ICIC Express Letters
Volume **13**, Number **7**, July **2019**

 ISSN 1881-803X
pp. **557–567**

USING PUBLIC VULNERABILITIES DATA TO SELF-HEAL SECURITY ISSUES IN SOFTWARE SYSTEMS

Attiq ur Rehman Jaffar[1], Muhammad Nadeem[1], Mamdouh Alenezi[2] and Yasir Javed[2]

[1]Computer Engineering Department
Balochistan University of Information Technology Engineering and Management Sciences
Airport Road, Baleli, Quetta 87300, Pakistan
attiq.jaffar@hotmail.com; dr.nadeem@ieee.org

[2]College of Computer and Information Sciences
Prince Sultan University
P.O.Box No. 66833 Rafha Street, Riyadh 11586, Saudi Arabia
{ malenezi; yjaved }@psu.edu.sa

Received December 2018; accepted February 2019

Abstract. *Hackers are worryingly becoming able to trick both naive and well-informed users into becoming security victims. Software applications nowadays have become more complex due to the increased interaction of the user, the limitless need for innovative capabilities, the use of open source, and third-party libraries known to be vulnerable. With each passing day, the techniques used to hack systems become ever more ingenious. Therefore, individual users and companies must prepare and raise their shields to safeguard their data and reputation. Web-applications are the main streams these days with regard to software systems requiring an inclusive approach to both prevent and reduce security weaknesses. During Cross-Site Scripting (XSS) vulnerabilities exploration, we assume that all security measures can fail and security depends on multiple levels of mechanisms that cover errors. Achieving this goal is possible by applying different methods and tools that can ensure security during or after the development of the application. This paper examines the possibility of using public vulnerability repositories in helping to suggest mitigation techniques in Common Weakness Enumeration (CWE) articles. The paper proposes a framework that can transform a vulnerable code to secure code. The evaluation suggests that the proposed system can successfully remove the vulnerable code by applying articles from the public vulnerability repository. Application of framework on two projects shows the removal of above 90% vulnerabilities after detection. The prototype implementation showed that the vulnerable code could be transformed into a secure code automatically without human intervention.*

Keywords: Self-healing, CWE, Secure systems, Vulnerable code, Security defects

1. **Introduction.** The software development process has become a very complex endeavor, which led to ignoring several security measures. Web-applications are the main streams these days with regard to software systems and are stored on a remote machine, which can be accessed globally. The architecture of this type of applications is based on client-server and can be accessed with any standard Internet browser by multiple clients or in a hybrid environment (desktop or Web services). It is not necessary to develop and test it in all possible configurations and versions of the operating system. This makes it far easier to develop and troubleshoot this kind of applications.

Software attacks have increased exponentially after 2014 in form of denial of service, Web application or cyber espionage [1,2]. It is observed that most companies invest less in applications and products and the percentages of breaches have increased.

DOI: 10.24507/icicel.13.07.557

The ubiquity of Web browsers, the ability to update and manage Web applications without the distribution and installation of software on thousands of computers and their cross-platform compatibility were the main reasons for the popularity of Web applications [3]. The evolution of Web technology greatly improves the variety of services offered on the Web. However, the modern development of Web applications presents many security challenges, which are often not taken into account seriously. Although techniques such as threat analysis are increasingly recognized as essential for software development, there are also other practices that can be adopted while developing software applications.

Cyberwar has led to change of attacks over government or countries. Attacks on common installation, leakage of data through national centers, manipulating political results are to name few [4]. Some of the common reasons for Web attacks are vulnerable software design, code defects and default installations [5,6]. The main challenge facing the security community is to understand software security weaknesses by communicating a common language that is reported by different vendors. This will help discover more in-depth knowledge for any vulnerability and attack patterns found in software. The only way to succeed in application security is to use a process that continuously analyzes and evaluates new threats, evolves and establishes defenses and monitors those defenses to ensure their effectiveness.

The main reason for having vulnerable software is because of the lack of knowledge at software architects or developers. There have been substantial Web security breaches in corporate, military, e-commerce and banking sectors. Such breaches not only affect these sectors financially but also destroy their public reputations [4,7].

It is very essential to introduce new methods that can alleviate such risks and make Web-applications less prone to cyber-attacks. Web application developers have begun to think about intrinsic security that builds security throughout the Software Development Life Cycle (SDLC). SDLC is commonly used by the software industry to produce an in time and high-quality software solution after rigorous testing. Most Web application security testing efforts are concentrated around penetration testing which is an art based on hacker's awareness who is obsessed with figuring out new exploits to hack your application, thought process and determination to exploit software vulnerabilities. In this paper, we introduce a framework that takes input from public vulnerabilities data repositories and provides suggestions to remove vulnerabilities. It also will provide auto-correction based on mitigation techniques to remove any vulnerability existing in code.

In Section 2 literature review is presented in form of division into security requirements, attacks, and validation techniques. Section 3 presents the software self-healing framework that can be used to reduce the vulnerabilities such as cross-site scripting from the code. Section 4 presents the research methodology and research questions to be addressed. Section 5 presents the results obtained from the proposed framework and analysis of the results. Section 6 presents the conclusions.

2. **Literature Review.**

2.1. **Web application security.**

Web applications have introduced a number of new issues that were not even there with static websites. Moreover, due to the openness in a Web application, it has become a practice battle group for checking any new kind of attacks by even script kiddies or newbies. The attacks are ever evolving while the technologies have not evolved, thus leaving a gap between the application and attackers [8,9]. Due to the awareness among the developers and some new technologies, some of the critical attacks have vanished from the surface. Web application infrastructure requires in-depth knowledge of handling security at each stage. Identification of security vulnerabilities requires not only good tools but also good security practices [10]. Security community suggested to use white box analysis for detection of security issues or other

practices such as black box testing scanners check Web application security for different vulnerabilities [11].

Checking for security issues is an essential part and ensures that at least the major vulnerabilities are not there. Understanding the specific Web vulnerabilities is a key element of the application that should be considered in all organizations and should be the main domain of penetration assessors, code reviewers and other security professionals [12].

2.2. **Cross-site scripting or XSS attack.** Cross-site scripting, also known as XSS attack, is a type of attack that can be performed to engage users of a website. Exploiting an XSS error allows attackers to inject scripts into Web pages viewed by users [13]. Although there is no specific classification of XSS attacks as they depend on the context of the attack where malicious script or file can be sent in a shape of file or input field researchers normally classify XSS attack into three categories Type-0 attacks, Type-1 attacks, and Type-2 attacks. These are briefly described below.

- Type-0 attack is also known as Document Object Model (DOM) based attacks [13], in which the attacker injects the script on the client side and this is often imitated by convincing a victim to click on a certain link containing malicious code. During this attack pattern, the attacker does not need to send the payload to the server rather executing it on the client's browser.
- Type-1 attack, also known as Stored or Persistent XSS, in which the attacking hacker inserts malicious code into Web pages and then stores it on the Web server for future use. The payload can be returned to other users who requested the stored information from an infected Web page and are executed in its context [14]. The severity of the attack is directly proportional to the number of users visiting the compromised Web server. Applications such as social networking, forums, and blogs are the major victims to initiate this type of attacks.
- Type-2 attacks are also known as a Reflected XSS. During this attack, pattern hacker creates a link in form of website ads or by sending it in victims' emails asking them to reset their passwords by opening that link. A request sent by the user to the server is not executed in the context of that server rather sending the user a malicious code that executes in victim browser. After executing malicious script in the browser, the hacker can get control of the user's session and can steal sensitive information [14,15].

2.3. **Input validation.** Input validation attacks constitute most of the vulnerabilities. Hence, a strong mechanism is required to implement those validation rules, although it may vary from one application to another or based on the choice of languages used by the developer [16]. It is also believed that unfiltered input is responsible for the wide range of vulnerabilities. There are different methods that have been employed to avoid unfiltered data on the client-side or server-side [17].

Some known practices are adopted by the programmers to filter such input data prior to processing it. Figure 1 shows an example of white list for PHP language where alphanumeric characters, white spaces or any other specific inputs are allowed. This type of list confirms that user-supplied input is safe to process. In the same way, blacklist contains all those patterns, which are not allowed, and the program will validate everything else.

Validating the user input for any malicious activity is easy but may lead too many false positive results to process all incoming data that user may enter. There may be another way to address input validations issues as described earlier; i.e., it depends on the choice of programming language preferred by developers. HtmlLawed [18] is one of the tools which is PHP based used to remove cross-site scripting and neutralize the HTML code. Due to static in nature, it is also recommended to revise these lists regularly as the range

```php
<?php
function xss_whitelist($input, $limit = null, $offset = 0)
{
  // Force input to be a string0
        $x = (string) $input;

        // Allow alphanumeric characters, whitespace, and specific characters
        $x = preg_replace("/[^a-zA-Z0-9 -:,.!?\/|]/", "",$x);

        // Limit characters
        if ($limit) {
                $x = substr($x, $offset, $limit);
        }

        // Convert characters to HTML entities and return the sanitized string
        return htmlentities($x, ENT_QUOTES, 'UTF-8');
}
?>
```

FIGURE 1. XSS whitelist for PHP language that shows a safe input check

of input data may change over time since it also helps to decrease the false positive or false negative results.

3. **Software Self-Healing Framework.** Software framework can be defined as a set of abstract rules or libraries on which applications can be built and can help to reduce the overall complexity of the application during its development phase. Application frameworks not only increase the performance of the developer but can also help to produce complex applications in a given period [19]. The selection of the framework depends on different factors such as platform support, the type of platform they support, programming language preferences, developer availability, or the cost of software licenses.

3.1. **Proposed framework.** In the proposed framework shown in Figure 2, we will explain the different modules. The reason behind the selection of a public repository is that they can help us to modify the code by utilizing the mitigation reported in public vulnerabilities database. We discussed the framework by dividing it into a public repository, static analysis and code transformation for further details.

3.2. **Public vulnerability repositories.** In this section, we analyzed the list of different vulnerability repositories, which can further help us to choose the type of repository to mitigate some of our focused vulnerabilities. Software buyers want to ensure that software products that they purchase are reviewed for known types of security breaches and software companies making it part of their future agreements. The basic goal of the vulnerabilities repository is to classify the security-related issues so that the developers can be timely informed to overcome security flaws. For this goal, different security organizations started centralized databases to address those flaws and develop a mutual consensus on the basic taxonomy of security flaws [20].

3.3. **National vulnerability database.** This repository is developed by the United States (US) government. Vulnerability management information is based on standards represented by the use of content automation protocol for security. This information allows automation of vulnerability management, security measures and compliance. National

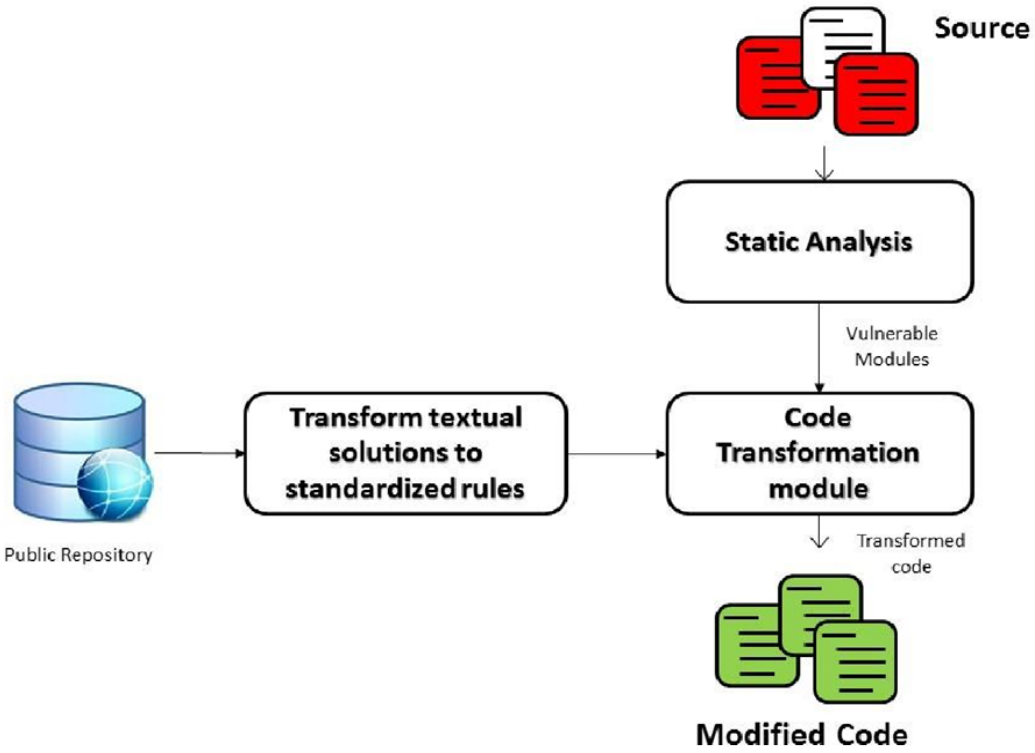

FIGURE 2. Proposed software self-healing framework for identification and removal of software security vulnerabilities

Vulnerability Database (NVD) includes security checklist databases, security software defects, incorrect configurations, product names and impact metrics. Security Content Automation Protocol (SCAP) provides a detailed low-level guide to establish security for operating systems and applications [21].

3.4. **Common Weakness Enumeration (CWE).** CWE was launched early 2006 as a result of a workshop hosted by the NIST in 2005. The goal of the workshop was to analyze the security diagnostic tools and detect security problems. The motive behind the CWE was to improve software quality by introducing common reference dataset of reported vulnerabilities within the source code of the application and to provide common mitigation techniques for reported flaws [22]. CWE is assembled and updated by various international groups of security experts and academic institutions to explain the complexity of content. CWE provides a standardized vocabulary for software developers to specify potential risks and their proposed solutions by the security community. Each section has its own significance to understand the nature of weaknesses during the life cycle of any application. There are over 700+ weaknesses listed on the CWE website [23].

A thorough survey suggests that there are a number of other vulnerability repositories including security focus, WPscan and OSVDB, etc. [23]. Every repository has its own benefits and issues in terms of understanding software weaknesses. Some of these, such as WPscan focus solely on Word press applications or specific to the platform [24]. We found that CWE repository covers a wide range of reported vulnerabilities, most of which are discussed with their potential mitigations. A major focus of our research is those articles which are related to coding issues. After analyzing a dataset of articles provided by CWE, we conclude them based on their weaknesses abstraction type in Table 1.

3.5. **Static analysis tools.** These tools scan applications code for possible errors without executing them as illustrated in Figure 4. Employing these tools provides the first line of defense against any security threat. Finding bugs in the source code with a static

TABLE 1. CWE articles analysis showing weakness type in class level, base level and variant abstraction

Description	Research concepts	Development concepts	Architectural concepts
Class weaknesses abstraction type	89	85	42
Base weaknesses abstraction type	328	328	117
Variant weaknesses abstraction type	289	289	61
Compound weaknesses abstraction type	8	4	3
Major categories	0	42	12
Total listed articles	714	707	223

analysis has been a challenging task for the security community that is conducted through a number of tools to understand the flagged warnings or errors generated by these tools which is also another daunting task for the code reviewer. Researchers also recognize the impact of false positive or false negative reports [25] generated by the static analysis tools which can also reduce the quality of code analysis.

As many factors need to be considered during code analysis, such as programming language in which static analysis tools are built. Answers should be found for how well they are aligned with the type of software being developed. How fast the development team can adapt the possible security threats recommended by the tools and, finally yet importantly, how well these tools integrate with the SDLC during development process [26,27].

Figure 3 shows that with the help of a static analysis tool, the framework checks the source code for possible weaknesses that are vulnerable to security attacks. Analysis report generated by the static analysis will help us evaluate the performance of our framework and understand how the flagged warnings or vulnerabilities are mitigated with the proposed methodology.

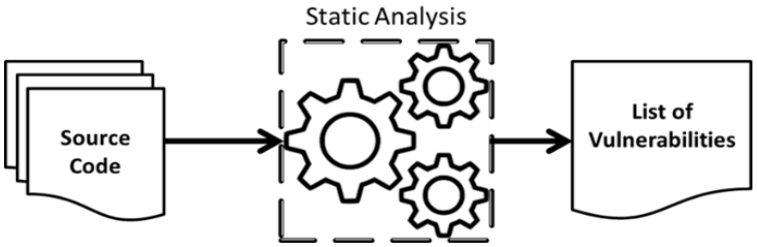

FIGURE 3. Static analysis workflow

3.6. **Code transformation module.** Applying changes after scanning the source code for possible security vulnerabilities will be the core module in our framework. Modification of code in a vulnerable source code file can produce unexpected results and may compromise the overall execution of the application.

4. **Research Methodology.** In this section, we discuss our purposed approach by answering the following Research Questions (RQs).

RQ 1: What is a software self-healing framework, which utilizes public repositories?

In the proposed framework, the CWE articles are well explained and we can analyze their data from different angles such as the development concepts, research concepts and architectural concept.

RQ 2: How can the mitigation strategies in the CWE repository be transformed into standardized rules?

We studied a different type of vulnerabilities from the CWE repository where it proposes different mitigation solutions on different levels of the application development.

RQ 3: How can the standardized rules, discussed in the previous question, be used to transform vulnerable code to secure code?

As we discussed in RQ2, the mitigation techniques explained in the articles can be converted into rules as we will translate those suggestions with the help of regular expression. In Figure 4, we showed that when we select the article CWE-79 and transformed the textual solutions to standard rules, it can ensure that the XSS attack will not occur again.

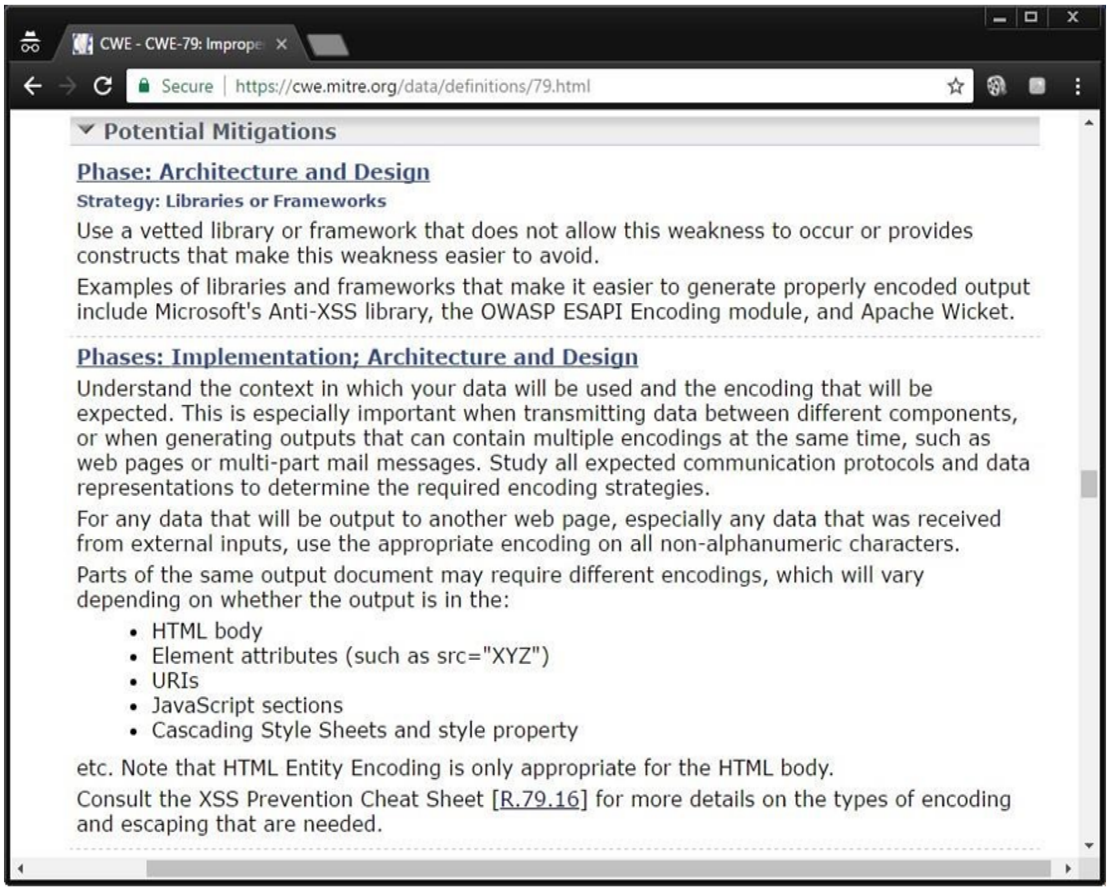

FIGURE 4. CWE-79 article showing the possible mitigation technique for handling design to prevent XSS attack

RQ 4: How is the performance of the proposed framework in terms of resolution of security issues?

This has been answered in the experiments and results section where the proposed framework shows a solution of around 83% of the issues.

5. **Experiments and Results.** In this section, the login form is analyzed to check the vulnerabilities and how the insertion of secure code can make the code secure.

Step 1: We are scanning a standard Login form for any vulnerability. The following sample page contains the different components such as Textbox controls and Button control. The following page, illustrated in Figure 5, was developed in Asp.net 4.0, and the basic structure of the code is distributed in two files, namely, Login.aspx and Login.aspx.cs.

In Figure 6, the controls being used belong to the input type, which is the first entry for a hacker to insert their malicious code.

Step 2: In this step, we observed a CWE-79 article where it focuses on input validation and we convert it to rule for this specific input element. The article explains one of the

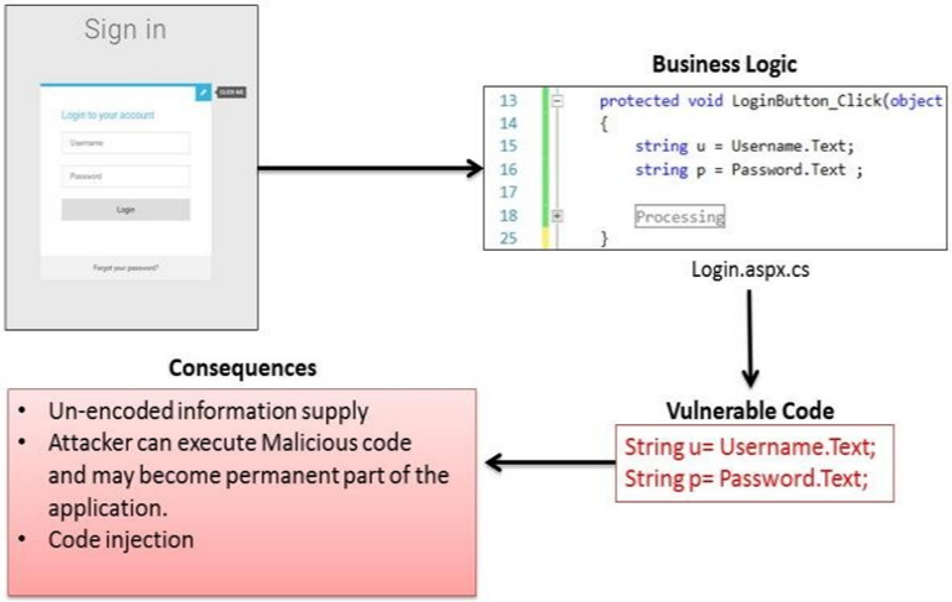

FIGURE 5. Showing the sample test page for the Login form

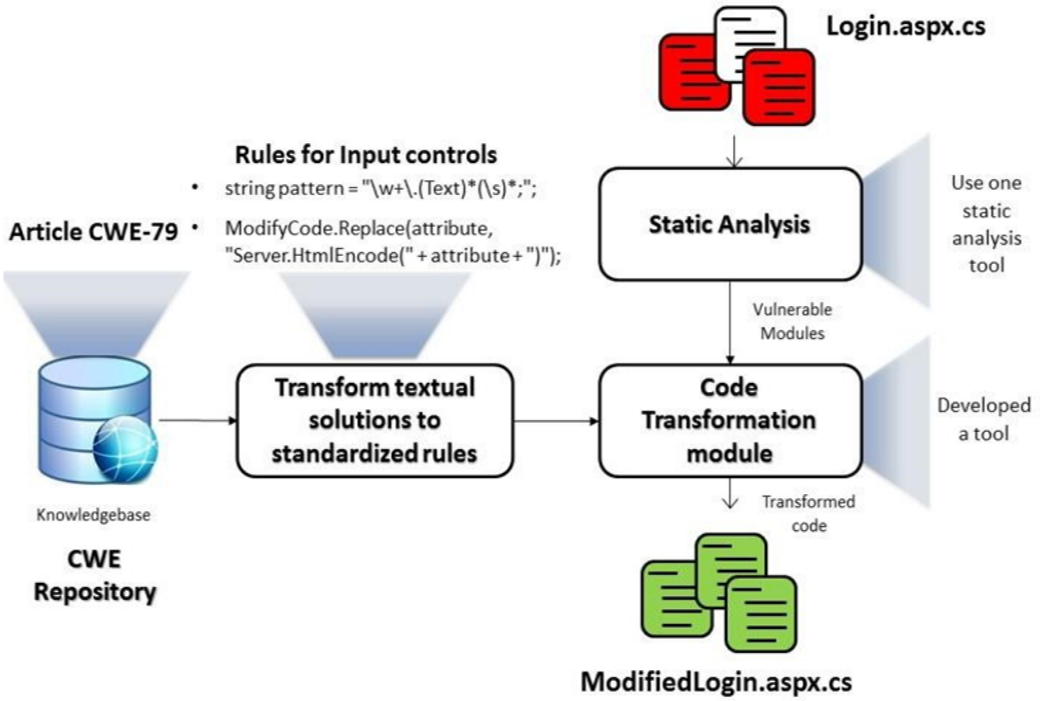

FIGURE 6. Framework processes showing how article CWE-79 is used to extract rules and applied on Login form to do input validation

mitigation strategies for input that is if the input controls are not encoded before sending user data to the server, it may give chance to the attacker to execute some malicious codes on the server-side or client-side.

As mentioned before, the rules we are proposing are language specific as in this case we configure rule specific to asp.net language discussed in Step 1.

In this case, we write a rule for all the input controls in Asp.net to check whether the controls are properly encoded as per CWE-79 proposed mitigation solution. We transformed the rule for encoding mitigation by extracting the input controls with the help of regular expressions as described in Figure 7.

```
17      private List<string> GetVulnerableCode(string htmlString)
18      {
19          List<string> Collection = new List<string>();
20
21          string pattern = @"\w+\.(Text)*(\s)*;";
22          Regex rgx = new Regex(pattern, RegexOptions.IgnoreCase);
23          MatchCollection matches = rgx.Matches(htmlString);
24
25          for (int i = 0, l = matches.Count; i < l; i++)
26          {
27              Collection.Add(matches[i].Value);
28          }
29
30          return Collection;
31      }
32  }
```

FIGURE 7. Input rule for checking vulnerable code

```
var regex = new RegExp("\\w{1,}\\.(Text){0,}(\\s){0,};", "g"); var testString
= ""; // fill this in var match;
while ((match = regex.exec(testString)) != null) {
 // javascript RegExp has a bug when the match has length 0    if
(match.index === regex.lastIndex) {
   ++regex.lastIndex;
 }
 // the match variable is an array that contains the matching groups }
```

Vulnerable Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Text.RegularExpressions;
public partial class Login : System.Web.UI.Page
{ protected void Page_Load(object sender, EventArgs e)
{

}
protected void LoginButton_Click(object sender, EventArgs e)
{
string u = Username.Text;
string p = Password.Text ;

#region Processing


#endregion
}
}
```

Healed Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Text.RegularExpressions;
public partial class Login : System.Web.UI.Page
{ protected void Page_Load(object sender, EventArgs e)
{

}
protected void LoginButton_Click(object sender, EventArgs e)
{
string u = Server.HtmlEncode(Username.Text);
string p = Server.HtmlEncode(Password.Text) ;

#region Processing


#endregion
}
}
```

String u= Username.Text; Converted to Server.HtmlEncode(Username.Text)
String p= Password.Text; Converted to Server.HtmlEncode(Password.Text)

FIGURE 8. The proposed healed code

In Figure 8, we described a rule to collect all input text controls, which are not encoded. The modified code saved in a new file as ModifyLogin.aspx.cs, which can be integrated with the existing project solution.

In Table 2, it is shown that for any specific type of code elements for input and response.write() are scanned and vulnerable elements are resolved. The probability of resolving vulnerable codes is based on filters applied and current practices reported in the

TABLE 2. Results for application of a framework for two Web resources

Domain	Pages scanned	LoC	Response.Write found	Textbox	Resolved	Syntax error	Total
Carskilla.com	19	3550	12	35	39	3	47
Educational management system	51	7429	78	226	290	9	304

CWE-79 article, which can be further improved with the adoption of new CWE articles and their proposed mitigation techniques.

Table 2 shows that with the help of the proposed solution the accuracy of the code is 83% for the carskilla.com domain whereas syntax error produced after resolving issues is 6%. This concludes that by improving the proposed prototype in terms of the scanner and all the suggested mitigation techniques provided in CWE articles, the ratio of syntax error to resolving issues can be minimized.

6. **Conclusions.** The major contribution of this work resides in the development of a framework that can utilize mitigation techniques provided in CWE entries. The proposed framework consists of different modules integrated with each other to achieve the desired result. CWE entries provide a different suggestion of each category such as architecture, development, and implementation. We closely analyzed the entry structure, which helped to extract mitigation techniques for the potential mitigation section. Then, we identified the particular vulnerability and converted its textual information to rules. Based on these rules, we scanned the code for possible vulnerability with regular expression based pattern(s). The porotype was furtherly developed to Add, Remove/Comment or Replace vulnerable code found during scanning with the possible mitigation solution. With the combination of publically reported vulnerable repositories and code transformation, this will help mitigate vulnerable code automatically. The prototype implementation showed that the vulnerable code can be transformed into a secure code automatically without human intervention.

In the future work, we plan to develop more advanced techniques to mitigate code vulnerabilities as the buffer overflows. The proposed framework is still on early stage, but the rules defined in vulnerability repository can be extracted with advanced NLP based processing of repositories to produce sophisticated regular expressions, which can help to reduce code anomalies after modification. To automate the selection of CWE articles from the repository, a particular vulnerability can also be done with the help of a mapping algorithm, to get the most relevant mitigation scheme for vulnerability.

Acknowledgment. This work is fully supported by Prince Sultan University under the Project ID SSP-18-5-08. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] A. S. Khan, Y. Javed, J. Abdullah, J. M. Nazim and N. Khan, Security issues in 5G device to device communication, *IJCSNS*, vol.17, no.5, pp.366-375, 2017.

[2] R. H. Veni, A. H. Reddy and C. Kesavulu, Identifying malicious Web links and their attack types in social networks, *IJSRCSEIT*, vol.3, no.4, pp.1060-1066, 2018.

[3] M. Babiker, E. Karaarslan and Y. Hoscan, Web application attack detection and forensics: A survey, *Proc. of the 6th Int. Symp. Digit. Forensic Secur. (ISDFS 2018)*, pp.1-6, 2018.

[4] M. Uma and G. Padmavathi, A survey on various cyber attacks and their classification, *Int. J. Netw. Secur.*, vol.15, no.5, pp.390-396, 2013.

[5] Y. Javed and M. Alenezi, Defectiveness evolution in open source software systems, *Procedia Comput. Sci.*, vol.82, pp.107-114, 2016.

[6] M. Alenezi and Y. Javed, Open source Web application security: A static analysis approach, *2016 Int. Conf. Eng. MIS*, pp.1-5, 2016.
[7] M. Alenezi and Y. Javed, Developer companion: A framework to produce secure Web applications, *Int. J. Comput. Sci. Inf. Secur.*, vol.14, no.7, p.5, 2016.
[8] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee and S.-Y. Kuo, Securing Web application code by static analysis and runtime protection, *Proc. of the 13th Conf. World Wide Web – WWW'04*, p.40, 2004.
[9] X. Li and Y. Xue, *A Survey on Web Application Security*, Isis.Vanderbilt.Edu, Nashville, 2011.
[10] B. Arkin, S. Stender and G. McGraw, Software penetration testing, *IEEE Secur. Priv.*, vol.3, no.1, pp.84-87, 2005.
[11] M. Khanna, N. Chauhan, D. Sharma and A. Toofani, A novel approach for regression testing of Web applications, *Int. J. Intell. Syst. Appl.*, vol.10, no.2, pp.55-71, 2018.
[12] A. Rashid et al., Scoping the cyber security body of knowledge, *IEEE Secur. Priv.*, vol.16, no.3, pp.96-102, 2018.
[13] S. Gupta and B. B. Gupta, Cross-Site Scripting (XSS) attacks and defense mechanisms: Classification and state-of-the-art, *Int. J. Syst. Assur. Eng. Manag.*, vol.8, pp.512-530, 2017.
[14] Z. S. Alwan and M. F. Younis, Detection and prevention of SQL injection attack: A survey, *Int. J. Comput. Sci. Inf. Technol.*, vol.6, no.8, pp.5-17, 2017.
[15] G. R. K. Rao, K. V. J. S. S. Ram, M. A. Kumar, R. Supritha, S. A. Reza and B. Tech, Cross site scripting attacks and preventive measures, *Int. Res. J. Eng. Technol.*, vol.4, no.3, pp.2016-2019, 2017.
[16] D. A. Kindy and A. S. K. Pathan, A detailed survey on various aspects of SQL injection in Web applications: Vulnerabilities, innovative attacks and remedies, *Int. J. Commun. Networks Inf. Secur.*, vol.5, no.2, pp.80-92, 2013.
[17] F. Holik and S. Neradova, Vulnerabilities of modern Web applications, *The 40th Int. Conv. Inf. Commun. Technol. Electron. Microelectron.*, pp.1256-1261, 2017.
[18] M. S. Mahindrakar, Prevention to cross-site scripting attacks: A survey, *International Journal of Science and Research*, vol.3, no.7, pp.414-418, 2014.
[19] M. Salehie and L. Tahvildari, Self-adaptive software: Landscape and research challenges, *Trans. Auton. Adapt. Syst.*, vol.4, no.2, 2009.
[20] S. Christey and R. A. Martin, *Vulnerability Type Distributions in CVE*, MITRE, Common Weakness Enumer., pp.1-38, 2007.
[21] S. Zhang, D. Caragea and X. Ou, An empirical study on using the national vulnerability database to predict software vulnerabilities, *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol.6860, no.1, pp.217-231, 2011.
[22] M. Howard, Improving software security by eliminating the CWE top 25 vulnerabilities, *IEEE Secur. Priv.*, vol.7, no.3, pp.68-71, 2009.
[23] R. A. M. Coley, S. Christey, J. E. Kenderdine and L. Piper, *CWE Version 2.9*, 2015.
[24] A. K. Kyaw, F. Sioquim and J. Joseph, Dictionary attack on wordpress: Security and forensic analysis, *The 2nd Int. Conf. Inf. Secur. Cyber Forensics (InfoSec 2015)*, pp.158-164, 2016.
[25] Z. P. Reynolds, A. B. Jayanth, U. Koc, A. A. Porter, R. R. Raje and J. H. Hill, Identifying and documenting false positive patterns generated by static code analysis tools, *Proc. of IEEE/ACM the 4th Int. Work. Softw. Eng. Res. Ind. Pract. SER IP*, pp.55-61, 2017.
[26] A. K. Talukder et al., Security-aware Software Development Life Cycle (SaSDLC): Processes and tools, *IFIP Int. Conf. Wirel. Opt. Commun. Networks (WOCN'09)*, pp.1-5, 2009.
[27] M. Nadeem, E. B. Allen and B. J. Williams, A method for recommending computer-security training for software developers: Leveraging the power of static analysis techniques and vulnerability repositories, *Proc. of the 12th Int. Conf. Inf. Technol. New Gener. (ITNG)*, pp.534-539, 2015.