A Review On Remote Code Execution Vulnerability Detection and Mitigation

Preprint · December 2024

DOI: 10.13140/RG.2.2.28612.03202

1 author:

Prashadi Heshiki Halpe
Sri Lanka Institute of Information Technology
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

A Review On Remote Code Execution Vulnerability Detection and Mitigation

Prashadi H. Halpe
Dept.of Information Technology
Sri Lanka Institute of
Information Technology (SLIIT)
Colombo, Sri Lanka
it19017716@my.sliit.lk

***Abstract*—Since they are able to meet both commercial and consumer objectives, web apps are becoming more and more popular. Web apps may now deliver business services to its stakeholders in the most efficient and effective manner possible. Today, a variety of services are offered via web apps, and the efficiency of such services is gauged by the speed at which they handle requests and the usefulness of their informational features. But concurrently, a vulnerability to such services might arise through faulty authorization. At this time, cyber-attacks are a serious concern for any global digital transition. Various kinds of application level vulnerabilities still exist in the web system as a result of incautious coding practices throughout implementation and an inadequate level of security awareness. Among the security vulnerabilities in the modern world the major is the remote code execution (RCE). RCE has been identified as the second-most vital online application vulnerability. An exhaustive review on RCE vulnerability is presented in this article.**

***Keywords*—Remote Code Execution, Vulnerability, security threat, SQLIA, Code Injection, DoS, Impact, Detection, Mitigation, Sanitized inputs, access control**

I. Introduction

Remote Code Execution (RCE) is kind of a security flaw that enables attackers to access to a distant machine's networks and execute arbitrary code on it. It is a technique for leveraging the internet to remotely infiltrate and run code in a target machine or system.

Since they may be attacked even though an intruder has never had control over the system or device, RCE are arguably the most dangerous form of ACE. Loss of data, service interruption, the introduction of ransomware or other malware, and defensive positioning of the attacker to other critical applications are just a few of the grave repercussions which may occur from RCE, which is similar to a complete compromise of the impacted application or system by the code.

This attack takes use of the potential for executable code to be infiltrated into a string or file and then run or assessed. This may occur as a result of the programming language's parser failing to check input data and allowing it to proceed. Typically, infiltrated codes are written in the targeted application's programming language. These languages might be Ruby, PHP, Java, Python, etc.

The advantages provided by the process an intruder is striving for will usually be acquired when the code is executed, based on the vulnerability the attacker uses. RCE is typically followed by efforts to increase rights and benefits and seize control at the admin or core level because of this. Unluckily, this increased access also gives attackers the ability to cover their tracks more expertly. The remote code execution vulnerability, however, still has the risk of doing significant damage even in the absence of further unusual power.

As Remote Code Execution can be directly responsible for the sever impacts financial sector and other organizations such as security breaches, leakage of sensitive data, possibility of ransomware attacks and crypto-jacking vulnerabilities etc. it is very important to secure organizations applications, servers and systems from having intrusions by executing remote inject codes. Because of the vulnerabilities of the websites created by small to medium-sized entrepreneurs, and in some scenarios even large corporations or billion-dollar companies, there will undoubtedly be an increase in the amount of breaches of information security as even smaller companies transform to digital platforms to enhance their businesses after the Covid19 outbreaks strikes. Ensuring security and privacy to reduce data security breaches must be done with the extreme care as it impacts not only the business that is damaged by the security flaw but also its customers, workers, investors, and other stakeholders as well.

II. Research Objectives

This paper provides an in-depth existing literature review on Remote Code Execution Vulnerabilities. Majorly this paper reviews on RCE vulnerability detection and mitigation approaches. Apart from that, this paper will provide a precise handful of information on the types of RCE attacks, RCE exploit techniques and impacts of RCE attacks which are extracted from the previous studies.

III. Literature Review

A. *What is Remote Code Execution - RCE*

Majorly previous researches pointed out that the major cyber-attack is the remote code execution (RCE) where the client data will be executed on the server side [1]. Client side code injection will be used to obtain unauthorized access. [1]. Remote code execution is a hacking technique that can access someone’s computing device and make modifications via the internet [2]. Attacker runs server commands on a remote server [2]. The public availability of web applications makes them a pretty easy target and one of the primary vectors for compromising system and network security. Furthermore, the huge amount of installations make both online systems and host systems such as personal computers, web servers, internet of things devices, VMs are desirable objective for malware that proliferate across internal and external networks by taking advantage of web-related flaws [3]. Even though RCE vulnerability has the potential to cause harms severely, due to the peculiarities of RCE vulnerability, it received very small attention [4].

B. *How RCE works*

Different system services grant credentials for trusted code behavior, computer technology, and remote attestation. A trustworthy program behaving in a certain way picks up data and information on the server side and transfers it to the attacker. Attacker creates malicious scripts, such as echo "hello," that are used to attack the target website's RCE vulnerability. This created code is then delivered to the server via the RCE-based susceptible website. Malicious software runs the remote server and responds to the attacker's server message. It will be deemed an RCE susceptible website if this message is pertinent to the needs of the attacker [2].

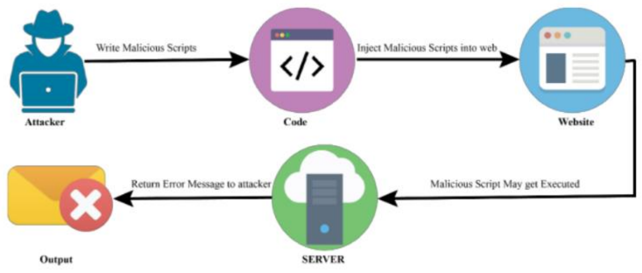

Write Malicious Scripts
Inject Malicious Scripts into web
Attacker
Code
Website
Return Error Message to attacker
Malicious Script May get Executed
Output
SERVER

Figure 1: Remote Code Execution Process [2]

C. *Types of RCE Attacks*

It is known as the Web Based RCE vulnerability, because it exists in a web application and allows an attacker to execute system commands on the web server [2] and a service running on any platform (Android, Mac, Windows computer) that is vulnerable to allowing an attacker to run system commands is known as a System Based RCE vulnerability [2]. It is possible for a web-based application to have a system vulnerability or weakness. Inadequate form validation or input sanitization, improperly configured web servers, and security weaknesses in the application's architecture all contribute to security breaches [2]. SQL Injection and Cross-Site Scripting (XSS) flaws were discovered in more than half of the web applications over a 2015–2016 period, based on a previously conducted code injection detection research study [3].

SQL injections are vulnerabilities that are widely recognized and may even be detected in monitoring tools [3]. When an application transmits dubious data to an interpreter, code injection vulnerabilities (also known as injection weaknesses) develop. Most frequently, injection issues may be identified in OS commands, XML parsers, SMTP headers, LDAP, XPath, or NoSQL queries, as well as program arguments, XML parsers, LDAP, and NoSQL queries etc. [3]. When reviewing the source code as opposed to conducting testing, injection vulnerabilities are often easier to identify [3]. A code injection method known as SQL injection is used to exploit data-driven systems by inserting malicious SQL commands into input fields for execution [5]. It must employ weakly coded and unexpectedly executed SQL commands to exploit user input that has been wrongly checked for escape characters used in string literals. Though it may be used to target any sort of SQL database software, SQL injection is a common attack vector for online applications [3]. The attacker inserts Structured Query Language code into a web form's text field or an HTTP/S request header in order to access or modify data. Performance and confidentiality are both impacted by the SQL injection flaw which enables the intruder to send commands directly to the web application's internal database [3].

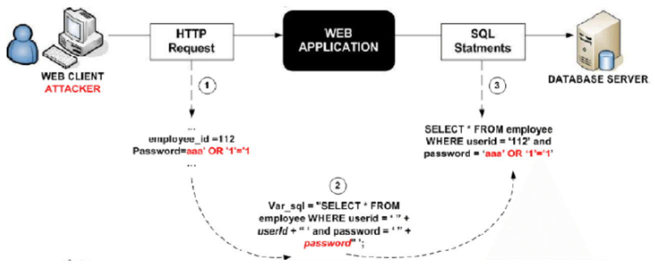

WEB CLIENT
ATTACKER
HTTP Request
WEB APPLICATION
SQL Statments
DATABASE SERVER
1
employee_id =112
Password=aaa' OR '1'='1
2
Var_sql = "SELECT * FROM employee WHERE userid = ' " + userId + " ' and password = ' " + password" ';
3
SELECT * FROM employee WHERE userid = '112' and password = 'aaa' OR '1'='1'

Figure 2: SQL Injection Attack [6]

D. *RCE Exploit Techniques*

According to a previous research study on REC it is found that through request-based fields, such as URL base parameters and input field-based parameter requests, an attacker of a web application can exploit an RCE vulnerability. When a server receives a request from an attacker via a third party, the server is anticipated to perform it as authorized users and react to the attacker. [2].

Many exploitations Techniques can grant client-level access to a device's root level. Thus, utilizing flaws is also a possibility. Therefore, it is crucial to start at a minimal level and ascend from there eventually achieve the core [2]. Exploiting the $_GET and $_POST methods is becoming more common because of a shortage of security agents. The intruder uses a variety of strategies to gain access to the administrator interface using a cmd. For instance, an intruder may use Netcat, which creates and accepts TCP and UDP networks and utilizes them to read and write data until they are terminated. This communication subsystem's TCP/UDP capabilities allow users to engage with net services and applications on the application layer naturally or via scripts [2].

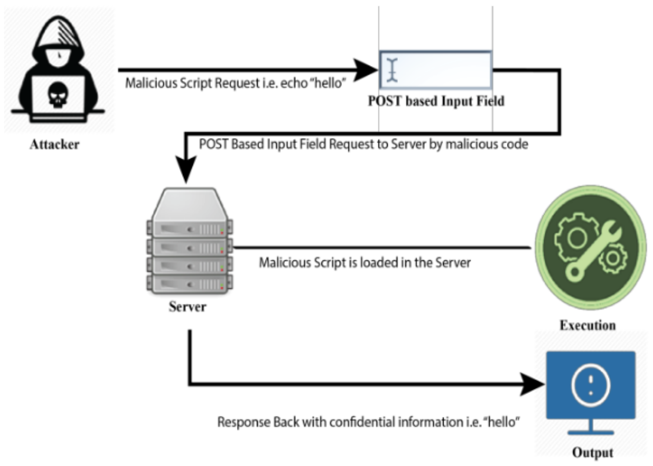

Attacker
Malicious Script Request i.e. echo "hello"
POST based Input Field
POST Based Input Field Request to Server by malicious code
Server
Malicious Script is loaded in the Server
Execution
Response Back with confidential information i.e. "hello"
Output

Figure 3: RCE Exploit Techniques [2]

E. *Impact of RCE Attacks*

Generic input validation issues are the cause of many application security flaws [7]. Intensive preventive controls to serve as a line of protection against input intrusions include appropriate input authentication. Even though the notion of authentication mechanism is straightforward, it can be difficult to implement since different applications have different definitions of what constitutes acceptable input [7]. There is not a general correlation between input and danger of vulnerability since there is no trend of illicit injection [7]. There are numerous instances when specialized authentication is needed for each field, not every input can be sanitized by a single validation rule [7]. One of the main grounds for care is the risk to code injection since, if an illegal incursion is performed through the weak spot in the security system's armor, the security system as a whole is put at risk [7].

F. *RCE Vulnerabilities Detection*

Typically from the past, scanners and fuzzes were used to discover injection vulnerabilities [3]. However, they become victim to complicated strikes, especially those using unknown routes. [3]. Given the variety of stealth tactics that attackers have perfected throughout time, preventing against a code injection attack is a difficult undertaking. Examples of evasion techniques include the use of escape and comment symbols that may evade processing, script and code encapsulation to bypass static, truncation, encoding and decoding (base64, URL, etc.), and more generally blind SQL injection with complex syntax to take advantage of web applications with concealed database feedback [3]. To identify code injection attacks (SQLIA) and denial of service attacks, among other things, possible machine learning and deep learning methods are discovered [3]. In order to pinpoint queries that did not fit several models of normal queries at runtime, including string model and data type-independent model, the AMNESIA tool, a technique to identify SQLIA (SQL Injection Attacks) employing neural networks, was suggested. The approach demonstrated the tremendous potential of deep learning in malicious query identification even though it did not achieve high accuracy [8]. An approach called "Swaddler," which evaluates a web application's internal state and learns the correlations between the application's vital execution points and its internal state, was previously presented in another study as a method for detecting attacks against web applications, specifically those developed using PHP [9]. A novel minimalist strategy is proposed to trace both Denial of Service (DoS) and Code Injection (CI). With a detection rate against flood strikes ranging from 61.9% to 98.5%, it recognizes DoS and CI attacks as anomalies utilizing an auto-encoder for dynamic response anomaly detection [10]. The research paper on web application intrusion detection proposes a tool that employs DNN and auto-encoders for semi-supervised and unsupervised learning for web attack detection, including SQLIA [11]. With the aim of modeling SQL commands and parsing the tree structure of SQL queries as features, a clustering technique is presented for mapping SQL queries to applications and recognizing harmful malicious and non-malicious queries [12].

Table 1: Code injection detection methods based machine learning and deep learning [3]

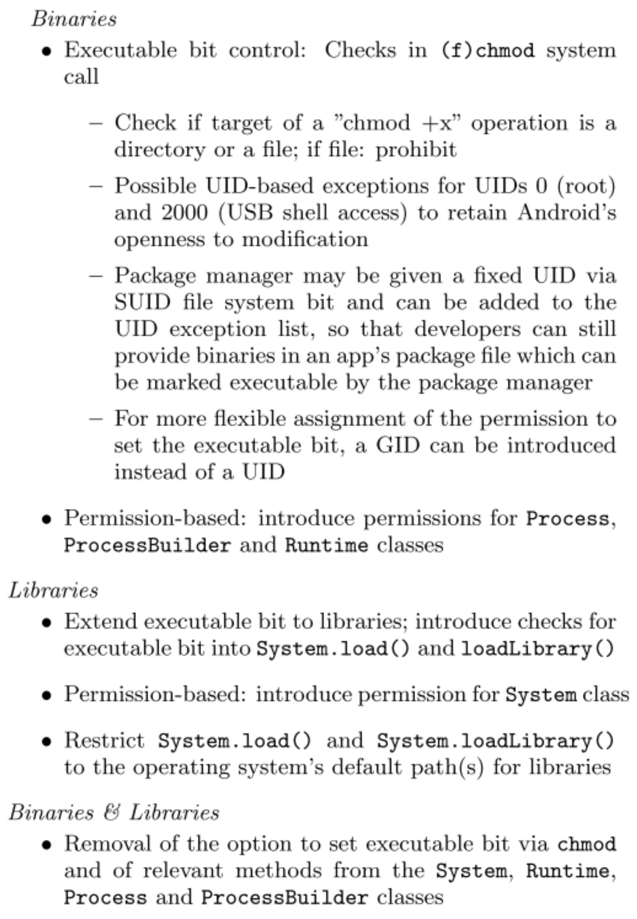
Year	Method	Learning	Language
2005[13]	AMNESIA	NDFA	SQL
2007[9]	Swaddler	libAnomaly	PHP
2008[14]	Unnamed	OC-SVM	PHP, SQL
2009[12]	Unnamed	Clustering	SQL
2013[15]	Unnamed	Bayesian	SQL
2017[16]	HDLN	Hybrid	JavaScript
2017[17]	Unnamed	CNN, RNN	SQL
2017[18]	AMODS	SVM	SQL, XSS
2018[19]	ConsiDroid	ConcolicExecution	SQL
2018[20]	DeepXXS	LSTM	XSS
2019[10]	Unnamed	Autoencoder	SQL

Another earlier research reported that the detection accuracy for SQL and PHP code injection attacks using the current RCE vulnerability detection system, which includes a payload-based anomaly detection approach by combining structural information from a protocol analyzer [14], was 49%. [3]. Traditional SQL injection attacks may be detected using the currently available techniques, and more complicated and novel threats may also be detected [3].

The detection of RCE vulnerabilities in PHP scripts was suggested in another piece of existing literature using a path and context sensitive inter procedural static analysis [21]. With fewer false positives, they successfully identified RCE vulnerabilities in PHP scripts using a unique approach that took into account both string and non-string behavioral patterns [21].

G. ***RCE vulnerabilities Mitigation***

An existing research of preventing unauthorized access by executing a remote code (RCE attack) on android applications introduced an effective method of domain isolation has been implemented using compulsory systems of access control like SEAndroid and Tomoyo [22]. In addition to exploit execution, they block illegal entry to other applications' resources [22], [23]. Since it is deployed at the middleware layer, the [22] technique enables the consumer to be adequately informed about halted executions. Nowadays, any program with Internet connectivity is capable of downloading and run any native code, including core vulnerabilities, from the Internet. The [22] strategy offers an answer to this issue. Although it might not offer complete protection, it will increase the requirements for successful exploitation to the level where every native code-based attack now in use would be ineffective [22]. As rogue applications can no more access and execute arbitrary native code files via the Internet, it reduces the devices' potential vulnerabilities [22]. Recently, every application with internet connectivity has the possibility to download and run any native remote code, including root vulnerabilities. Several suggestions for regulating native code on the Android platform have been made [22]. The following figure include the list the suggestions.



Binaries

- Executable bit control: Checks in `(f)chmod` system call
  - Check if target of a "chmod +x" operation is a directory or a file; if file: prohibit
  - Possible UID-based exceptions for UIDs 0 (root) and 2000 (USB shell access) to retain Android's openness to modification
  - Package manager may be given a fixed UID via SUID file system bit and can be added to the UID exception list, so that developers can still provide binaries in an app's package file which can be marked executable by the package manager
  - For more flexible assignment of the permission to set the executable bit, a GID can be introduced instead of a UID
- Permission-based: introduce permissions for `Process`, `ProcessBuilder` and `Runtime` classes

Libraries

- Extend executable bit to libraries; introduce checks for executable bit into `System.load()` and `loadLibrary()`
- Permission-based: introduce permission for `System` class
- Restrict `System.load()` and `System.loadLibrary()` to the operating system's default path(s) for libraries

Binaries & Libraries

- Removal of the option to set executable bit via `chmod` and of relevant methods from the `System`, `Runtime`, `Process` and `ProcessBuilder` classes

Figure 4: Unauthorized Access Controlling Measures Suggested for Android Applications [22]

According to the [7] literature to prevent from the code injection vulnerabilities, Utilization of stored procedures or parameterized queries as against to string concatenation when accessing a database. Due to the separation of a query's semantics and data, this will block commands added by the user from being carried out. A single accidental query bypass might be enough to expose the application to risk [7], [24], [25], [26], [27], [28], [13]. Check the URL query string and text input for incorrect characters. Reduce the number of characters in URL query strings and input fields on the web to a reasonable number. All legitimate data must be allowed, and any data that could be harmful must be denied or cleansed. When permissible letters or sequences of characters also have specific significance for the subsystem, this might be challenging and could need evaluating the data oppose a syntax. For instance, changing " to &quot, to &lt, > to &gt" is a straightforward way to sanitize data that is shown in a browser [7], [24], [25], [26], [27], [28], [13]. Every inputs should be validated both on the server and client sides. From a security standpoint, server side validation is vital. By disabling javascript and vbscript in the browser, client side validation may be simply avoided. But in addition to server-side validation, client side validation should be employed for a number of many other advantages [7], [24], [25], [26], [27], [28], [13]. Failures that reveal details about the database or the original source code really should not be shown to the users. The fact that error messages provide extra information from the database that would not otherwise be exposed makes them helpful to attackers. When anything turns out badly, it is frequently considered to be beneficial for the program to provide the user an error message so that they may provide critical info to the technical support team if the situation continues. It is recommended to provide a basic error message with a simple "some problem has encountered with a distinct id" in it. Upon this server, which is accessible to the technical support staff, the distinctive id will be recorded together with the real fault diagnosis [7], [24], [25], [26], [27], [28], [13]. Run the database using a user with limited privileges. An attacker may be able to execute almost endless commands on the database if they run a program that establishes a connection to that utilizing database's admin privileges [7], [24], [25], [26], [27], [28], [13]. Remove the stored procedures on the system. An intruder might leverage stored procedures to carry out remote code execution if the data base server's software implementation is operating as SYSTEM, which is analogous to the Administrator Level on Windows [7], [24], [25], [26], [27], [28], [13].

IV. Future Research

There are plenty of studies conducted to implement methods on detecting SQL injection attacks and PHP code injection attacks. But systems and applications that are implemented using other programming languages such as Python, JavaScript, Java and C# etc. can also have code injection vulnerabilities which may tempt the system / application fall in to remote code execution. It is important and will be very beneficial in future to research on detection methods and tools for Python, JS, Java, and C# code injection vulnerabilities to prevent the RCE vulnerabilities occur in the systems / applications written in those programming languages.

Not only that, but also upgrading the existing RCE vulnerability detection tools and applications in to automated mode and introducing the automated tools to detect not only code injection vulnerabilities, also other several RCE vulnerabilities concurrently is a huge step to benefit the users to safeguard from most RCE attacks.

Continuing more research and studies on RCE exploit techniques: Remote Code Evaluation and Stored Code Evaluation are important for the Information Security field in the future.

V. Conclusion

Any system, application, network or server is always possible to have a RCE attack if there are any vulnerabilities. Therefore, it is important to detect those vulnerabilities and take actions to prevent them beforehand facing the consequences. If RCE attack is occurred, legal users may face the detrimental impacts of sensitive data breaches, denial of service (DoS), escalating privileges, penetration and etc. This will cause serious information security threats such as cryptomining / cryptojacking and ransomware.

ACKNOWLEDGMENT

The page. The author would like to express gratitude towards Mr. Amila Senarathne, Senior Lecturer at the Computer Systems and Networking Engineering Department, Faculty of Computing, Sri Lanka Institute of Information Technology, for his guidance and the opportunity given to successfully complete a review paper on an information security-related subject. Additionally, the author would like to express gratitude to the other authors of the materials cited as sources for this review.

REFERENCES

[1] S. Sen, S. Patel, and P. Richhariya, "A Critical Review for Remote Code Execution Vulnerability Detection," vol. 2, no. 8, pp. 70–77, 2014.

[2] S. Biswas, M. K. Sohel, M. M. Hasan Khan Sajal, and T. Afrin, "A Study on Remote Code Execution Vulnerability in Web Applications, International Conference on Cyber Security and Computer Science," no. November, pp. 1–8, 2018, [Online]. Available: https://www.researchgate.net/publication/328956499.

[3] S. Abaimov and G. Bianchi, "CODDLE: Code-Injection Detection with Deep Learning," *IEEE Access*, vol. 7, pp. 128617–128627, 2019, doi: 10.1109/ACCESS.2019.2939870.

[4] S. Bier, B. Fajardo, O. Ezeadum, G. Guzman, K. Z. Sultana, and V. Anu, "Mitigating Remote Code Execution Vulnerabilities: A Study on Tomcat and Android Security Updates," in *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, 2021, pp. 1–6, doi: 10.1109/IEMTRONICS52119.2021.9422666.

[5] J. E. T. Akinsola, A. Oludele, I. A., and S. Kuyoro, "SQL Injection Attacks Predictive Analytics Using Supervised Machine Learning Techniques," *Int. J. Comput. Appl. Technol. Res.*, vol. 9, pp. 139–149, Apr. 2020, doi: 10.7753/IJCATR0904.1004.

[6] A. Tajpour, S. Ibrahim, and M. Masrom, "SQL Injection Detection and Prevention Techniques," *Int. J. Adv. Comput. Technol.*, vol. 3, pp. 82–91, Aug. 2011, doi: 10.4156/ijact.vol3.issue7.11.

[7] S. Madan and S. Madan, "Security Standards Perspective to Fortify Web Database Applications from Code Injection Attacks," in *2010 International Conference on Intelligent Systems, Modelling and Simulation*, 2010, pp. 226–230, doi: 10.1109/ISMS.2010.50.

[8] F. Valeur, D. Mutz, and G. Vigna, *A Learning-Based Approach to the Detection of SQL Attacks*. 2005.

[9] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna, *Swaddler: An Approach for the Anomaly-Based Detection of State Violations in Web Applications*. 2007.

[10] A. Gurina and V. Eliseev, "Anomaly-Based Method for Detecting Multiple Classes of Network Attacks," *Information*, vol. 10, p. 84, Feb. 2019, doi: 10.3390/info10030084.

[11] Y. Pan *et al.*, "Detecting web attacks with end-to-end deep learning," *J. Internet Serv. Appl.*, vol. 10, Dec. 2019, doi: 10.1186/s13174-019-0115-x.

[12] C. Bockermann, M. Apel, and M. Meier, "Learning SQL for Database Intrusion Detection using Context-Sensitive Modelling (Re-submission)," Jul. 2009, doi: 10.17877/DE290R-2052.

[13] W. G. J. Halfond and A. Orso, "AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks," 2005.

[14] H. Jahankhani, S. Fernando, M. Z. Nkhoma, and H. Mouratidis, "Information systems security: cases of network administrator threats," *Int. J. Inf. Secur. Priv.*, vol. 1, p. 13+, Nov. 2007, [Online]. Available: https://link.gale.com/apps/doc/A172169656/AONE?u=anon~d3bbb40a&sid=googleScholar&xid=fc8725ee.

[15] E.-H. Cheon, Z. Huang, and Y. S. Lee, "Preventing SQL Injection Attack Based on Machine Learning," *Int. J. Adv. Comput. Technol.*, vol. 5, pp. 967–974, 2013.

[16] R. Yan, X. Xiao, G. Hu, S. Peng, and Y. Jiang, "New deep learning method to detect code injection attacks on hybrid applications," *J. Syst. Softw.*, vol. 137, pp. 67–77, 2018.

[17] R. Cai, B. Xu, X. Yang, Z. Zhang, and Z. Li, “An Encoder-Decoder Framework Translating Natural Language to Database Queries,” Nov. 2017.

[18] Y. Dong *et al.*, “An adaptive system for detecting malicious queries in web attacks,” *Sci. China Inf. Sci.*, vol. 61, Mar. 2018, doi: 10.1007/s11432-017-9288-4.

[19] E. Edalat, B. Sadeghiyan, and F. Ghassemi, *ConsiDroid: A Concolic-based Tool for Detecting SQL Injection Vulnerability in Android Apps*. 2018.

[20] Y. Fang, Y. Li, L. Liu, and C. Huang, “DeepXSS: Cross Site Scripting Detection Based on Deep Learning,” in *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*, 2018, pp. 47–51, doi: 10.1145/3194452.3194469.

[21] Y. Zheng and X. Zhang, “Path sensitive static analysis of web applications for remote code execution vulnerability detection,” in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 652–661, doi: 10.1109/ICSE.2013.6606611.

[22] R. Fedler, M. Kulicke, and J. Schütte, “Native Code Execution Control for Attack Mitigation on Android,” in *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, 2013, pp. 15–20, doi: 10.1145/2516760.2516765.

[23] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry, *Practical and lightweight domain isolation on Android*. 2011.

[24] M. Muthuprasanna, K. Wei, and S. Kothari, *Eliminating SQL Injection Attacks - A Transparent Defense Mechanism*. 2006.

[25] E. Bertino, A. Kamra, and J. Early, *Profiling Database Application to Detect SQL Injection Attacks*. 2007.

[26] S. Thomas and L. Williams, *Using Automated Fix Generation to Secure SQL Statements*. 2007.

[27] K. Wei, M. Muthuprasanna, and S. Kothari, *Preventing SQL injection attacks in stored procedures*. 2006.

[28] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao, *A Static Analysis Framework For Detecting SQL Injection Vulnerabilities*, vol. 1. 2007.

AUTHOR PROFILE

Prashadi Heshiki Halpe is a final year undergraduate at the Sri Lanka Institute of Information Technology, Malabe, Sri Lanka following the degree B.Sc. (Hons) in Information Technology Specialized in Data Science. She is currently employed as a Trainee Data Analyst at KPMG, Sri Lanka. She is interested in the fields of Machine Learning, Deep learning, Data Science and Artificial Intelligence.