International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 11, No.1 (Jan-2022)

http://dx.doi.org/10.12785/ijcds/110156

Software Security Validation through Regular Expressions

Omar Abahussain[1], Mustafa Hammad[1] and Fawzi Albalooshi[1]

[1] *Department of Computer Science University of Bahrain, Sakhir, Bahrain*

Received 10 April. 2020, Revised 22 June. 2020, Accepted 31 July. 2020, Published 31 Jan. 2022

Abstract: In modern society, software security has become an essential part of most software systems. As nowadays, new systems roll out more than ever, cybercriminals and unethical hackers tend to target those new systems to abuse and exploit its vulnerability to achieve a specific goal regardless of the consequences. Thus, validating software security is a challenging task and of crucial importance. The paper aims to find an optimal logical approach to test and validate software security through static analysis using regular expressions to optimize and secure the source code of the software.

Keywords: Software Security Testing, Static Analysis, Regular Expressions, Software Static Analysis Tool Prototype, Source Code Optimization

1. INTRODUCTION

In the field of software engineering, there is a set of analytical and technical steps to be followed. In general, the steps are all requirement: gathering, analyzing, designing, implementing, and lastly and not least testing the implementation. Testing consists of validations and verifications alongside the tests. Testing is significant as it makes sure the developed systems are fully functional, secure, and reliable as quality assurance. Finally, releasing and maintaining the software in the deployed system. These steps may be the main concerns of every developer and engineer, but security testing is very troublesome. Security is part of the software's non-functional requirements, a quality attribute, and a constraint in an indirect way.[1]

Nowadays, the majority of businesses are moving towards information systems as technology advances in that sector. Information systems have become a key to their business process. Thus, security has become of very high and significant importance to protect data and information from different types of security risks and abuse related to Information Communication Technology (ICT). Unfortunately, the security attribute in the Software Development Life Cycle (SDLC) is not considered as of any importance to the majority of the stakeholders. Software engineers as well think security may seem to be a waste of system resources depending on the type of developed software.

According to Lars Backman [2], a study conducted to assess where and how the issue persists within the stakeholders themselves through factors such as Insufficient knowledge, misplaced trust, and inadequate testing policies. Another Study by Agata McCormac et al. [3] was done to check the score of information security based on age, gender, personality, and risk-taking factor. The study yielded that age and gender have no impact, but instead, it was the personality and risk-taking factor that affected. Moreover, the importance of the software security code is conducting a cybersecurity awareness campaign study by Maria Bada et al. [4] that aims to spread awareness and understand where the flaws are at and fix it. Alongside them are Affan Yasin et al. [5], which aimed to spread the awareness of software security through a serious game that describes what would happen if security is not taken into consideration seriously. Thus, the security attribute is of very high importance as it focuses on the system's ability to secure and protect the data from being tempered, stolen, or abused.

Validation, verification, and testing of software security are incredibly challenging for software engineers. There are many ways to do, but each does not cover all the angles. Thus, engineers perform multiple batch operations to validate, verify, and test software security. Engineers should always keep the security of a system up-to-date whenever possible through patches and updates to keep the software secure. As it seems, Jiantao Pan [6] has written about the importance of the validation, verification, and testing of the software security phase in

E-mail address: 20110708@stu.uob.edu.bh, mhammad@uob.edu.bh, falblooshi@uob.edu.bh

the SDLC despite being a trade-off between budget, time, and quality. This phase defines the reliability of the software and its quality according to the results of all the tests.

This paper extends the work done in [7]. The paper will start with a literature review to get an idea of what has been done so far. Afterward, the paper will go through method and approach as the paper reviews current Validation, verification, and testing tools to obtain a general idea of what is available and to propose a unique approach. Next, the paper goes through a case study and its analysis of the study results. At last, the limitations and conclusion.

2. LITERATURE REVIEW

There are plenty of research papers and patents on the topic of software testing, validation, and verification. However, the papers about testing, verification, and validation in software security are few and hard to be found. In order to get such papers, the search should be going further in-depth and more specific as in taking each of the three words (testing, verification, validation) alongside the keyword "software security". The paper has a collection of what should cover different angles of Software Security Verification, Validation, and Testing (VVT).

A. Patents

Worldwide Cisco Technology Inc [8] inventors Jason Young et al. focused on the verification of the source code, the concept of analyzing a specific declared secret type variable, and find how that variable is configured and how it will move along other variables. Such verification and analysis will ensure sensitive data will not leak out from the source code as the secret variable moves to other variables without consideration of its value. It will create an exception in the source code verification process as the source code lacks secure handling of data inside the source code.

As for the inventor Matthew Allan Newman from Newman Infinite Inc [9], focused on different aspects of securing of JavaScript and source code from systems, related methods to other means. The configuration of everything can be done on the client machine to determine whether a debugging console is active or not and deny access to the JavaScript and source code if it is active. Additionally, the client device may request access to the JavaScript and source code. Thus, the determination of the request if it from a trusted referrer or not is of high importance and denies access if the request is untrusted and grant it when it is trusted. Moreover, Shape Security Inc inventors Sergey Shekyan et al. [10] have also focused on securing the source code of web applications by attempting to conduct vulnerability tests and exploit it by mixing different attacks against the web system content.

B. Papers

According to Brad Arkin et al. [11], security is considered as a severe problem. Simply because of the majority of the security defects and weaknesses are not related to the security functionality, but rather an intentional misuse of the software itself. Security testers should dig deep into the software for any security risks to understand how the system behaves when being attacked. One of the methods to test software security is through a penetration test where the stakeholder subjects his software to penetration testing as part of the final acceptance regimen. However, they stated the main major limitation of this approach is that it almost always represents very little and very late to involve security implementation/enhancements at the end of the software development cycle.

As for David Gilliam et al. [12], proposed a security assessment tool that assesses the code and finds security risks and vulnerabilities. The idea is to set up a database that consists of these risks. The tool then when used to assess another new software, it will start with what exists in the database then go over new things. Once that is done, the process of software security verification is complete.

Moving on to Bruce Potter and Gary McGraw [13], stating that software security is all about how to make the software behave correctly if there is any malicious attack on it. Regardless of the software random real-time unexpected failures occur unintentional misuse. Ironically, standard software testing is mainly concerned with what happens upon software failure irrespective of the user's intent. Software safety and software security are different in terms of the presence of an intelligent rival that wishes to break the system. Software safety is how the system behaves when misused, and software security is how the software data is secure. Their method was using a risk-based approach to help solve every security concern while the software is still in production through software security testing professionals.

An analytical research study was conducted by Ganesan Deepa and Santhi Thilagam [14], which aims to understand the approaches and challenges of securing web applications from different security threats and attacks from injections to logic vulnerabilities. Nowadays, we move towards web applications to do our daily activities; a single flaw would allow an attacker to gain access to sensitive information or cause harm to others. The study was conducted on an identified recently published articles from different well-known digital libraries. A total of 86 studies are selected and were divided into three classifications which are: 35 articles related to XSS, 34 articles related to logic flaws, and 17 articles related to SQLI. However, the conclusion is that there is no single solution to eliminate or reduce all the flaws and that more

research is required in the field of fixing an application's source code flaws.

Gu Tian-yang et al. [15] have researched different aspects of software security testing. Going through different types of software security tests with definitions and what it would cover as a finding. The outcome of the research was of a high significance in knowing what methods and tools to use for proper testing to cover all angles.

In [16], Peter Gilbert et al. proposed an application inspector that analyze the software and generate reports of possible security and privacy violations. The way how application inspector work is by executing the software and let the application inspector monitor and log how the data flow alongside the type of data flowing. Although their paper speaks of protecting mobile device users from being abused by improper use of users' data, the approach can be used as well to expose software security flaws and to be fixed to prevent any system abuse. That will validate the software security and test for any flaws.

Durability perspective were the thoughts of Rajeev Kumar et al. [17]. As long the system is durable, it is more likely to be secure. Durability is an attribute of security. Thus, testing durability from different angles would also test the security of the system itself. Not to mention, if the system is durable enough, security would be the second line of defense that would be used only if durability breaks. That shows the importance of system durability in giving the system a second line of defense. The tests done are basically on the durability of different aspects of security, such as durability with integrity, confidentiality, availability, etc.

Lastly, Zeineb Zhioua et al. [18] stated that it is a difficult task to build secure software. Furthermore, it gets more complicated if the stakeholders ask for security requirements. In order to make sure things go in the correct direction, static code analysis was suggested to capture any security vulnerability. Static code analysis is a manual trace of how data would be going and whether it is secured or matches the stakeholders' security requirements. Thus, it covers validation and verification alongside testing.

Software security VVT is of high importance that should be considered to prevent any cybercriminal or abusive user from abusing the system. From all of the papers reviewed, static code analysis seems to be very wide and well covering different angles of the software security while ignoring other software factors. However, mixing between different criteria of the reviewed papers would yield excellent results. A system that is very stable, error-handled, and durable is considered more secure than an unstable unhandled unendurable system.

3. METHODS AND APPROACH

A. *Current tools*

Nowadays, there are different tools available online that do static code analysis alongside code quality checking [19], rather than listing them all, key features are used to eliminate redundancy. The key features alongside their description will assist in understanding what the tools can do in Table I.

TABLE I. SECURITY APPROACHES TO SECURE DATA TRANSMISSION

Key features	Description
Run-time and logic errors:	the tool analyzes and assesses potential bugs before program execution.
Mathematical and logic verification	verifies the code and proves the absence of overflow, divide-by-zero, out-of-bounds array access, and specific other run-time errors.
Size and complexity	measure the files and code size and complexity.
Quality assessment tool	mainly for design assessment, supporting detection of implementation and design smells, computation of various code quality metrics, and trend analysis.
Security code analyzer	Analyze and Detects various security vulnerability patterns: SQLi, XSS, CSRF, XXE, Open Redirect, etc.
Real-time secure scan	real-time secure code analysis for common vulnerabilities.
Security Guard	the main focus is on the identification of potential vulnerabilities such as SQL injection, cross-site scripting (XSS), CSRF, cryptography weaknesses, hardcoded passwords, etc.
Dead code detection	finds and locates the unused code and unnecessary lines.
Wrong definition scan	detects any misspelled words that cause an error.
Crypto-related	checks for any incorrect uses of cryptographic APIs.
Google JavaScript Style Guide	it is precisely for JavaScript language, ensures the code is following the guidelines as well automatically fixes common errors.
Code cleaner	cleans code and maintains its consistency.
Dependency and complexity	find all dependencies and generates complexity reports.
Software Layers	focuses on finding dependencies between layers and enforces correct rules.
Bug detection	finds and fixes any bugs.
Combined security scan	a combination of Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), Interactive Application Security Testing (IAST) security scans [20, 21].

As mentioned, there are many tools online that target different languages rather than multiple, having a few key features rather than many. Combining various key features would yield better results as using multiple tools may not be suitable as using one tool.

B. *The method*

After examining the tools and their key features, it seems the majority of the tools do not have quality metrics

or code cleaning nor rules and guidelines. Instead, the tool checks and assesses the software for security vulnerabilities. The written code should have some standards to be secure then checked for risks, as writing the code properly should yield proper secure and risk-free software. Writing a proper code falls under the category of Quality Metrics (QM) [22]. QM helps in making more stable software by eliminating/reducing bugs and unwanted code, aside from reducing the number of lines. The proposed tool has a QM analysis and applies changes to improve the code.

The proposed tool aims to cover four criteria. The tested system should be durable, safe, stable, and secure. The proposed tool was developed using AngularJS, AngularMaterial with HTML5; it is designed as a prototype and is accessible online [23]. The tool currently inspects JavaScript source code and can be expanded to include other languages as future works, as the tool is very light and flexible. It is good to have such a tool online that requires no installation and aids in generating the desired outcome.

The tool examines the source code for any logs and eliminates them. Logging is mainly used for debugging and maybe misused. It is terrible to leave any consoles and loggers active as it may exploit and tell about the system structure to be specific when the log is coming from an ajax call, GET or POST method with a response of success or error. Aside from that, the system might break. As a console log may not be supported from a browser to another [24,25].

The tool then makes sure any server submission would be encrypted or hashed to maintain integrity and confidentiality. The tool goes through the source code and locates all submission parts (assuming code is JavaScript. The tool finds all submit methods, which would be AJAX, POST, and GET, then check the data whether it is passed on directly or wrapped with an encryption method or if the value since the definition has been changed).

At last, the tool generates a security quality report to help in knowing whether the system resources are wasted or well used. However, the report generating is considered as Dynamic Application Security Testing rather than Static Application Security Testing. Thus, it will be kept as future works.

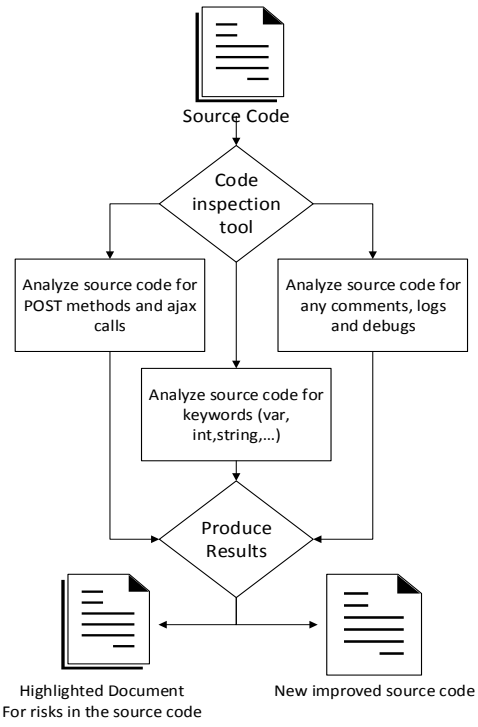

Figure 1. The proposed method

By looking at Fig 1, the source code goes into the tool to be analyzed and inspected. Once that is done, the results will go through multiple steps of analyzes in order to optimize the source code. In the next subheadings, Fig 1 will be broken down in detail as well as the method explained more.

1) The Logic of the Tool as an Algorithm: The tool is based on Regular Expressions (RE) [26]. RE is a sequence of symbols and characters expressing a string or pattern to be searched for within a longer piece of text. RE has multiple flags, which are `/g`, `/I`, and `/m`, and the tool is based on the combination of all as `/igm`. G stands for global as in global search, I is case insensitive as the form is multiline. Having igm means the search for the word global case insensitive in all lines of the text, which is the source code. The syntax assuming the language is JavaScript (JS) would be "/The characters match/igm". Table II has further demonstration.

TABLE II. JS EXAMPLE WITH REGULAR EXPRESSION OUTCOME

JavaScript Sample Code	Regular Repression	Outcome

function simple() {
Var x;
vaR y;
}
	/vAr/igm	2 Matches found

The tool would detect `var` according to the flags returning two as results. If g was not there, then the tool would have stopped after the first encounter with the result as 1. As for i, caps are ignored; everything is treated as small letters without it, the tool results would be zero. At last, m as multiline, the tool would have returned zero as the first line does not contain `var`.

2) Analyze code for comments, logs, and debugs: In Fig 2, the code will be inspected for anything that can explain the code or assist in reverse engineering it such as comments, logs, and debuggers. These are every developer's essence in troubleshooting the software and a guideline in understanding what is happening. At the same time, in the wrong hands, they can be used to abuse the software as in embedding harmful malware that targets that specific software when the source code is fully understood.

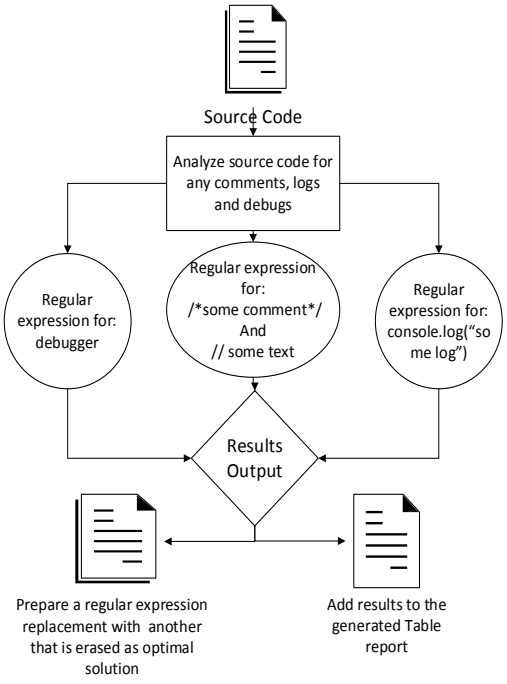

Figure 2. The logic behind code analysis for comments, logs and debugs

As mentioned earlier, leaving console logs in production systems is very dangerous as it may expose the system structure in the hands of cybercriminals.

3) Analyze code for keywords: Fig 3 shows that the code will be inspected for specific keywords, more specifically variable definition keywords. It is in order to ensure that the variable cannot be abused or altered while the software is running, also ensuring if it is sensitive, it should be secure and unreadable.

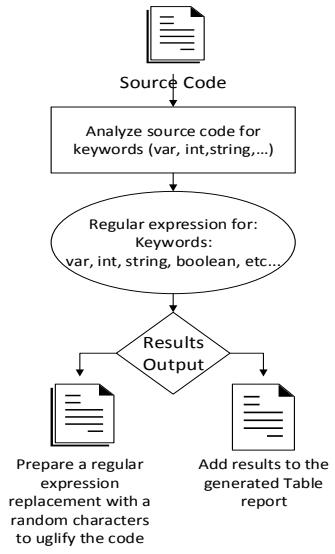

Figure 3. The logic behind code analysis for keywords

4) Analyze code for POST methods and ajax calls: As for Fig 4, the code will be inspected for all methods of sending data over the network in order to assess if the data being sent is secure or not. After all, a man in the middle attack can be performed to steal whatever data is being transferred.

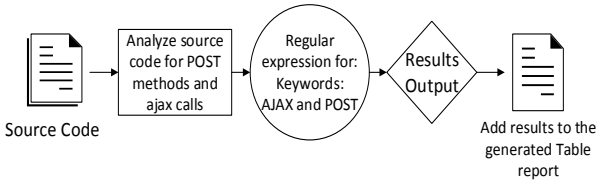

Figure 4. The logic behind code analysis for keywords

5) Produced Results: Two Outputs are generated. First, it will have things highlighted for anything that is considered a risk. Then it generates alongside what must be removed and to be added. The second document will have results of the new expected source code after the cleaning process of the code is complete (removing unnecessary codes).

4. CASE STUDY

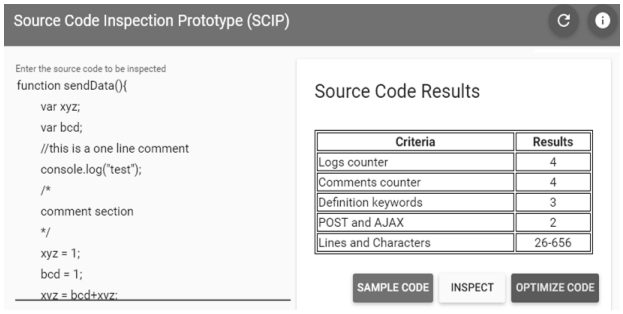

Figure 5. Source code inspection tool has inspected and produced the results

The prototype tool in Fig 5, which can be accessed at [23], has pointed out what should be looked at in the source code. The logic of the tool is based on multiple regular expressions. These regular expressions are meant to capture every criterion that is mentioned in the source code results table. The tool then optimizes the source code by removing what it can remove, leaving behind a better, shorter, and cleaner code that is less likely to be abused by any unethical user. As mentioned before, the tool inspections are based on a well written regular expression where the search flags as mentioned earlier. Table III shows that wherever the comment is, it is captured.

TABLE III. JS EXAMPLE WITH REGULAR EXPRESSION OUTCOME

#	Entered Code	No. of Comments, Logs, and Debugs	Expected output
1	*function UseCase1 () {*	**3 comments found.** @line2 // this is a … @line4 //same line… and @line5 /*in the…*/ **1 log found** @line3 console.log… **1 debug found** @line6 debugger;	*function UseCase1 () {*
2	*// this is a simple comment*		*var x;*
3	*var x; console.log("logging");*		*var y;*
4	*var y; //same line comment*		*var z;*
5	*var /*in the middle comment*/ z = x+y;*		*}*
6	*debugger;*		
7	*}*		

Similarly, for definitions counter, and POST/ajax calls the regular expressions with the same criterion of being global, case insensitive, and multiline. The tool counts wherever there is a match. The logic can be seen in Table IV.

TABLE IV. JS EXAMPLE WITH REGULAR EXPRESSION OUTCOME

#	Entered Code	No. of Definitions and Callings
1	*function UseCase2 () {*	**5 definitions found** @lines2,3,4,5 Keywords: var, int, boolean… **2 calling methods found**
2	*var abc;*	
3	*int x,y;*	
4	*boolean flag;*	
5	*var xhttp = new XMLHttpRequest();*	
6	*$.ajax({url: "someDummyDemo.txt", success: function(result){;}});*	@lines5,6 Keywords: XMLHttpRequest, ajax …
7	*}*	

5. VALIDITY AND COMPARISON

The proposed method uses Regular Expressions (RE), which has not been seen in any of the current tools. RE can be used to target multiple languages by expanding the keywords for each criterion that should be looked at in the source code. The tool can be easily expanded to include furthermore features.

However, the only drawback is that the tool, as it has been developed using AngularJS, the possibility of running dynamic tests are bounded to only web applications. As for accuracies, the tool may require plenty of tests to cover all different possible bugs and errors.

6. LIMITATIONS

Time was the only factor to limit enhancements and features added to the tool prototype, as the prototype shows enhancements that makes the source code valid more than validating as it bypasses the full report generation related to validity and makes a valid code. The tool, however, is capturing every criterion of software security. Thus, the tool could have done further tests and generate a detailed report of before and after showing the percentage of verification and validity, such as finding the final value of each variable and detecting errors and wrong coding practices. Aside from that, the most crucial point is that since the tool is still in a prototype phase, it still needs to perform multiple test cases to be able to eliminate every bug and error. The tool must handle and cover all aspects of a human error situation; to prevent the tool from miss-filtering the code. The prototype currently expects the correct JavaScript code to be analyzed.

7. CONCLUSION

The tool has covered the source code statically all aspects of VV, excluding T. Verification was through ensuring the source code of the system is at least vulnerable as it is supposed to be as an essential requirement. Validation that the system meets any security requirement from the stakeholder if requested and minimal security requirement from the software engineer side with no impact on the system performance. Testing of the source code is done dynamically, and the proposed tool works statically.

This study can be expanded by including Dynamic Application Security Testing (DAST), allowing the generation of quality reports of the consumed resources. Also, covering other languages and verifying the code is well written. Additionally, perform more in-depth analyzes in SAST mode to do better optimization and broader analysis to the source code. A document

highlighting for risks is another thing that the tool should be producing as an output.

REFERENCES

[1] O. Abahussain and M. Hammad, "Securing Systems and Software: Current State and Challenges," 2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO), Apr. 2019 [Online]. Available: https://doi.org/10.1109/ICMSAO.2019.8880396 [Accessed: 26-Mar-2020]

[2] L. Backman, "Why is security still an issue? : A study comparing developers' software security awareness to existing vulnerabilities in software applications," Dissertation, 2018 [Online]. Available: http://liu.diva-portal.org/smash/record.jsf?pid=diva2%3A1271102&dswid=1822 [Accessed: 29-Mar-2020]

[3] A. McCormac, T. Zwaans, K. Parsons, D. Calic, M. Butavicius, and M. Pattinson, "Individual differences and Information Security Awareness," Computers in Human Behavior, vol. 69, pp. 151–156, Apr. 2017 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0747563216308147 [Accessed: 29-Mar-2020]

[4] M. Bada, A. Sasse, and J. Nurse, "Cyber Security Awareness Campaigns: Why do they fail to change behaviour?," 2015 [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1901/1901.02672.pdf [Accessed: 29-Mar-2020]

[5] A. Yasin, L. Liu, T. Li, R. Fatima, and W. Jianmin, "Improving software security awareness using a serious game," *IET Software*, vol. 13, no. 2, pp. 159–169, Apr. 2019 [Online]. Available: https://doi.org/10.1049/iet-sen.2018.5095 [Accessed: 29-Mar-2020]

[6] J. Pan, "Software testing," Dependable Embedded Systems, 1999 [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.7121&rep=rep1&type=pdf [Accessed: 02-Mar-2019]

[7] O. Abahussain and M. Hammad, "Validating Software Security using Regular Expressions," 2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), Sep. 2019 [Online]. Available: https://doi.org/10.1109/3ICT.2019.8910303 [Accessed: 10-Apr-2020]

[8] J. Young, M. Zurko, J. Sobel, C. Bruggeman, and J. Taylor, "Securing Web Application Code by Static Analysis and Runtime Protection," Google Patents, 2017 [Online]. Available: https://patents.google.com/patent/US9841972B2/en [Accessed: 29-Mar-2020]

[9] M. Newman, "Apparatus and Method for Securing Web Application Server Source Code," Google Patents, 2019 [Online]. Available: https://patents.google.com/patent/US20190303601A1/en [Accessed: 30-Mar-2020]

[10] S. Shekyan, M. Coates, W. Hales, T. Peacock, and J. Call, "Mitigating security vulnerabilities in web content," Google Patents, 2019 [Online]. Available: https://patents.google.com/patent/US20190394223A1/en [Accessed: 30-Mar-2020]

[11] B. Arkin, S. Stender, and G. McGraw, "Software penetration testing," IEEE Security and Privacy Magazine, vol. 3, no. 1, pp. 84–87, Jan. 2005 [Online]. Available: https://doi.org/10.1109/msp.2005.23 [Accessed: 02-Mar-2019]

[12] D. P. Gilliam, J. C. Kelly, J. D. Powell, and M. Bishop, "Development of a software security assessment instrument to reduce software security risk," Proceedings Tenth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. WET ICE 2001 [Online]. Available: https://doi.org/10.1109/enabl.2001.953404 [Accessed: 02-Mar-2019]

[13] B. Potter and G. McGraw, "Software security testing," IEEE Security & Privacy Magazine, vol. 2, no. 5, pp. 81–85, Sep. 2004 [Online]. Available: https://doi.org/10.1109/msp.2004.84 [Accessed: 02-Mar-2019]

[14] G. Deepa and P. S. Thilagam, "Securing web applications from injection and logic vulnerabilities: Approaches and challenges," Information and Software Technology, vol. 74, pp. 160–180, Jun. 2016 [Online]. Available: https://doi.org/10.1016/j.infsof.2016.02.005 [Accessed: 31-Mar-2020]

[15] G. Tian-yang, S. Yin-sheng, and F. You-yuan, "Research on Software Security Testing," Zenodo, vol. 4, no. 9, pp. 1446–1450, Sep. 2010 [Online]. Available: https://doi.org/10.5281/zenodo.1081389 [Accessed: 26-Mar-2020]

[16] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung, "Vision," Proceedings of the second international workshop on Mobile cloud computing and services - MCS '11, 2011 [Online]. Available: https://doi.org/10.1145/1999732.1999740 [Accessed: 02-Mar-2019]

[17] R. Kumar, S. A. Khan, and R. A. Khan, "Revisiting Software Security: Durability Perspective," International Journal of Hybrid Information Technology, vol. 8, no. 2, pp. 311–322, Feb. 2015 [Online]. Available: http://dx.doi.org/10.14257/ijhit.2015.8.2.29 [Accessed: 15-Mar-2019]

[18] Z. Zhioua, S. Short, and Y. Roudier, "Static Code Analysis for Software Security Verification: Problems and Approaches," 2014 IEEE 38th International Computer Software and Applications Conference Workshops, Jul. 2014 [Online]. Available: https://doi.org/10.1109/compsacw.2014.22 [Accessed: 02-Mar-2019]

[19] Matthias, "mre/awesome-static-analysis," GitHub, 2019. [Online]. Available: https://github.com/mre/awesome-static-analysis [Accessed: 30-Mar-2019]

[20] S. Koussa, Ed., "What do SAST, DAST, IAST and RASP mean to developers?," Softwaresecured.com, 02-Nov-2018. [Online]. Available: https://www.softwaresecured.com/what-do-sast-dast-iast-and-rasp-mean-to-developers/ [Accessed: 26-Mar-2020]

[21] "PT Application Inspector," Ptsecurity.com, 2020. [Online]. Available: https://www.ptsecurity.com/ww-en/products/ai/ [Accessed: 26-Mar-2020]

[22] L. Papadopoulos, C. Marantos, G. Digkas, A. Ampatzoglou, A. Chatzigeorgiou, and D. Soudris, "Interrelations between Software Quality Metrics, Performance and Energy Consumption in Embedded Applications," Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems - SCOPES '18, 2018 [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3207719.3207736 [Accessed: 20-Jun-2020]

[23] O. Abahussain, "Source Code Inspection Prototype (SCIP)," Msc Prototypes 20110708, 2019. [Online]. Available: https://20110708.000webhostapp.com/#/SCI [Accessed: 28-May-2019]

[24] "Javascript and the Dangers of Console.log," Amido, 13-Mar-2019. [Online]. Available: https://amido.com/blog/javascript-and-the-dangers-of-console-log/ [Accessed: 06-Aug-2019]

[25] DEV Community, "Don't leave console logs in production," The DEV Community, 29-Oct-2018. [Online]. Available:

https://dev.to/mornir/-dont-leave-console-logs-in-production-14na [Accessed: 06-Aug-2019]

[26] J. Goyvaerts and S. Levithan, Regular Expressions Cookbook: Detailed Solutions in Eight Programming Languages. "O'Reilly Media, Inc.," 2012 [Online]. Available: https://books.google.com.bh/books?id=0Msuh5Vq-uYC [Accessed: 02-Aug-2019]

Omar Abahussain received his BSc. Degree in Computer Science from University of Bahrain, Bahrain 2016 and is currently studying MSc. In Software Engineering. Has published few conference papers and his interests fall in the field of software engineering, software security, and designing the software source code in a secure way to be safe and secure against abusers and hackers…

Mustafa Hammad received his MSc. Degree in Computer Science from Al-Balqa Applied University, Jordan in 2005. He completed his PhD in Computer Science at New Mexico State University in 2010. His research interests include wireless sensor networks, software engineering with focus on software analysis and evolution…

Fawzi Albalooshi is a faculty member in the department of computer science at the college of IT in the University of Bahrain. Dr. Fawzi Has earned his Ph.D. from the University of Wales in the field of Software Engineering. He has published many research articles in international journals and conferences. He has authored and edited books in the field of IT.