Mitigating Remote Code Execution Vulnerabilities: A Study on Tomcat and Android Security Updates

Stephen Bier[1], Brian Fajardo[2], Obinna Ezeadum[3], German Guzman[4], Kazi Zakia Sultana[5], Vaibhav Anu[6]
Department of Computer Science
Montclair State University
Montclair, New Jersey, USA
{biers1[1], fajardob1[2], ezeadumo1[3], guzmang3[4], sultanak[5], anuv[6]}@montclair.edu

2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS) | 978-1-6654-4067-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/IEMTRONICS52119.2021.9422666

***Abstract*—The security of web-applications has become increasingly important in recent years as their popularity has grown exponentially. More and more web-based enterprise applications deal with sensitive personal and private information, which, if compromised, can not only lead to system downtime, but can also cause mean millions of dollars in damages to the organization. It is critical to protect web-applications from the constant onslaught of hacker attacks. Remote Code Execution (RCE) attacks are one of the most prominent security threats for software systems, especially Java-based systems. In the current study, we have studied the security update reports for RCE vulnerabilities published by two Java-based projects: Apache Tomcat and Android. We analyzed and categorized the code-fixes (i.e., patches/updates) that were applied to mitigate/fix fifty-one (51) RCE vulnerabilities in the two above-mentioned Java projects. Our analysis showed that a significant majority of the RCE vulnerabilities found in Java projects can be mitigated with just five (5) types/categories of code-fixes. Overall, our goal was to study RCE vulnerabilities in an effort to provide programmers with a handy list of code-fixes, thus making it easier for them to effectively mitigate known RCE vulnerabilities in their own Java-based applications.**

Keywords—software security, software engineering, vulnerabilities, remote code execution, open source software

I. INTRODUCTION

Software security bugs, also known as vulnerabilities, continue to be an important and potentially the most expensive issue affecting all aspects of our cyber society. There has been significant research effort toward preventing vulnerabilities from occurring in the first place, as well as toward automatically discovering vulnerabilities, but so far these results remain fairly limited [15, 16].

Remote Code Execution (RCE) has been recognized as one of the most harmful threats for web applications [1]. Although RCE is a special kind of cross-site scripting attack, RCE attacks have some variants including requiring state consideration of both server and client, both string and non-string manipulation of client inputs, and involving multiple requests to more than one server-side scripts [1]. Static analysis tools can be potentially used for detecting vulnerabilities [17, 18, 19]. Static code analysis tools locate vulnerabilities within source code using data flow analysis or taint analysis techniques [20]. As RCE attacks mostly depend on path conditions and involve both string and non-string operations, most static analysis tools fail to detect RCE attacks as they follow context free grammar and model only string operations [21, 22]. As a result, the false positive rates are high in those tools. On the other hand, the existing literatures do not focus on how RCE vulnerabilities have been resolved in real world applications so that developers can have a handy list of techniques to mitigate those problems.

In this paper, we focus on identifying the various ways developers (i.e., programmers) mitigate/fix RCE vulnerabilities that are reported in Java-based software systems (fixing a reported or known security vulnerability is more commonly referred to as security update or patch and is generally done by changing/adding/deleting lines of code).

In a sense, the primary objective of this study is to identify the most common types of code changes (i.e., updates/patches) that are applied by programmers when RCE vulnerabilities are reported in their software. To meet our objective, we reviewed different systems that publish security update reports and determine if there are any similarities in the RCE vulnerabilities and the updates that were implemented to fix them. In our research, we reviewed security updates reports for Apache Tomcat and Android. The major contributions of our work have been stated below:

1. The study is conducted on two major Java-based systems: Apache Tomcat and Android. The study has identified the most frequently used mitigation techniques for fixing RCE vulnerabilities that can be exemplary for Java based software developers.
2. We anticipate the findings in this study may be of assistance to the developers in avoiding frequent programming mistakes that can lead to RCE attacks.
3. The common security updates discussed in this paper will help the developers to mitigate (or fix) RCE issues and thus reduce the likelihood of RCE attacks in the future.

In Section II, we discuss some related works to our study. Section III describes our research methodology. Section IV focuses on data analysis and results. In Section V, we discuss the limitations of our work. Section VI provides a brief discussion on the implications of our findings and finally Section VII concludes the paper with some future plan.

II. RELATED WORK

In this section, we highlight some existing works that focused on remote code execution (RCE) vulnerability analysis and detection.

Remote Code Execution is considered as a special kind of Cross Site Scripting (XSS) attacks [1]. Like XSS and SQL injection attacks, RCE occurs when invalid client-side inputs

978-1-6654-4067-7/21/$31.00 ©2021 IEEE

are undesirably converted to scripts and executed [1]. Although researchers have already put significant efforts on identifying and mitigating XSS and SQL injections vulnerabilities [2-7], RCE vulnerability got very little attention due to its unique characteristics [1]. Zheng et al. [1] proposed a path and context sensitive inter procedural static analysis to detect RCE vulnerabilities in PHP scripts. They devised a novel algorithm featuring both string and non-string behavior of a program and successfully could detect RCE vulnerabilities in PHP scripts with less false positive rates [1]. In another study [8], the authors assessed the multi-variant code execution technique to prevent the execution of malicious code. The idea of multi-variant code execution is detecting any malicious attempt during run-time. While running two or more slightly different variants of the same program in lockstep on a multiprocessor, the variants are monitored and any divergence from the regular behavior raises an alarm indicating the possible anomaly. The trade-off between security and performance is the major limitation of this approach [8]. Hannes et al. [9] studied expert opinions on how three variable (i) non-executable memory, (ii) access and (iii) exploits for High or Medium vulnerabilities as defined by the Common Vulnerability Scoring System contribute to the successful remote code execution attacks. Both access and the severity of the exploited vulnerability were perceived as important by the experts; non-executable memory was not seen as relevant to RCE according to the study [9]. In [10], the authors presented a case study on RCE vulnerability and analyzed different types of RCE and their impact on applications. Another paper [13] proposed a new mechanism for trusted code remote execution. The method creates a trusted platform integrating the identity authentication, platform authentication and behavior authentication based on trusted computing technology, remote attestation and trusted behavior for remote code execution [13]. There are some other research studies on remote code execution which focused on remote code execution vulnerabilities in specific domains or platforms [11, 12, 14].

Overall, there is a shortage of research on modeling of discovered security vulnerabilities to capture how and why an implementation fails to achieve the desired level of security. This paper analyzes some real vulnerable code and their fixes so that programmers can be aware of those frequently happened programming mistakes and are aware of their possible mitigation techniques.

More specifically, we focused on Java-based applications and identified common code changes/fixes that are used to mitigate RCE vulnerabilities (which has not been investigated in earlier research). Most of the previous studies either devised techniques to prevent RCE or detect RCE during runtime. Those studies lack in highlighting some common programming practices that are used by developers to fix RCE vulnerabilities. In our study, we present RCE updates/fixes so that developers can be guided during the maintenance phase and can ensure future software releases are secure.

III. Methodology

This section describes the Research Questions (RQs) and the data collection process for this study.

A. Research Questions

The following research questions were formulated to guide the data collection for this study:

RQ1: Do software systems suffer from Remote Code Execution (RCE) vulnerabilities more frequently when compared to the other types of security vulnerabilities?

RQ2: What types of patches (i.e., code-fixes) are usually added to mitigate the known RCE vulnerabilities in Java-based software systems?

B. Data Collection

The following paragraphs describe the data that we collected to answer the two research questions (RQs).

To answer *RQ1*, we collected the vulnerability-counts for the last 5 years (2015 to 2019) for the most common types of vulnerabilities (RCE, Denial of Service, Overflow, XSS, SQL injection) reported to a vulnerability datasource called CVE Details (https://www.cvedetails.com/).

With respect to *RQ2*, we focused specifically on collecting information about Remote Code Execution (RCE) vulnerabilities reported in Java-based software projects. Furthermore, we wanted to collect information about *how programmers fix the RCE vulnerabilities* reported in their Java-based software systems. Many open-source software projects publish *security update reports* on their project websites (for example, the security reports for Apache Tomcat are publicly available and can be found here: http://tomcat.apache.org/security.html).

We identified two such open-source Java-based software projects: ***Apache Tomcat*** (mentioned above) and ***Android*** (https://source.android.com/security/bulletin). An overview of the steps taken to collect the code-fixes (i.e., patched) applied to fix RCE vulnerabilities reported in each of the two systems is provided below:

1) Apache Tomcat Data Collection

To gather the data from Tomcat, we first went to the Tomcat's Security Reports page: http://tomcat.apache.org/security.html. This page displays a list of the known security vulnerabilities for each version of Tomcat as illustrated in Fig. 1. Tomcat Release 3.x was selected and all the Remote Code Execution (RCE) updates within this release were identified and reviewed. This was the most time and effort intensive step of our data collection process. Please note that our goal was to collect information regarding what kind of updates (i.e., patches or code-fixes) are applied by the programmers to known RCE vulnerabilities.

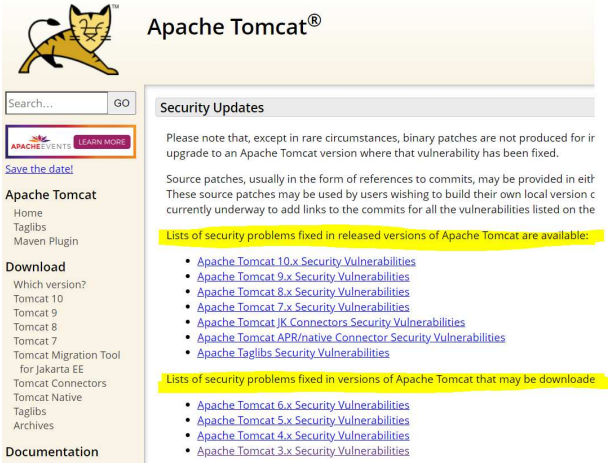

Fig.1. Tomcat: Security Updates

Fig.2. Tomcat: Security Update Details for a Sample RCE Vulnerability found in Tomcat

Therefore, for each RCE update found during our search, the revision number was selected as presented in the information page for that particular revision. Fig. 2 contains the information page for Revision 1809921. As can be seen in Fig. 2, each revision page contains a "path changed" link which in turn contains the line(s) of code that were added and/or removed to fix/mitigate the reported RCE vulnerability.

Table I presents a sample set of RCE vulnerabilities in

TABLE I. TOMCAT: SAMPLE SET OF RCE VULNERABILITY PATCHES/UPDATES THAT WERE FOUND AND STUDIED

Common Vulnerabilities and Exposures No. (CVE No.)	Affected Versions	Fixed version
2013-4444	7	7.0.39
2016-8735	7, 8, 9	9.0.0.M12
2017-12615	7	7.0.80
2017-12617	7, 8, 9	9.0.0.M15
2019-0232	7, 8, 9	9.0.17

Apache Tomcat that were identified and investigated during this study.

2) Android Data Collection

Similar to Tomcat, for Android we reviewed the Security Bulletins page of the Android website (https://source.android.com/security/bulletin). From here we selected a year and month from the dropdown on the left side of the webpage as shown in Fig. 3. Next, we searched for RCE updates (see Fig. 4) and clicked on the selected reference number which brought up the information page for that specific update. The information page for a sample RCE update is shown in Fig. 5. By selecting the "diff" link (highlighted in yellow in Fig. 5), a page containing the code that was added and/or changed to fix/mitigate the reported RCE vulnerability was displayed.

Following the data collection process described above, we collected a total of fifty-one (51) RCE updates/patches (including both the systems, Tomcat and Android). We analyzed these patches to understand if there are certain frequently used patterns in these RCE updates/patches. The data analysis conducted using the above-mentioned 51 RCE updates/patches is presented in Section IV. B.

IV. DATA ANALYSIS AND RESULTS

This section presents the results and findings obtained from analyzing the data collected during this study. This section is organized around the two research questions (RQs) that were described in Section III.A.

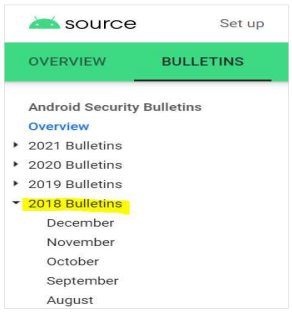

Fig.3. Android: Security Update Reports (Listed by Year)

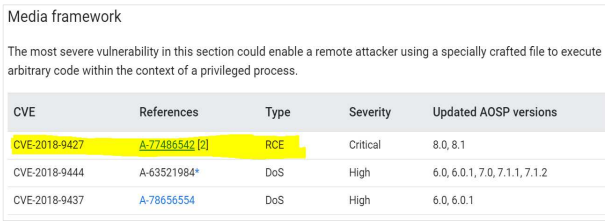

Fig.4. Android: Identifying RCE Vulnerabilities and their Respective Updates/Patches

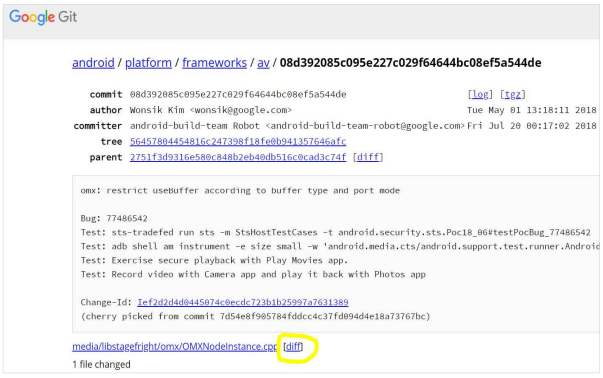

Fig.5. Android: Security Update Details for a Sample RCE Vulnerability found in Android

A. *RQ1: Do software systems suffer from Remote Code Execution (RCE) vulnerabilities more frequently when compared to the other types of security vulnerabilities?*

As mentioned before, we hypothesized that RCE vulnerabilities are the most frequently found vulnerabilities in software systems. In order to evaluate our hypothesis, we collected the vulnerability-count data from the "CVEDetails.com" datasource. This datasource receives its vulnerability data through National Vulnerability Database (NVD) xml feeds provided by NIST (National Institute of Standards and Technology). We collected the vulnerability-count data for the top-5 vulnerability types for the recent five (5) years, i.e., from the year 2015 to 2019 (please note that currently the CVEDetails datasource has vulnerability-count data till the year 2019).

Fig. 6 provides an overview of the data analyzed for RQ1. As can be seen in Fig. 6, RCE vulnerabilities were reported more frequently than other types of vulnerabilities during the years 2015, 2018, and 2019. Even during the other years (2016 and 2017), RCE vulnerabilities remained in the top-2 most reported vulnerabilities.

Furthermore, in the year 2019, the count of reported RCE vulnerabilities (2277) was significantly higher than the count of next most reported vulnerability (1593 Cross-Site Scripting

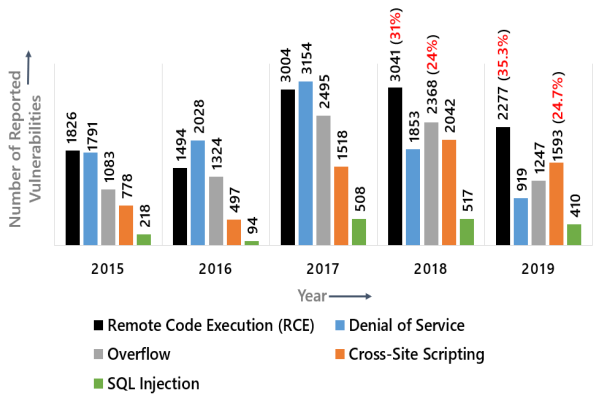

Fig. 6. Vulnerability-count by Type (for five recent years)

vulnerabilities were reported). Therefore, 35.3% of all the vulnerabilities reported during the year 2019 were of the type RCE (as can be seen in Fig. 6).

The data analysis described above and displayed in Fig. 6 clearly shows that software systems often suffer from RCE vulnerabilities more frequently than the other types of vulnerabilities. This in turn leads to a frequent need for programmers to fix RCE vulnerabilities through adding/editing lines of code in their software systems (i.e., adding updates/patches to mitigate the reported RCE vulnerabilities). This motivated us to identify some common patterns that are used by programmers when they are trying to fix the RCE vulnerabilities (with a focus on RCE vulnerabilities in Java-based software systems). The next section describes some of the patterns that we identified during this study.

B. *RQ2: What types of patches (i.e., code-fixes) are usually added to mitigate the known RCE vulnerabilities in Java-based software systems?*

As described in Section III.B (Data Collection), we identified a total of fifty-one (51) patches/updates that were made to fix RCE vulnerabilities in two Java-based software projects (Tomcat and Android). These patches are essentially changes/edits that made to lines of code to mitigate or fix a reported security vulnerability. Our primary goal with RQ2 was to identify and list some code-fix patterns that were used frequently to mitigate reported RCE vulnerabilities.

For a study such as ours, projects such as Tomcat and Android are a great resource as they highlight exactly what code changes/edits were made in order to fix a vulnerability. As an example, in Fig. 7, in order to fix the vulnerability titled CVE-2019-0232, the following variable was added: cgiServlet.invalidArgumentDecoded.

Similar to the process described above, we analyzed the updates/patches (i.e., code changes) that were applied to all fifty-one (51) RCE vulnerabilities that were part of this study.

```
    cgiServlet.find.found=Found CGI: name [{0}], path [{1}], script name [{2}
    cgiServlet.find.location=Looking for a file at [{0}]
    cgiServlet.find.path=CGI script requested at path [{0}] relative to CGI l
+   cgiServlet.invalidArgumentDecoded=The decoded command line argument [{0}]
    cgiServlet.invalidArgumentEncoded=The encoded command line argument [{0}]
    cgiServlet.runBadHeader=Bad header line [{0}]
    cgiServlet.runFail=I/O problems processing CGI
```

Fig. 7. Code Change (or Patch) Applied to Fix/Mitigate an RCE Vulnerability (CVE-2019-0232) in Apache Tomcat

Next, in order to find if there were similarities (i.e., patterns) between the coding changes that were made by the programmers in order to fix the RCE vulnerabilities, we converted the code-changes (patches/updates) into pseudocode. After converting the code-changes into pseudocode, we found that the code-changes (patches) could be classified into five (5) categories or patterns.

The five types of code-changes or updates (identified as a result of our analysis of 51 RCE vulnerabilities) that can be used to fix or mitigate a majority of RCE vulnerabilities are described as follows:

RCE Update Type 1 – *Check if the Packet Size is a Positive Integer*: For this update the programmer added an If Statement to check that the packet size was a positive integer (i.e., not negative or zero). That is, check if the source buffer contained enough bytes to copy the packet and check that the packet size does not exceed the destination buffer. The pseudocode for this type of update is shown in Table II.

TABLE II. RCE UPDATE TYPE 1

Pseudocode for RCE Update Type 1

if ((size <= 0) || ((read - sizeof(var1) - sizeof(var2)) < size) || (sizeof(msg) < size)) {
  return -1;
}

RCE Update Type 2 – *Checking for the Proper Variable Size*: Another common update made by the programmers was that they checked for the proper size. In this If statement, they check if the variable is greater than the max size. And if the variable is greater than the max size, then set the variable to the max size. The pseudocode for this type of update is shown in Table III.

TABLE III. RCE UPDATE TYPE 2

Pseudocode for RCE Update Type 2

if (result->num_val > MAX_ATTR_SIZE) {
  errorWriteLog;
  result->num_val = MAX_ATTR_SIZE;
}

RCE Update Type 3 – *Applying an Offset*: In many reported RCE vulnerabilities, it was found that the buffer is not properly calculated causing a memory overflow. To fix this, they adjusted the calculation by dividing the offset. The pseudocode for this type of update is shown in Table IV.

TABLE IV. RCE UPDATE TYPE 3

Pseudocode for RCE Update Type 3
display->buffer = buffer + (offset / FACTOR);

RCE Update Type 4 – In another commonly used patch/update for RCE vulnerabilities, the programmer moved the If Statement to the top. The intention is to run the fail-check before creating a new class and assigning the size. The pseudocode for this type of update is shown in Table V.

RCE Update Type 5 – *If statement to prevent out of bounds in the function*: In another common patch/update, the programmer added an If statement to check validity of pSettings->noOfPatches to prevent out of bounds in the function, which can also cause the memory size to be negative.

The pseudocode for this type of update (i.e., Update Type 5) is shown in Table VI.

As is evident from the above-mentioned five commonly used updates/patches, most of the code updates was to account for changes in size of boundaries and buffers. When the size was not properly accounted for, it caused errors in the application which in turn leads to a potential for bad actors to perpetrate a Remote Code Execution (RCE) attack.

Overall, we believe that the five update types that we have identified can provide a good starting point for programmers when they are trying to fix/mitigate reported RCE vulnerabilities in their Java-based software systems. We anticipate that a list of commonly used updates/patches (such as the one presented in this study) can improve the efficiency of programmers when they are trying to determine the best way to fix vulnerable code in their software system.

V. Threats to Validity

In this section, we describe the major threats to the validity of the results found in this study.

One major validity threat is to the generalizability of our results. This is because we have studied RCE vulnerabilities and their respective updates/patches in a limited number of systems (two systems, Tomcat and Android). Owing to this, even though we have been able to locate some viable fixes (i.e., updates/patches) for RCE, they may not resolve each and every RCE vulnerability. Through our study we saw that many instances of RCE vulnerabilities can be mitigated by fixing buffering and boundary issues. At this time, our research is limited to Tomcat and Android, and thus there are likely other instances of RCE vulnerabilities and their respective updates/patches that we have not come across in our research.

Another limitation arises from the unavailability of public vulnerability datasets for software projects. Most software projects do not make their vulnerabilities and related fixes public (with a few exceptions such as Android and Tomcat). The scarcity of vulnerable dataset makes any vulnerability related research challenging.

The authors also note that our findings do not guarantee prevention against an RCE attack from a malicious actor (i.e., attacker). Our goal is simply to provide a readily usable list of updates/patches that can be potentially employed for fixing RCE vulnerabilities. The final decision about using the most appropriate update/patch has to be made by the programmer by conducting a thorough evaluation of the vulnerability they are trying to fix.

VI. Discussion on Implication of Results

In this section we provide a brief discussion about the implications of our findings on programming practices that lead to injection of RCE vulnerabilities. Although, our main goal in this study was to identify what kind of code-changes (i.e., updates/patches) are commonly used to fix RCE vulnerabilities, our data analysis also highlighted some weaknesses or issues in coding practices (when the software systems are being developed). The paragraphs below provide a discussion related to such bad coding/programming practices that lead to injection of RCE vulnerabilities when the software is being developed.

Overall, we found that the vulnerability landscape for remote code execution (RCE) needs to be approached with a

TABLE V. RCE Update Type 4

Pseudocode for RCE Update Type 4

```
status = function();

      if (status != SUCCESS) {
         variable1 = NULL;
         return ERROR;
      }
variable1 = new class;
variable1->size = sizeof(object);
```

TABLE VI. RCE Update Type 5

Pseudocode for RCE Update Type 5

```
if (noOfPatches > 0) {
   int target = array[x].targetStartBand + array[x].numBandsInPatch;

   int size = (64 - target) * sizeof(FIXP_DBL);
   if (!useLP) {
      for (i = startSample; i < stopSampleClear; i++) {
         function1(&array2[i][target], size);
         function1(&array3[i][target], size);
      }
   } else
      for (i = startSample; i < stopSampleClear; i++) {
         function1(&array2[i][target], size);
      }
}
```

persistent and analytical approach. We must not only rely on advisories but also correlate the weakness types and attack vectors that are associated with each vulnerability type. Having such insight is meaningful in making informed decisions as well as prioritize each vulnerability based on their risk factor. Although not every RCE instance will have the same weakness type, we learned that some weakness types still correlate with the root causes that were found for the associated vulnerability. Our research was able to successfully identify a root cause (size of boundaries and buffers) that frequently leads to injection of RCE vulnerabilities.

Our research has shown that the opportunity for RCE vulnerabilities can be reduced by simply ensuring that buffers and boundaries are developed with proper sizing. With this is mind, developers can develop more efficient code and avoid at least some of the on-going RCE attacks being deployed by hackers worldwide. This research has also highlighted that some cognitive issues such as carelessness or inattention when coding can lead to vulnerability injection. We intend to integrate existing research [23, 24, 25, 26] on human cognition and human error in our future research on vulnerability prevention and vulnerability mitigation. Section VII further highlights our future research directions.

VII. Conclusion and Future Work

We have conducted a detailed analysis of the updates/patches (i.e., code-changes) that were applied by programmers to mitigate/fix fifty-one (51) RCE vulnerabilities reported in two Java-based software projects: Apache Tomcat and Android.

Based on our analysis, we proposed a list of five common updates/patches (see Table II through Table VI) that can be used to mitigate or fix a significant majority of RCE vulnerabilities in Java-based systems. We believe that our findings about these common RCE updates/patches can be

handy and readily-usable when programmers are trying to determine ways or means to fix RCE vulnerabilities in their own system. Therefore, we anticipate that our list of common RCE updates (shown in Tables II to VI) will help in reducing the time that is required by programmers to fix RCE vulnerabilities that have been reported in their system. To our knowledge, this is the first study of its kind that has focused on analyzing RCE vulnerabilities and their relevant updates/patches.

The results from this initial investigation are anticipated to be beneficial in reducing RCE attacks and hence the results motivate further research in the area. We plan to extend our research to other programming languages and systems to determine if such update (i.e., code-fix) patterns exist in systems coded in languages such as Python, PHP, C#, etc. Once this research is extended to other languages and systems, the natural evolution of this research is to study more vulnerabilities such as Elevation of Privilege, Information Disclosure, and SQL Injection in the future. In closing, our intent is to continue to learn about the nature of vulnerabilities and how to mitigate/fix them so that we may enhance our research to help prevent future exploits or attacks.

ACKNOWLEDGMENT

The co-authors on this paper, Dr. Kazi Zakia Sultana and Dr. Vaibhav Anu have grant funding from the National Science Foundation (NSF) of the USA through an REU grant.

REFERENCES

[1] Y. Zheng and X. Zhang, "Path sensitive static analysis of web applications for remote code execution vulnerability detection," *2013 35th International Conference on Software Engineering (ICSE)*, San Francisco, CA, USA, 2013, pp. 652-661

[2] D. Bates, A. Barth and C. Jackson, "Regular expressions considered harmful in client-side XSS filters," *In Proceedings of the 19th international conference on World wide web (WWW '10)*. NC, USA, 2010, pp. 91–100.

[3] M. V. Gundy and H. Chen, "Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-Site Scripting Attacks," *In Proceedings of the Network and Distributed System Security Symposium*, San Diego, California, USA, 2009.

[4] W. Halfond and A. Orso, "Preventing SQL injection attacks using AMNESIA," *In Proceedings of the 28th international conference on Software engineering*, Shanghai, China, 2006, pp. 795-798.

[5] M. T. Louw and V. N. Venkatakrishnan, "Blueprint: Robust Prevention of Cross-site Scripting Attacks for Existing Browsers," *2009 30th IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 2009, pp. 331-346.

[6] Y. Nadji, P. Saxena and D. Song, "Document Structure Integrity: A Robust Basis for Cross-site Scripting Defense," *In Proceedings of the Network and Distributed System Security Symposium*, San Diego, California, USA, 2009.

[7] G. Wassermann and Z. Su, "Sound and Precise Analysis of Web Applications for Injection Vulnerabilities," *In Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '07)*, San Diego, California, USA, 2007, pp. 32-41.

[8] Todd Jackson, Babak Salamat, Gregor Wagner, Christian Wimmer, and Michael Franz, "On the effectiveness of multi-variant program execution for vulnerability detection and prevention," *In Proceedings of the 6th International Workshop on Security Measurements and Metrics (MetriSec '10)*, Bolzano, Italy, 2010, pp. 1-8.

[9] H. Holm, T. Sommestad, U. Franke and M. Ekstedt. "Success Rate of Remote Code Execution Attacks - Expert Assessments and Observations," *Journal of Universal Computer Science*, vol. 18, pp. 732-749, 2012.

[10] S. Biswas, M. Sohel, M. Sajal, Md. Mizanur, T. Afrin, T. Bhuiyan, and M. Hassan, "A Study on Remote Code Execution Vulnerability in Web Applications," *International Conference on Cyber Security and Computer Science (ICONCS'18)*, Oct 18-20, 2018, Safranbolu, Turkey.

[11] Q. H. Mahmoud, D. Kauling and S. Zanin, "Hidden android permissions: Remote code execution and shell access using a live wallpaper," *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, 2017, pp. 599- 600.

[12] S. Mohammad and S. Pourdavar, "Penetration test: A case study on remote command execution security hole," *2010 Fifth International Conference on Digital Information Management (ICDIM)*, Thunder Bay, ON, 2010, pp. 412-416.

[13] L. Zhang, H. Zhang, X. Zhang and L. Chen, "A New Mechanism for Trusted Code Remote Execution," *2007 International Conference on Computational Intelligence and Security Workshops (CISW 2007)*, Heilongjiang, 2007, pp. 574-578.

[14] M. Carlisle and B. Fagin, "IRONSIDES: DNS with no single-packet denial of service or remote code execution vulnerabilities," *2012 IEEE Global Communications Conference (GLOBECOM)*, Anaheim, CA, 2012, pp. 839-844.

[15] D. Votipka, R. Stevens, E. M. Redmiles, J. Hu, and M. L. Mazurek, "Hackers vs. Testers: A Comparison of SoftwareVulnerability Discovery Processes," *2018 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2018, pp. 374-391.

[16] A. Austin and L. Williams, "One Technique Is Not Enough: A Comparison of Vulnerability Discovery Techniques," *2011 International Symposium on Empirical Software Engineering and Measurement,* Banff, AB, Canada, 2011, pp. 97-106.

[17] A. Kiezun, V. Ganesh, P. J. Guo, P. Hooimeijer and M. D. Ernst. "HAMPI: a solver for string constraints," *ACM Trans. Softw. Eng. Methodol.*, vol. 21, no. 4, Article 25, 2013.

[18] W. Halfond, S. Anand and A. Orso. "Precise Interface Identification to Improve Testing and Analysis of Web Applications," *In Proceedings of the eighteenth international symposium on Software testing and analysis (ISSTA '09),* Chicago, IL, USA, 2009, pp. 285-296.

[19] Dinis Cruz, "OWASP O2 Platform - Open Platform for Automating Application Security Knowledge and Workflows", *Web Application Security Conference*, 2010, pp. 5-5.

[20] S. Tyagi and K. Kumar, "Evaluation of Static Web Vulnerability Analysis Tools," *2018 Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, Solan, India, 2018, pp. 1-6.

[21] Y. Xie and A. Aiken. "Saturn: A scalable framework for error detection using Boolean satisfiability," In ACM Trans. Program. Lang. Syst. May, 2007, vol. 29, no. 3, pp. 16–es.

[22] M. Das, S. Lerner, M. Seigel. "ESP: path-sensitive program verification in polynomial time," *In Proceedings of the ACM SIGPLAN 2002 conference on Programming language design and implementation (PLDI '02),* 2002, Berlin, Germany, pp. 57-68.

[23] V. Anu, G. Walia, W. Hu, J. C. Carver and G. Bradshaw, "Using a Cognitive Psychology Perspective on Errors to Improve Requirements Quality: An Empirical Investigation," *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, Ottawa, ON, Canada, 2016, pp. 65-76.

[24] W. Hu, J.C. Carver, V. Anu et al., "Using human error information for error prevention", *Empir. Software Eng.*, vol. 23, pp. 3768-3800, 2018.

[25] V. Anu, G. Walia, W. Hu, J. C. Carver and G. Bradshaw, "Issues and Opportunities for Human Error-Based Requirements Inspections: An Exploratory Study," *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Toronto, ON, Canada, 2017, pp. 460-465.

[26] V. Anu, K. Z. Sultana and B. K. Samanthula, "A Human Error Based Approach to Understanding Programmer-Induced Software Vulnerabilities," *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Coimbra, Portugal, 2020, pp. 49-54.