the Java Parser library. And the conditions inside "RealtimeParser.java" checks for violations by checking the logics written at "ClassLevelViolation.java", "MethodLevelViolation.java" and "PackageLevelViolation.java" classes. Finally, the syntax of the identified violations is highlighted and shown visually to the developer by using the "Syntax Highlight" library in java IntelliJ. At the end of the project, the developer is allowed to save violated rules to a MySQL database to be used at module 3 after pressing a button.

Module three will compare the results given by the two modules which represent two phases of the software development life cycle. And the steps used to compare the two stages are demonstrated with figure 4 and each step is discussed in detail.

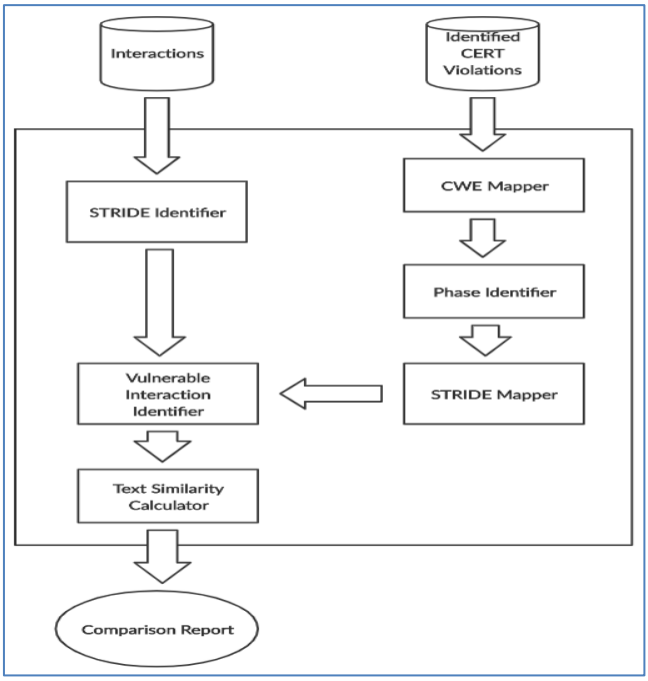

Fig-4: Architecture of Module 3

Interactions - They are the interactions identified by the Microsoft Threat Modeling Tool (TMT). These interactions contain relevant STRIDEs associated with each interaction. And there may be a number of STRIDEs for a specific interaction.

CWE Mapper - There was no approach to map CERT with the interactions directly. In order to perform that CWE mapper is used. Here the Identified CERT security guidelines are mapped with relevant CWE (Common Weakness Enumeration). There may be more than one matching CWE for one specific CERT.

Phase Identifier - This will identify whether the design phase or the implementation phase is responsible for the identified CWEs by the CWE Mapper.

STRIDE Mapper - This will map the CWEs with the associated STRIDEs as mentioned in the table. CWE site provides the associated security controls for each CWE and the security controls are mapped to respective matching STRIDE according to table.

STRIDE Identifier - This will identify the specific STRIDEs associated with the interactions identified in the report generated by the Microsoft Threat Modeling Tool.

Vulnerable Interaction Identifier - This will take inputs from both STRIDE mapper and the STRIDE identifier. From the STRIDE mapper it has a set of CWEs with the associated STRIDEs for each and from the STRIDE identifies it has a set of interactions with the associated STRIDEs for each.

This module is going to find whether there is a matching interaction for the available CWEs. In order to perform that it will consider the CWEs separately in which the interactions where the STRIDEs of the CWEs are a subset of the STRIDEs of the interactions are identified. If there is only one matching interaction for a specific CWE, then it will be identified as the interaction which is responsible for the considered CERT security guideline violation. Else if there are more than one matching interaction for a specific CWE then it will direct to the Text Similarity Calculator.

Text Similarity Calculator - This is used to find the highest matching interaction for a given CWE when there are more than one matching interaction is identified by the Vulnerable Interaction Identifier. Here a text similarity calculation is done between the descriptions provided for the CWE with the descriptions provided for the STRIDEs for the considered interaction. Interaction having the highest similarity value will be considered as the interaction responsible for the identified CERT security guideline violation. Cosine Similarity algorithm is used to calculate the similarity.

Comparison Report - This will generate a comparison report as the final output of this module. This report will contain details such as Identified CERT guideline violations, CWEs associated with each CERT, Classes and the lines where the CERT guideline violation has occurred, Responsible phase (design, implementation) and Interaction which is responsible for the CERT guideline violation if it is identified as a design fault.

In order to perform the comparison it requires data from Module one and Module two. Data from the two modules are stored in a hosted MySQL database. The database is normalized to third normal form in order to reduce the data redundancy. An interface is provided to the user where it gives the opportunity to select the required threat report and the vulnerability report to be compared. Used JFrames to design the interface. Further it facilitates the user to specify a location where to save the generated report. Then the required data is passed to the back end which is developed using Java. Back end contains the comparison logic. After completing the comparison it will generate a report.

RESULTS AND DISCUSSION

Most of the times security impact of the code is considered during the latter stages of the Software Development Lifecycle (SDLC). Due to this it cost a lot to overcome those issues. Suggested and implemented approach which is discussed with this paper will try to bring software security concern to the early stages of the SDLC. In order to achieve that aim, suggested solution is implemented in three modules where module 1 addresses the design phase, module 2 addresses the implementation phase and module three compares the two stages to find out which phase is responsible for an identified secure coding guideline violation during the implementation stage.

Implemented solution was evaluated as three separate modules. First module was evaluated using a study based evaluation. For that two research papers which used DREAD model to rank the threads were used. With the use of the Microsoft Thread Modeling Tool two separate thread reports were generated for the considered data flow diagrams in the selected papers. Then the thread reports were used as the input for first module and ranked the threads. Finally the ranked threads were evaluated against the results obtained in the papers. Accuracy and precision of first module is shown in figure 5.

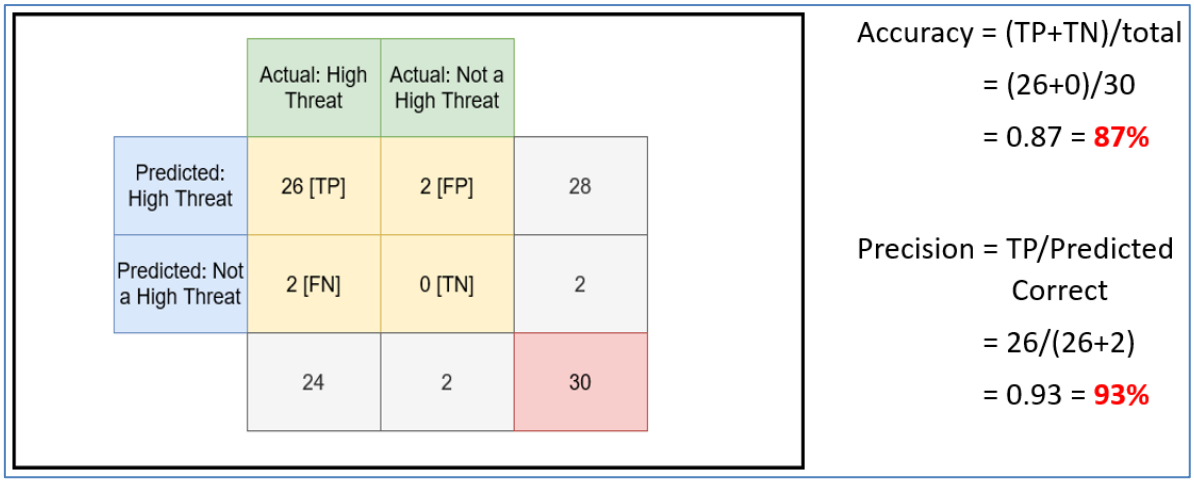

Fig-5: Evaluation Results of Module 1

At the moment this module only considers web applications. As a future development it is required to consider other types of applications.

Module two was evaluated using SEI CERT guidelines' sample code base evaluation. This was conducted to check the expected accuracy of the implemented plugin. SEI CERT website provides both compliant and non-compliant code samples. Non – compliant code was used to detect whether the plugin detects vulnerabilities and compliant code was used to check whether the plugin ignores the corrected code. Table 1 lists the results of the devaluation.

Table-1: Evaluation Results of Module 2

SEI CERT Rule Number	Detection status of Non-compliant code example(s)	Detection status of compliant code example(s)
MSC00J	Detected	Not Detected
SER01J	Detected	Not Detected
SER05J	Detected	Not Detected
NUM02J	Detected	Not Detected
SEC00J	Detected	Not Detected
FIO13J	Detected	Not Detected
LCK05J	Detected	Not Detected
ERR04J	Detected	Not Detected
ERR07J	Detected	Not Detected
MET09J	Detected	Not Detected
OBJ01J	Detected	Not Detected
OBJ05J	Detected	Not Detected
OBJ08J	Detected	Not Detected
OBJ09J	Detected	Not Detected
OBJ10J	Detected	Not Detected

In order to further evaluate module two, GitHub project based evaluation was done. For this analysis a set of open source Java projects were selected and those details are shown in table 2. Then selected code bases were scanned using the implemented plugin to detect secure coding guideline violations if available. And the same code bases were also analyzed against some popular secure coding tools used in the industry (SonarQube, Sonar Lint, Find Bugs, and Check Style). Results obtained are shown in table 3 and evaluation of the results is shown in table 4.

Table-2: Details about the Selected GitHub Projects

Github Project Name	Contributors	Number of lines in project
Java-socket-tcp-sample-master	Marsel Sampe Asang	10430
demo-serialize-optional	Nicolai Parlog	807
Arthas-master	85 Contributors	80825
Qulice-master	34 Contributors	26522
Strongbox	94 Contributors	150749
Teammates master	422 Contributors	889727
junrar-master	10 Contributors	14264
Java-JavaFx-Swing-Projects-Desktop-Application-GUI-Software-master	Soumyadip Chowdhury	12217
Minicopier-master	Adrian Courrèges	4285
Ant-antlibs-props-master	3 Contributors	3021

46

Table-3: GitHub Evaluation Details

Rule	Implemented Plugin	SonarQube	Sonar Lint	Find Bugs	Check Style
MSC00J	Detected	Not Detected	Not Detected	Not Detected	Not Detected
SER01J	Detected	Not Detected	Not Detected	Not Detected	Not Detected
SER05J	Detected	Not Detected	Not Detected	Not Detected	Not Detected
NUM02J	Detected	Detected	Detected	Not Detected	Not Detected
SEC00J	Detected	Not Detected	Not Detected	Not Detected	Not Detected
FIO13J	Detected	Not Detected	Not Detected	Not Detected	Not Detected
LCK05J	Detected	Not Detected	Detected	Not Detected	Not Detected
ERR04J	Detected	Detected	Detected	Not Detected	Not Detected
ERR07J	Detected	Detected	Detected	Not Detected	Not Detected
MET09J	Detected	Detected	Detected	Not Detected	Detected
OBJ01J	Detected	Not Detected	Detected	Not Detected	Not Detected
OBJ05J	Detected	Not Detected	Not Detected	Not Detected	Not Detected
OBJ08J	Detected	Not Detected	Not Detected	Not Detected	Not Detected
OBJ09J	Detected	Not Detected	Not Detected	Not Detected	Not Detected
OBJ10J	Detected	Detected	Detected	Not Detected	Detected

Table-4: GitHub Project Evaluation Results for Module 2

Selected Tool	Number of rules detected out of selected 15	Able to detect vulnerabilities at real-time?
Implemented Plugin	15	Yes
SonarQube	5	No
Sonar Lint	7	Yes
Find Bugs	0	No
Check Style	2	No

Hence the plugin identifies the vulnerabilities correctly for the selected fifteen rules; the number of rules should be increased in future in order to improve the commercial value of the solution.

GitHub project based evaluation was carried out for module three. For that a Java project was selected and a proper data flow diagram (DFD) for that specific project was designed with the help of an industry specialist. Then a thread report was generated for that DFD. And with the help of the plugin, identified the available CERT security guideline violations. With the above two inputs for module three generated the comparison report. Finally evaluated the comparison report against the selected project and the respective data flow diagram with the assistance of a domain specialist. Figure 10 shows the results obtained for phase identification (whether the fault is design related or implementation related) and figure 11 shows the results for interaction identification.

Table-5: Phase Identification Results for Module 3

CERT	Associated CWE	Domain Specialist Idea	Output given by the Comparison report
LCK05-J	CWE-820	Design Fault	Design Fault
ERR07-J	CWE-397	Design Fault	Design Fault
OBJ10-J	CWE-493	Implementation Fault	Implementation Fault
	CWE-500	Implementation Fault	Implementation Fault
OBJ01-J	CWE-766	Design Fault	Design Fault
NUM02-J	CWE-369	Implementation Fault	Implementation Fault
SEC00-J	CWE-266	Implementation Fault	Implementation Fault
	CWE-272	Implementation Fault	Implementation Fault
SER05-J	CWE-499	Implementation Fault	Implementation Fault

Accuracy of Phase Identification = 9/9 * 100 = 100%

CERT	CWE	Matching Interaction Identified by the Evaluator	Matching Interaction Identified by Comparison Module
LCK05-J	CWE-820	Book Details	Book Details
ERR07-J	CWE-397	Register User Data (Most probably)	Book Details
OBJ01-J	CWE-766	Book Details (Except one class)	Book Details

Accuracy of Interaction Identification = (2/3) * 100 = 66.7%

Fig-6: Interaction Identification Results

With the above evaluation it is clear that interaction identification section should be improved. In order to achieve that inputs for the cosign similarity calculation should be improved. Therefor more improved natural language processing techniques should be used in future.

CONCLUSION

Static analysis tools available in the market do not have much concern about the software security. With the implemented solution it will make the life of the developers easy by assisting them during the implementation about the secure coding violations with respect to with respect to fifteen selected CERT guidelines. Legacy systems may have several security related issues but it is really difficult to find where the issue is and what the priority of that issue is. With the implemented solution we will consider legacy systems with an available code base. Then solution will consider the available DFD and rank the threads according to a priority order. Hence it will provide a good understanding on what are the threats to be solved first. With the comparison between the implemented code and the DFD, it will allow to find which phase is responsible for violating the secure coding guidelines.

ACKNOWLEDGEMENT

We would like to extend our heartfelt gratitude to the following people who have helped and guided us in numerous ways to make the project a success. Our supervisors Mr. C.P.Wijesiriwardane and Dr. I.N.Manawadu, for the immense assistance and guidance they have always given us and motivating us towards achieving the aim of the project. All the academic staff at the Faculty of Information Technology of University of Moratuwa, who have

given us knowledge and guidance throughout our undergraduate years. All the non-academic staff of Faculty of Information Technology of University of Moratuwa, members of our families, our friends and batch mates who have supported us in numerous ways.

REFERENCES

1. Baltes, S., Schmitz, P., & Diehl, S. (2014, November). Linking sketches and diagrams to source code artifacts. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 743-746).
2. Boehm, B. W. (1991). Software risk management: principles and practices. *IEEE software*, *8*(1), 32-41.
3. Bruneliere, H., Cabot, J., Jouault, F., & Madiot, F. (2010, September). MoDisco: a generic and extensible framework for model driven reverse engineering. In *Proceedings of the IEEE/ACM international conference on automated software engineering* (pp. 173-174).
4. Daud, M. I. (2010, March). Secure software development model: A guide for secure software life cycle. In *Proceedings of the international MultiConference of Engineers and Computer Scientists* (Vol. 1, pp. 17-19).
5. Do, L. N. Q., Ali, K., Livshits, B., Bodden, E., Smith, J., & Murphy-Hill, E. (2017, May). Cheetah: just-in-time taint analysis for android apps. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)* (pp. 39-42). IEEE.
6. Favre, J. M., Lämmel, R., Leinberger, M., Schmorleiz, T., & Varanovich, A. (2012, October). Linking documentation and source code in a software chrestomathy. In *2012 19th Working Conference on Reverse Engineering* (pp. 335-344). IEEE.
7. Ferenc, R., Langó, L., Siket, I., Gyimóthy, T., & Bakota, T. (2014, September). Source meter sonar qube plug-in. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation* (pp. 77-82). IEEE.
8. Fujdiak, R., Mlynek, P., Mrnustik, P., Barabas, M., Blazek, P., Borcik, F., & Misurec, J. (2019, June). Managing the secure software development. In *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (pp. 1-4). IEEE.
9. Gomes, I., Morgado, P., Gomes, T., & Moreira, R. (2009). An overview on the static code analysis approach in software development. *Faculdade de Engenharia da Universidade do Porto, Portugal*.
10. Guan, H., Chen, W. R., Li, H., & Wang, J. (2011). STRIDE–Based Risk Assessment for Web Application. In *Applied Mechanics and Materials* (Vol. 58, pp. 1323-1328). Trans Tech Publications Ltd.
11. Khoo, Y. P., Foster, J. S., Hicks, M., & Sazawal, V. (2008, November). Path projection for user-centered static analysis tools. In *Proceedings of the 8th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering* (pp. 57-63).
12. Meier, J. D. (2003). *Improving web application security: threats and countermeasures*. Microsoft press.
13. Piessens, F. (2002, November). Taxonomy of causes of software vulnerabilities in internet software. In *Supplementary Proceedings of the 13th International Symposium on Software Reliability Engineering* (pp. 47-52). IEEE Computer Society Press, Los Alamitos, CA.
14. Prähofer, H., Angerer, F., Ramler, R., & Grillenberger, F. (2016). Static code analysis of IEC 61131-3 programs: Comprehensive tool support and experiences from large-scale industrial application. *IEEE Transactions on Industrial Informatics*, *13*(1), 37-47.
15. Ragunath, P. K., Velmourougan, S., Davachelvan, P., Kayalvizhi, S., & Ravimohan, R. (2010). Evolving a new model (SDLC Model-2010) for software development life cycle (SDLC). *International Journal of Computer Science and Network Security*, *10*(1), 112-119.
16. Rao, K. R. M., & Pant, D. (2010). A threat risk modeling framework for Geospatial Weather Information System (GWIS): a DREAD based study. *international Journal of Advanced Computer Science and Applications*, *1*(3).
17. Wichers, D. (2013). Owasp top-10 2013. *OWASP Foundation, February*.
18. Rao, K. R. M., & Pant, D. (2010). Security risk assessment of geospatial weather information system (gwis): An owasp based approach. *International Journal of Computer Science and Information Security*, *8*(5), 208-218.
19. Venkataraman, S., & Harrison, W. (2005). Prioritization of threats using the k/m algebra. In *Proceedings of Workshop on Software Security Assurance Tools, Techniques, and Metrics* (pp. 90-95).
20. Caulkins, J., Hough, E. D., Mead, N. R., & Osman, H. (2007). Optimizing investments in security countermeasures: A practical tool for fixed budgets. *IEEE Security & Privacy*, *5*(5), 57-60.
21. Zenah, N. H. Z., & Abd Aziz, N. (2011, December). Secure coding in software development. In *2011 Malaysian Conference in Software Engineering* (pp. 458-464). IEEE.
22. McGraw, G. (2004). Software security. *IEEE Security & Privacy*, *2*(2), 80-83.
23. McRee, R. (2014). Microsoft threat modeling tool 2014: identify & mitigate. *ISSA Journal*, *39*, 42.