OPEN ACCESS Freely available online

Journal Information Technology & Software Engineering

Research Article

Software Self-Healing Mechanism to Mitigate Security Vulnerabilities Using CI/CD Pipeline

Shumaila Hussain[1*], Junaid Baber[2], Muhammad Nadeem[3], Shariqa Fakhar[2]

[1]Department of Applied Computer Science, Sardar Bahadur Khan Women's University, Quetta, Pakistan; [2]Department of Applied Computer Science, University of Baluchistan, Quetta, Pakistan; [3]Department of Applied Computer Science, Habib University, Karachi, Pakistan

ABSTRACT

The software security vulnerabilities are reported frequently by CWE. These vulnerabilities result in huge financial loss to technological industry due to patches development and redistribution to handle the arising vulnerabilities. In this study we have proposed a platform or language independent software self-healing mechanism using CI/CD pipeline and CWE guidelines to automatically mitigate the software security vulnerabilities. We have selected improper input validation security vulnerability to implement the proposed mechanism. The improper input validation is listed at 4th position among the top 25 most impactful vulnerabilities by CWE. The prototype developed using the proposed software self-healing mechanism is capable of identifying the vulnerabilities and automatically healing them. The proposed software self-healing mechanism is cost effective and efficient way to mitigate the software security vulnerabilities.

Keywords: Improper input validation vulnerabilities; Software self-healing; Software security vulnerabilities; CI/CD pipeline; CWE guidelines

INTRODUCTION

Technological advancement has increased the complexity of the systems, making it difficult to tackle software vulnerabilities. Software vulnerabilities are weaknesses, flaws, or unwanted operations in the software system. It is very important to mitigate these security vulnerabilities to make the software reliable and efficient. The traditional method of mitigating software vulnerability takes too much human effort, cost, time, and expertise.

According to the research exploration about the financial impact of software security vulnerabilities in US by the National Institute of Standards and Technology (NIST) the US economy loses about $60 billion USD every year for patches development and redistribution to handle the arising software security vulnerabilities [1,2].

In 2012, Knight capital, lost $400 million USD just because of a security vulnerability in the system [3]. Similarly, the virus attacks like love bug, code red wannacry etc. each had an impact of 1 billion to 10 billion dollars [4,5]. The financial loss due to software errors; made the buyers rethink the huge investments in software [6]. The concept of patch development to heal vulnerabilities is another most adopted concept. The research found that 34% of the security patches developed triggered 52% of new security problems and these patches are incomplete and do not fully secure systems [7]. The research was conducted to analyze the effect of patches on the maintainability of the product using Better Code Hub and find it too harsh and not robust enough [8]. Another research regarding the maintainability of the patches says that the security patches can cause a negative impact on the maintainability of the product moreover, the developers need to pay more focus on maintainability while developing the patches for improper input validation vulnerability. He further added that the patch risk assessment should be integrated with CI/CD pipeline to work effectively [9].

Correspondence to: Shumaila Hussain, Department of Applied Computer Science, Sardar Bahadur Khan Women's University, Quetta, Pakistan, Tel: 03457801456; E-mail: shumaila.hussain@sbkwu.edu.pk

Received: 23-Dec-2022, Manuscript No. JITSE-23-21151; **Editor assigned:** 26-Dec-2022, PreQC No. JITSE-23-21151 (PQ); **Reviewed:** 09-Jan-2023, QC No. JITSE-23-21151; **Revised:** 20-Mar-2023, Manuscript No. JITSE-23-21151 (R); **Published:** 27-Mar-2023, DOI: 10.35248/2165-7866.23.13.332

Citation: Hussain S, Baber J, Nadeem M, Fakhar S (2023) Software Self-Healing Mechanism to Mitigate Security Vulnerabilities using CI/CD Pipeline. J Inform Tech Softw Eng. 13:332.

The concept of autonomous computing has brought the idea of software self-healing in order to avoid human involvement in mitigating software vulnerabilities.

The term software self-healing was first introduced by IBM in 2001, the concept involves a software system mechanism that is capable of identifying that the system is not working correctly and can adjust the fault automatically without human intervention [10]. This concept helped to gain the user's trust, and reduced time, cost, human effort, and expertise.

There exist public repositories to be aware of the security vulnerabilities and know the possible techniques to avoid such vulnerabilities. The CWE is a public repository known as Common Weakness and Enumeration (CWE); this includes comprehensive literature about the most commonly arising security vulnerabilities in software.

The CWE is owned by MITRE corporation it was established to strengthen software security, and to help software organizations and security expert's cop with the vulnerabilities. Thus this can be a great authentic source to mitigate common software security vulnerabilities.

Along with adopting new mechanisms to mitigate security vulnerabilities the process of software development SDLC showed its vital role in building successful software. Considering many important aspects of the software development processes the researchers come up with many solutions grabbing the attention towards continuous checks for security concerns in each SDLC stage and thus improving the software development process in order to maintain good quality software [11]. As time passes it seemed that the software development techniques have made the software development process very intricate and these complexities have given rise to security loopholes in each stage that are taken interest by hackers. It is therefore that a software engineering technique with continuous checks came up known as continuous integration.

In software engineering, Continuous Integration (CI) is the practice of merging all developers' working copies to share mainline several times a day [12]. In continuous integration development environments, software engineers frequently integrate new or changed code with the mainline code [13]. The software vulnerabilities need to be checked in every stage of SDLC in order to minimize the risk. Thus applying the process of continuous integration and the concept of self-healing together to mitigate security vulnerabilities can ensure reliable software.

Motivation and related work

The most common and oldest approach to finding software vulnerabilities is manual source code auditing. The technique involves reading the source code and finding the vulnerabilities, this is done by the team of security experts who work on the vulnerable software patch, it involves human effort, cost, and time and requires expertise [14]. The traditional software vulnerability checking techniques cannot cope with new software security vulnerabilities arising on daily basis obtained from specific conditions and change dynamically at run-time.

The concept of autonomic computing has brought up drastic changes in the technological world. Autonomic systems are able to reconfigure and optimize themselves in the unpredictable situation; these systems can recover and heal security vulnerabilities and can protect themselves from external attacks [15].

Autonomic computing has brought the idea of self-healing systems and these self-healing systems can successfully address the problems of software security vulnerabilities; and can increase software reliability [16,17].

The self-healing reduces human involvement in mitigating vulnerabilities; they have lowered the vulnerability maintenance costs and improved the existing vulnerabilities mitigation techniques [18].

The self-healing has been induced in the computing systems in many different ways. It has been implemented in the architecture of the system. The concept comprises of changing the architecture of the system so that the system attains self-healing capabilities [19]. The software self-healing systems adopting the architectural based models work with the concept of vulnerability identification and reconfiguration of the system by the external architectural based model.

The architectural based self-healing mechanism is performance based and focus the performance of external architectural components it is therefore difficult to identify the internal state of the system. This mechanism does not take any information from the internal system modules and it is difficult to develop variety of self-healing strategies without internal information of the system. It is not sufficient enough for the mitigation of the security vulnerabilities to make changes in the targeted system based on the information gathered by external architectural model only.

The component based self-healing mainly focus the internal components of the software system and make the components capable of identifying, and resolving the vulnerabilities at the component design stage. This technique divides the software system in to two parts or layers the service layer and the self-healing layer or component. The service layer is responsible to provide the intended services while the self-healing layer is supposed to monitor the activities of service layer identify the vulnerability and reconfigure them.

The component based self-healing mechanism focuses on events related to internal states and cannot determine internal status problems due to lack of resource and thus cannot effectively handle the security vulnerabilities.

Log-based self-healing approach enforces the self-healing by maintaining the log files generated by all different kind of software and auto correcting them. The vulnerability can only be identified if it is logged it is therefore difficult to analyze the vulnerability if the event is yet to be logged. The log mechanism focuses on the vulnerability after the event it is difficult to model constraints for monitoring, accessing, and healing based on the internal view of system if the event is unlogged. Moreover; just from the log we cannot identify the real impact of the vulnerability on the software.

The concept of roll back has also helped in making the system capable of self-healing. The rollback is helpful when the non-deterministic failures occur in the software system the system is then rolling back to the consistent state. The roll back approach is helpful in temporary basis as it does not fix and identify the vulnerability but temporarily provides a relief from the failure in the system.

The self-healing has not only been ensured by above mentioned techniques but the researchers tried to come up with other solutions to enable self-healing in the software. It has been identified that paying attention towards self-detecting can bring up the capability of self-healing in a software system. The self-detection and self-healing traditionally focused on the test-case generation and code analysis and skips the run time monitoring and checks which is very important to make the system reliable.

Similarly; another technique to temporarily provide relief was that after the occurrence of identification or unidentified security vulnerability the system is enforced to re-execute the failing part of the system under the controlled environment. Yet another way to temporary relief from the vulnerabilities was to handle the program execution smartly. The system tends to skip the execution of the faulty paths a tool FastFix was developed for this purpose.

The concept of using finite state automata has also been implemented to ensure the system reliability. The finite automata was used to identify the constraints of the faulty event. The concept of system reboot to avoid the failure has also been implemented to cop up with security vulnerabilities which again provides temporary relief.

The micro-reboots were the software resigned on this concept. Another way to enforce the self-healing capabilities is agent based self-healing. In this technique the concept of artificial intelligence is used and the software/hardware agents are developed to automatically heal the security vulnerabilities. This technique is implemented in various ways to ensure the self-healing mechanism like it is used for security and protection in monitoring using distributed network environment, in the grid computing and using cloud infrastructure. Recently genetic programming as a way to automatically fix software faults has been introduced (Table 1).

Table 1: The comparative analysis of software self-healing mechanisms to mitigate software security vulnerabilities.

Approach	**Description**
Architecture based	It works on the external architectural components of the system.
Component based	This technique divides the software system in to two parts or layers the service layer and the self-healing layer. The service layer is responsible to provide intended services while the self-healing layer is responsible to monitor the activities of service layer, identify the vulnerability and reconfigure them.
Roll back	The rollback is used when the non-deterministic failures occur in the software system the system is then rolling back to the consistent state.
Smart execution	The system tends to skip the execution of the faulty paths a tool Fastfix was developed for this purpose.
Micro-reboot	When the fault is identified the system tends to reboot the faulty component.
Agent based	In this technique the concept of artificial intelligence is used and the software/hardware agents are developed to automatically detect and heal the security vulnerability.
Self-detection of vulnerabilities	It focuses on test-case generation and code analysis or run time monitoring and check to make system reliable. Software self-checks the security vulnerabilities using machine learning.
Finite state automata	The finite state automata was used to identify the constraints of the faulty event.
System reboot	The concept of system reboot to avoid the failure has also been implemented to cop up with security vulnerabilities.

Genetic programming	Genetic programming is applying natural genetic process to the specific program. It uses heuristic or hill climbing search algorithm to search optimal solution.
Monitoring illegal control flow	It focused on the illegal transfer of control flow. Monitoring the execution of the control flow and managing them helped removing the security vulnerability.
Using rescue points	In case of any fault the system is provided with rescue points to adopt the path with rescue point for smooth execution.

The input validation is the assurance of accepting only required format or authorized data from all input sources in any computing application. The proper input validation prevents the malformed data to persist in the database that triggers the malfunctioning of the downstream components. The malicious input can include unauthorized input as command, piece of code and scripts to penetrate in the workflow of an application and harm it. The input validation should necessarily be performed on either synaptic or semantic level of any application. The improper input validation is one of the major cause of security vulnerabilities in any computing application. The improper input validation can trigger the SQL injection attack, cross-site scripting (XSS) attack, and buffer overflows, XML external entity attacks (XXE), directory traversal attacks and denial of services attacks. The CWE which is a project of MITRE organization and aims to be a dictionary of software weaknesses. It displays the weaknesses occurred in different software and list the weaknesses according to their severity.

According to Common Weakness Enumeration (CWE) in 2021 the input validation is reported at 4th place in the list of top 25 most occurring and dangerous security vulnerabilities.

The MITRE is one of the organizations like, SANS institute and OWASP that focuses on the awareness, importance and improvement of the security vulnerabilities. These organizations keep records and publish the security vulnerabilities occurring on daily basis in different software applications. The projects of MITRE Common Vulnerabilities and Exposure (CVE) which is a vulnerability identification system that 'aims to provide a unique CVE identification to publicly known security vulnerabilities and links the vulnerability databases and other capabilities while the CWE not only publish and provides a repository of the vulnerabilities but helps the programmers by providing the mitigation techniques of the security vulnerability as well. These repositories help the researchers and programmers to work on the vulnerabilities and developed the tools such as static code analysis dynamic tainting, combination of dynamic tainting and static analysis.

The Table 2 below shows the approaches used to detect and mitigate the security vulnerabilities raised due to improper input validation from 2010 to 2022 (Table 2).

Table 2: Approaches used to detect and mitigate the improper input validation security vulnerabilities from 2010 to 2022.

Vulnerability	Approach
Cross site scripting attacks	Detection of cross site scripting attack using modified CNN model.
	The client side execution-flow based method using finite-state automata to detect cross-site scripting attacks.
	Automated server-side XSS attacks detection using boundary injection.
	TT-XSS: A novel taint tracking based dynamic detection framework for DOM cross-site scripting.
	Detecting blind cross-site scripting attacks using SVM.
	Reducing attack surface corresponding to type 1 cross-site scripting attacks using secure development life cycle practices.
	DeepXSS: Cross site scripting detection based on deep learning LSTM recurrent neural network.
	Resolving cross-site scripting attacks through genetic algorithm and reinforcement learning.

	Cross-site scripting attack detection using machine learning with hybrid features.
	Detection of cross-site scripting attacks using dynamic analysis and fuzzy inference system.
	Preventing XSS attacks using application firewall.
	Preventing client side XSS by tracking the flow of information in browser.
	Preventing XSS attacks using preventive measures along with tainted information flow graph to audit the XSS attacks.
	Repairing XSS using unit testing.
	Noxes: Protects the information leakage to repair cross site scripting attack on client side.
Buffer overflow attacks	Automatic repair method for D2D communication routing buffer overflow vulnerability in cellular network using buffer overflow repair algorithm based on homomorphic analysis of node residual energy.
	Detecting return to libc buffer overflow attacks using network intrusion detection systems.
	Preventing buffer overflow attacks by memory randomization.
	Exploiting stack based buffer overflow using data execution prevention and address space layout randomization.
	Using string library function for detecting integer-overflow to buffer-overflow vulnerability.
	Automatic prevention of buffer overflow vulnerability using candidate code generation.
	Buffer overflow attack with multiple fault injection and a proven countermeasure using Address Space Layout Randomization (ASLR), Stack Guard (SG), Data Execution Prevention (DEP), Exec Shield (ES) and Input Size Limitation (ISL)
	Rupair: A data flow analysis algorithm for RUST code.
	Performed static buffer overflow detection and suggested for automatic detection.
	Proposed a compiler to automatically fix the integer overflow to buffer over flow vulnerability.
	Static buffer overflow detection and repair using bovlnspector tool.
	Automatically generating patch in C language using satisfiability modulo theories SMT to fix buffer overflow.
SQL injection attacks	Detection of SQL injection attack using decision tree.
	SQLrand: Preventing SQL injection attacks using instruction set randomization.

	An approach for SQL injection detection based on behavior and response analysis.
	A heuristic based approach for detecting SQL injection vulnerabilities in web applications.
	SQL injection attacks detection and prevention based on neuro fuzzy technique
	Prevention of SQL injection attacks to login page of a website application using prepared statement technique
	An algorithm for detecting SQL injection vulnerability using black-box testing.
	DIAVA: A traffic-based framework for detection of SQL injection attacks and vulnerability analysis of leaked data.
	A hybrid method consists of augmented database tables with symbols then an algorithm for queries and another algorithm designed for string matching is used for detection and prevention of SQL injection attacks.
	A second-order SQL injection detection method using instruction set randomization.
	Train the dataset, cluster them; similar clusters are matched and suggestions are generated to fix SQL injection attacks.
	Tool is developed to detect the SQL injection attacks and displays suggestions to fix it.
	Replacing the vulnerable API with secure API using java to fix SQL injection attack.
	Fixing the SQL injection vulnerable web application firewall using machine learning with multi-objective genetic algorithm.
	An algorithm to replace the SQL statements with prepared statements to fix SQL injection vulnerability.

MATERIALS AND METHODS

We have analyzed the traditional techniques of mitigating the software security vulnerabilities and identified the financial loss and difficulties in patches integration. Moreover; the existing techniques to mitigate the security vulnerabilities are also discussed. The research design of the study is shown in the Figure 1 below.

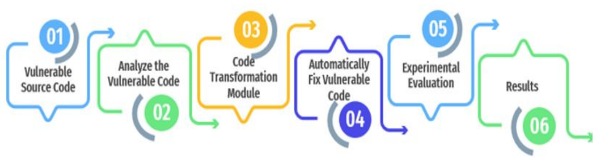

01
Vulnerable Source Code
Analyze the Vulnerable Code
02
03
Code Transformation Module
Automatically Fix Vulnerable Code
04
05
Experimental Evaluation
Results
06

Figure 1: Research design.

The vulnerable source code is selected and analyzed for security vulnerabilities a code transformation module transforms the codes and automatically heal the raised software security vulnerability.

The diagram of the proposed mechanism is given below (Figure 2).

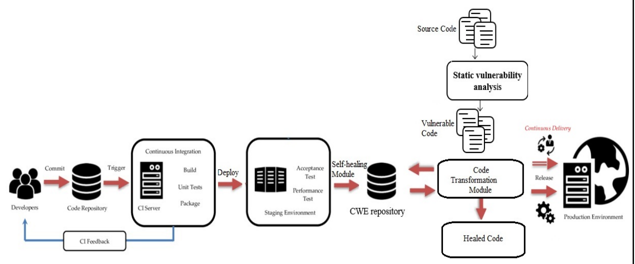


Figure 2: Proposed mechanism for software self-healing.

The proposed mechanism uses the CI/CD pipeline and CWE recommendations to mitigate the identified software security vulnerability.

The CI/CD pipeline is briefly explained in the diagram below (Figure 3).

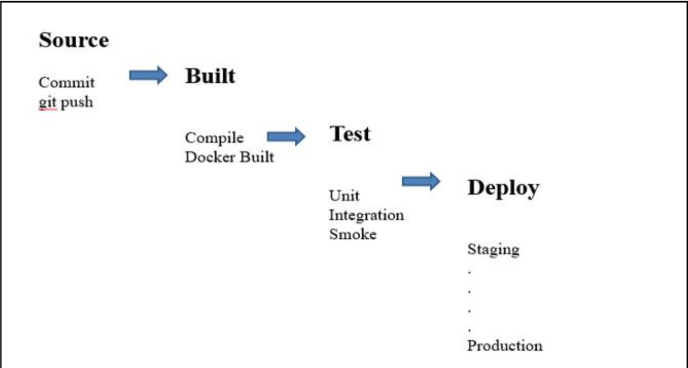
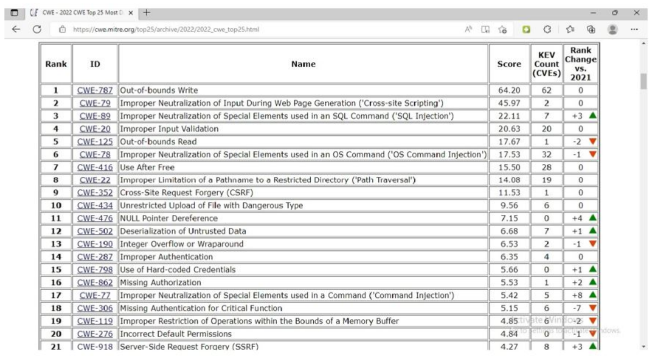


Figure 3: Continuous integration and continuous deployment pipeline.

The source stage is where commit in the source code triggers a notification to the CI/CD tool, which runs the corresponding pipeline. The application is then built to deployable instance or package. The test stage is responsible to perform unit test, integration test and smoke test on code. The code is finally deployed to the production.

Once the product reaches the deployment stage it is exposed to the CWE repository. The CWE is a project of CVE by MITRE. It is data dictionary of computing system flaws. It provides a standard measurement for software security tools. The code transition module is responsible to automatically heal the identified security vulnerability.

We have developed a prototype implementing the proposed software self-healing mechanism. The improper input validations security vulnerability is selected to be self-healed by proposed mechanism. The improper input validation vulnerability is at the fourth place among the top 25 most impactful security vulnerabilities in the CWE list (Figure 4).


CWE - 2022 CWE Top 25 Most D
https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html

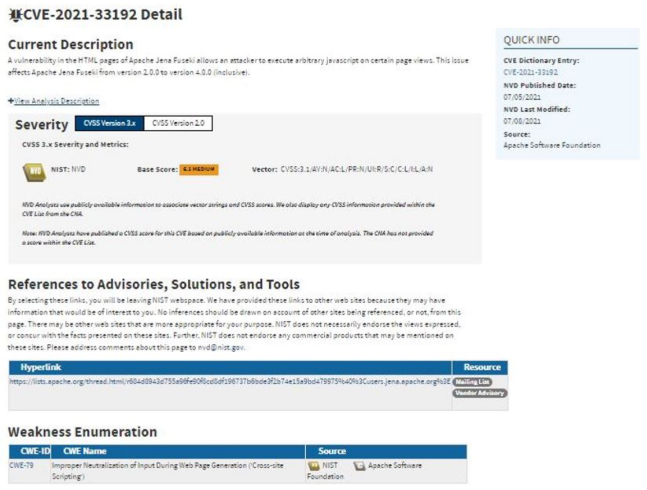
Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲

Figure 4: Position of improper input validation vulnerability among top 25 security vulnerabilities on Common Weakness Enumeration (CWE).

The vulnerable source code including improper input validation vulnerability is identified using CVE. The CVE stands for common vulnerabilities and exposures it is project of MITRE that classifies the security vulnerabilities. The prototypes self-heal the improper input validation vulnerabilities from the source code using the proposed mechanism.

RESULTS AND DISCUSSION

We have developed a prototype implementing the proposed mechanism to prove the self-healing concept. The open source software Apache Jena Fuseli is listed for improper input validation vulnerability by CVE. The Common Vulnerabilities and Exposures (CVE) is a project of MITRE corporations. It is a list of publicly disclosed computer security flaws. Each security flaw that's listed in CVE is assigned a CVE ID number to uniquely identify each security flaw (Figure 5).



CVE-2021-33192 Detail

Current Description

A vulnerability in the HTML pages of Apache Jena Fuseki allows an attacker to execute arbitrary javascript on certain page views. This issue affects Apache Jena Fuseki from version 2.0.0 to version 4.0.0 (inclusive).

+View Analysis Description

Severity [✓ CVSS Version 3.x] CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD Base Score: 6.1 MEDIUM Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

QUICK INFO

CVE Dictionary Entry: CVE-2021-33192

NVD Published Date: 07/05/2021

NVD Last Modified: 07/08/2021

Source: Apache Software Foundation

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

Hyperlink	Resource
https://lists.apache.org/thread.html/r684d8943d755a96fe90f8cd8df196737b6bde3f2b74e15a9bd479975%40%3Cusers.jena.apache.org%3E	Mailing List Vendor Advisory

Weakness Enumeration

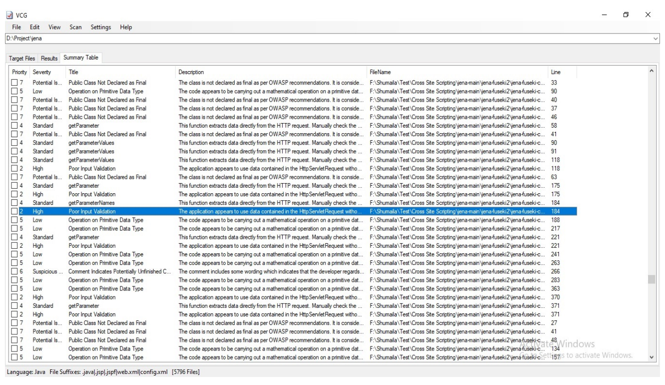
CWE-ID	CWE Name	Source
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	NIST Apache Software Foundation

Figure 5: Improper input validation issue in Apache Jena Fuseki listed by Common Vulnerabilities and Exposures (CVE).

The Apache Jena Fuseki 2.0.0 to 4.0.0 includes the improper input validation vulnerability in their source code. The Apache Jena Fuseki 2.0.0 is selected as vulnerable source code to induce the self-healing mechanism. Apache Jena Fuseki is a SPARQL Server. It can be run as operating system service, as a java web application (WAR file), and as a standalone server.

In order to analyze the improper input validation vulnerability in Apache Jena Fusiki the static code analysis tool is used. The static code analysis tools or Static Application Security Testing (SAST). It is the process of analyzing software security vulnerabilities in selected software.

The visual code grepper is selected for static code analysis as it is a powerful tool that can perform complex security vulnerabilities check on code with thousands of code files. The Apache Jena Fusiki is large software with thousands lines of code. The Apache Jena Fusiki is analyzed for reported input validation vulnerability using static analysis tool called Visual Code Grepper (VCG). The Visual Code Grepper (VCG) identified multiple improper input validation vulnerabilities in the Jena Fusiki (Figure 6).



Priority	Severity	Title	Description	FileName	Line
☐ 7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is conside...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	33
☐ 5	Low	Operation on Primitive Data Type	The code appears to be carrying out a mathematical operation on a primitive dat...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	90
☐ 7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is conside...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	40
☐ 7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is conside...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	37
☐ 7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is conside...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	46
☐ 4	Standard	getParameter	This function extracts data directly from the HTTP request. Manually check the ...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	58
☐ 7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is conside...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	41
☐ 4	Standard	getParameterValues	This function extracts data directly from the HTTP request. Manually check the ...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	90
☐ 4	Standard	getParameterValues	This function extracts data directly from the HTTP request. Manually check the ...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	91
☐ 4	Standard	getParameterValues	This function extracts data directly from the HTTP request. Manually check the ...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	118
☐ 2	High	Poor Input Validation	The application appears to use data contained in the HttpServletRequest witho...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	118
☐ 7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is conside...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	63
☐ 4	Standard	getParameter	This function extracts data directly from the HTTP request. Manually check the ...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	175
☐ 2	High	Poor Input Validation	The application appears to use data contained in the HttpServletRequest witho...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	175
☐ 4	Standard	getParameterNames	This function extracts data directly from the HTTP request. Manually check the ...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	184
☐ 2	High	Poor Input Validation	The application appears to use data contained in the HttpServletRequest witho...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	184
☐ 5	Low	Operation on Primitive Data Type	The code appears to be carrying out a mathematical operation on a primitive dat...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	188
☐ 5	Low	Operation on Primitive Data Type	The code appears to be carrying out a mathematical operation on a primitive dat...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	217
☐ 4	Standard	getParameter	This function extracts data directly from the HTTP request. Manually check the ...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	221
☐ 2	High	Poor Input Validation	The application appears to use data contained in the HttpServletRequest witho...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	221
☐ 5	Low	Operation on Primitive Data Type	The code appears to be carrying out a mathematical operation on a primitive dat...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	241
☐ 5	Low	Operation on Primitive Data Type	The code appears to be carrying out a mathematical operation on a primitive dat...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	263
☐ 6	Suspicious ...	Comment Indicates Potentially Unfinished C...	The comment includes some wording which indicates that the developer regards...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	266
☐ 5	Low	Operation on Primitive Data Type	The code appears to be carrying out a mathematical operation on a primitive dat...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	283
☐ 5	Low	Operation on Primitive Data Type	The code appears to be carrying out a mathematical operation on a primitive dat...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	363
☐ 2	High	Poor Input Validation	The application appears to use data contained in the HttpServletRequest witho...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	370
☐ 4	Standard	getParameter	This function extracts data directly from the HTTP request. Manually check the ...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	371
☐ 2	High	Poor Input Validation	The application appears to use data contained in the HttpServletRequest witho...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	371
☐ 7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is conside...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	27
☐ 7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is conside...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	41
☐ 7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is conside...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	48
☐ 5	Low	Operation on Primitive Data Type	The code appears to be carrying out a mathematical operation on a primitive dat...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	134
☐ 5	Low	Operation on Primitive Data Type	The code appears to be carrying out a mathematical operation on a primitive dat...	F:\Shumaila\Test\Cross Site Scripting\jena-main\jena-fuseki2\jena-fuseki-c...	157

Language: Java File Suffixes: .java|.jsp|.jspf|web.xml|config.xml [5796 Files]

Figure 6: Improper input validation issue in Apache Jena Fuseki listed by visual code grepper.

The proposed software self-healing mechanism is used to mitigate the selected security vulnerability. We have developed a prototype using the proposed self-healing mechanism. The prototype is capable of identifying and automatically mitigating the security vulnerabilities using CI/CD pipeline and CWE guidelines (Figures 7 and 8).

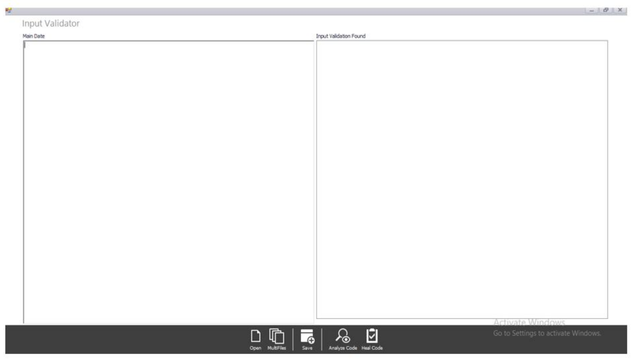
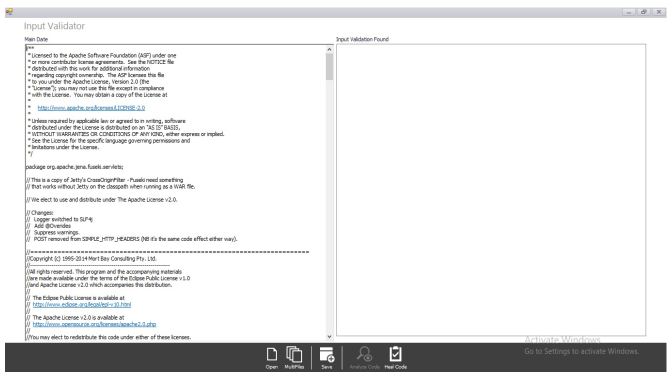


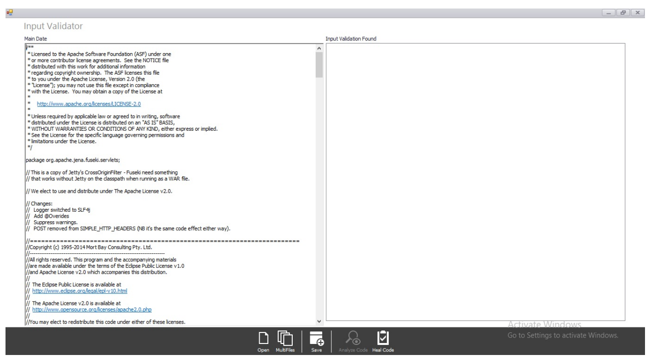
Figure 7: Interface of developed prototype using proposed self-healing mechanism.



```
/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.jena.fuseki.servlets;

// This is a copy of Jetty's CrossOriginFilter - Fuseki need something
// that works without Jetty on the classpath when running as a WAR file.

// We elect to use and distribute under The Apache License v2.0.

// Changes:
//   Logger switched to SLF4j
//   Add @Overides
//   Suppress warnings.
//   POST removed from SIMPLE_HTTP_HEADERS (NB it's the same code effect either way).

//========================================================================
//Copyright (c) 1995-2014 Mort Bay Consulting Pty. Ltd.
//------------------------------------------------------------------------
//All rights reserved. This program and the accompanying materials
//are made available under the terms of the Eclipse Public License v1.0
//and Apache License v2.0 which accompanies this distribution.
//
//  The Eclipse Public License is available at
//  http://www.eclipse.org/legal/epl-v10.html
//
//  The Apache License v2.0 is available at
//  http://www.opensource.org/licenses/apache2.0.php
//
//You may elect to redistribute this code under either of these licenses.
```

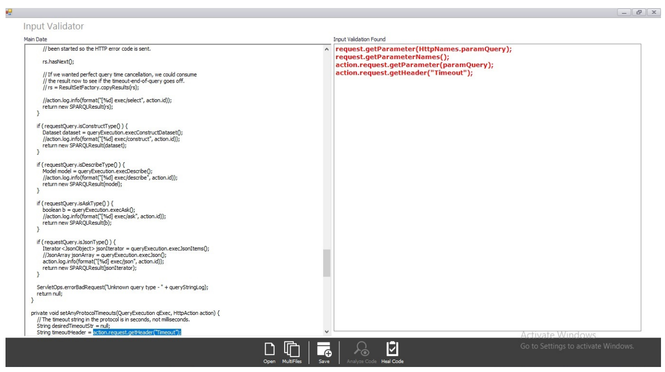
Figure 8: Prototype opens the source code of Apache Jena Fuseki.

The prototype is provided with the source code of Apache Jena Fuseki to analyze the security vulnerabilities (Figures 9 and 10).



```
/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.jena.fuseki.servlets;

// This is a copy of Jetty's CrossOriginFilter - Fuseki need something
// that works without Jetty on the classpath when running as a WAR file.

// We elect to use and distribute under The Apache License v2.0.

// Changes:
//   Logger switched to SLF4j
//   Add @Overides
//   Suppress warnings.
//   POST removed from SIMPLE_HTTP_HEADERS (NB it's the same code effect either way).

//========================================================================
//Copyright (c) 1995-2014 Mort Bay Consulting Pty. Ltd.
//------------------------------------------------------------------------
//All rights reserved. This program and the accompanying materials
//are made available under the terms of the Eclipse Public License v1.0
//and Apache License v2.0 which accompanies this distribution.
//
//  The Eclipse Public License is available at
//  http://www.eclipse.org/legal/epl-v10.html
//
//  The Apache License v2.0 is available at
//  http://www.opensource.org/licenses/apache2.0.php
//
//You may elect to redistribute this code under either of these licenses.
```

Figure 9: Prototype analyses the source code of Apache Jena Fuseki.

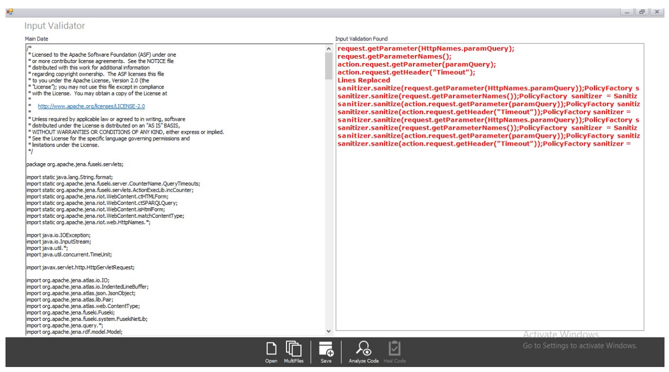


```
            //been started so the HTTP error code is sent.

            rs.hasNext();

            // If we wanted perfect query time cancellation, we could consume
            // the result now to see if the timeout-end-of-query goes off.
            // rs = ResultSetFactory.copyResults(rs);

            //action.log.info(format("[%d] exec/select", action.id));
            return new SPARQLResult(rs);
        }

        if ( requestQuery.isConstructType() ) {
            Dataset dataset = queryExecution.execConstructDataset();
            //action.log.info(format("[%d] exec/construct", action.id));
            return new SPARQLResult(dataset);
        }

        if ( requestQuery.isDescribeType() ) {
            Model model = queryExecution.execDescribe();
            //action.log.info(format("[%d] exec/describe", action.id));
            return new SPARQLResult(model);
        }

        if ( requestQuery.isAskType() ) {
            boolean b = queryExecution.execAsk();
            //action.log.info(format("[%d] exec/ask", action.id));
            return new SPARQLResult(b);
        }

        if ( requestQuery.isJsonType() ) {
            Iterator<JsonObject> jsonIterator = queryExecution.execJsonItems();
            //JsonArray jsonArray = queryExecution.execJson();
            action.log.info(format("[%d] exec/json", action.id));
            return new SPARQLResult(jsonIterator);
        }

        ServletOps.errorBadRequest("Unknown query type - " + queryStringLog);
        return null;
    }

    private void setAnyProtocolTimeouts(QueryExecution qExec, HttpAction action) {
        // The timeout string in the protocol is in seconds, not milliseconds.
        String desiredTimeoutStr = null;
        String timeoutHeader = action.request.getHeader("Timeout");
```

```
request.getParameter(HttpNames.paramQuery);
request.getParameterNames();
action.request.getParameter(paramQuery);
action.request.getHeader("Timeout");
```

Figure 10: Prototype identifies improper input validation vulnerabilities from the source code of Apache Jena Fuseki.

We have selected the improper input validation security vulnerability identified in Apache Jena Fuseki 2.0.0 (Figure 11).

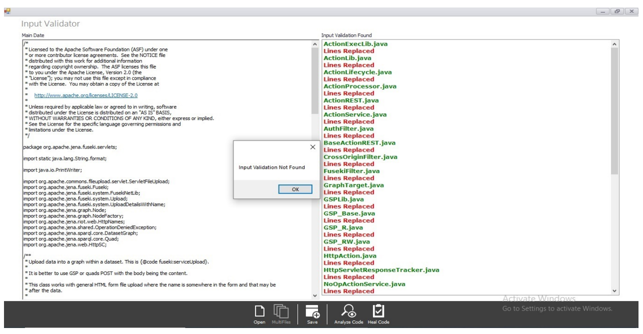


```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.jena.fuseki.servlets;

import static java.lang.String.format;
import static org.apache.jena.fuseki.server.CounterName.QueryTimeouts;
import static org.apache.jena.fuseki.servlets.ActionExecLib.incCounter;
import static org.apache.jena.riot.WebContent.ctHTMLForm;
import static org.apache.jena.riot.WebContent.ctSPARQLQuery;
import static org.apache.jena.riot.WebContent.isHtmlForm;
import static org.apache.jena.riot.WebContent.matchContentType;
import static org.apache.jena.riot.web.HttpNames.*;

import java.io.IOException;
import java.io.InputStream;
import java.util.*;
import java.util.concurrent.TimeUnit;

import javax.servlet.http.HttpServletRequest;

import org.apache.jena.atlas.io.IO;
import org.apache.jena.atlas.io.IndentedLineBuffer;
import org.apache.jena.atlas.json.JsonObject;
import org.apache.jena.atlas.lib.Pair;
import org.apache.jena.atlas.web.ContentType;
import org.apache.jena.fuseki.Fuseki;
import org.apache.jena.fuseki.system.FusekiNetLib;
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
```

```
request.getParameter(HttpNames.paramQuery);
request.getParameterNames();
action.request.getParameter(paramQuery);
action.request.getHeader("Timeout");
Lines Replaced
sanitizer.sanitize(request.getParameter(HttpNames.paramQuery));PolicyFactory s
sanitizer.sanitize(request.getParameterNames());PolicyFactory  sanitizer  = Sanitiz
sanitizer.sanitize(action.request.getParameter(paramQuery));PolicyFactory sanitiz
sanitizer.sanitize(action.request.getHeader("Timeout"));PolicyFactory sanitizer =
sanitizer.sanitize(request.getParameter(HttpNames.paramQuery));PolicyFactory s
sanitizer.sanitize(request.getParameterNames());PolicyFactory  sanitizer  = Sanitiz
sanitizer.sanitize(action.request.getParameter(paramQuery));PolicyFactory sanitiz
sanitizer.sanitize(action.request.getHeader("Timeout"));PolicyFactory sanitizer =
```

Figure 11: Prototype self-heals the improper input validation vulnerabilities from the identified source code of Apache Jena Fuseki.

The prototype analyzes the type and cause of improper input validation vulnerability and matches with the guidelines provided by CWE repository and automatically heal the code according to the CWE guidelines for the vulnerability (Figure 12).



```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.jena.fuseki.servlets;

import static java.lang.String.format;

import java.io.PrintWriter;

import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.apache.jena.fuseki.Fuseki;
import org.apache.jena.fuseki.system.FusekiNetLib;
import org.apache.jena.fuseki.system.Upload;
import org.apache.jena.fuseki.system.UploadDetailsWithName;
import org.apache.jena.graph.Node;
import org.apache.jena.graph.NodeFactory;
import org.apache.jena.riot.web.HttpNames;
import org.apache.jena.shared.OperationDeniedException;
import org.apache.jena.sparql.core.DatasetGraph;
import org.apache.jena.sparql.core.Quad;
import org.apache.jena.web.HttpSC;

/**
 * Upload data into a graph within a dataset. This is {@code fuseki:serviceUpload}.
 *
 * It is better to use GSP or quads POST with the body being the content.
 *
 * This class works with general HTML form file upload where the name is somewhere in the form and that may be
 * after the data.
 *
```

```
Input Validation Not Found
OK
```

```
ActionExecLib.java
Lines Replaced
ActionLib.java
Lines Replaced
ActionLifecycle.java
Lines Replaced
ActionProcessor.java
Lines Replaced
ActionREST.java
Lines Replaced
ActionService.java
Lines Replaced
AuthFilter.java
Lines Replaced
BaseActionREST.java
Lines Replaced
CrossOriginFilter.java
Lines Replaced
FusekiFilter.java
Lines Replaced
GraphTarget.java
Lines Replaced
GSPLib.java
Lines Replaced
GSP_Base.java
Lines Replaced
GSP_R.java
Lines Replaced
GSP_RW.java
Lines Replaced
HttpAction.java
Lines Replaced
HttpServletResponseTracker.java
Lines Replaced
NoOpActionService.java
Lines Replaced
```

Figure 12: The source code of Apache Jena Fuseki is self-healed.

In order to validate the mitigation of input validation security vulnerabilities from Apache Jena Fuseki the software is tested again for the vulnerability and the results shows that the prototype developed using proposed software self-healing mechanism has automatically mitigated the security vulnerabilities. The proposed software self-healing mechanism can be implemented to self-heal any software security vulnerability.

CONCLUSION

In this study we have proposed a software self-healing mechanism using CI/CD pipeline and CWE guidelines to automatically mitigate the vulnerabilities. The CWE repository is used to identify the security vulnerabilities in software and provides a baseline to mitigate and prevent the identified vulnerabilities. We have selected input validation vulnerability placed at the 4th position in the list of top 25 most impactful software security vulnerabilities on CWE. The open source software Apache Jena Fuseki 2.0.0 is selected as it is listed as vulnerable to input validation.

The prototype is developed using the proposed software self-healing mechanism to prove the concept. The prototype is developed using CI/CD pipeline using security vulnerability prevention and mitigation guidelines from CWE repository. The proposed mechanism is capable of self-healing any security vulnerability. Any software developed using proposed mechanism is capable of self-healing any software security vulnerability.

Technological advancement has raised the risk of security vulnerabilities. The latest technologies are more complex and are associated with new security risks. The software security vulnerabilities are arising on daily basis and it is difficult to address all security vulnerabilities at once. We have selected improper input validation vulnerability any other security vulnerability can be selected in future to be mitigated.

REFERENCES

1. Planning S. The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology, 2002.
2. Zhivich M, Cunningham RK. The real cost of software errors. IEEE Security and Privacy. 2009;7(2):87-90.
3. Strasburg J, Bunge J. Loss swamps trading firm. Wall Str J Mag. 2012;1-5.
4. Geppert L. Lost radio contact leaves pilots on their own. IEEE spectrum. 2004;41(11):16-17.
5. Berr J. Wannacry-ransomware-attacks-wannacry-virus-losses. 2017.
6. Chen Y, Chen J, Gao Y, Chen D, Tang Y. Research on software failure analysis and quality management model. IEEE Explore,16-20 July, Lisbon, Portugal, 2008;94-99.
7. Li F, Paxson V. A large-scale empirical study of security patches. In proceedings of the 2017 ACM SIGSAC conference on computer and communications security, Berkeley, CA, USA, 2017.
8. Olivari M. Maintainable production, a model of developer productivity based on source code contributions. Maintainable production. 2017;1-106.
9. Reis S, Abreu R, Cruz L. Fixing vulnerabilities potentially hinders maintainability. Empir Softw Eng. 2021;26:1-27.
10. Batarseh FA, Gonzalez AJ. Predicting failures in agile software development through data analytics. Softw Qual J. 2018;26:49-66.
11. Fowler M, Foemmel M. Continuous integration. 2006.
12. Piantadosi V, Scalabrino S, Oliveto R. Fixing of security vulnerabilities in open source projects: A case study of Apache http server and Apache tomcat. IEEE Explore, 22-27 April, Xi'an, China, 2019;68-78.
13. Kephart JO, Chess DM. The vision of autonomic computing. Computer. 2003;36(1):41-50.
14. Arcuri A, Yao X. A novel co-evolutionary approach to automatic software bug fixing. IEEE Explore, Hong Kong, China, 2008;162-168.
15. Gorla A, Pezze M, Wuttke J, Mariani L, Pastore F. Achieving cost-effective software reliability through self-healing. Comput Inform. 2010;29(1):93-115.
16. Schneider C, Barker A, Dobson S. A survey of self-healing systems frameworks. Softw Pract Exp. 2015;45(10):1375-1398.
17. Dashofy EM, van der Hoek A, Taylor RN. Towards architecture based self-healing systems. In proceedings of the first workshop on Self-healing systems, New York, NY, United States, 2002;21-26.
18. Garlan D, Schmerl B. Model based adaptation for self-healing systems. In proceedings of the first workshop on self-healing systems, Pittsburgh, PA, 2002;27-32.
19. Garlan D, Cheng SW, Huang AC, Schmerl B, Steenkiste P. Rainbow: Architecture based self-adaptation with reusable infrastructure. Computer. 2004;37(10):46-54.