Pertanika J. Sci. & Technol. 31 (3): 1245 - 1265 (2023)

PERTANIKA JOURNALS

SCIENCE & TECHNOLOGY

Journal homepage: http://www.pertanika.upm.edu.my/

Automation of Quantifying Security Risk Level on Injection Attacks Based on Common Vulnerability Scoring System Metric

Aditya Kurniawan[1]*, Mohamad Yusof Darus[2], Muhammad Azizi Mohd Ariffin[2], Yohan Muliono[1] and Chrisando Ryan Pardomuan[1]

[1]*School of Computer Science, Bina Nusantara University, Kota Jakarta Barat, Daerah Khusus Ibukota 11530, Jakarta, Indonesia*
[2]*Faculty of Computer and Mathematical Sciences, UiTM, Shah Alam, 40450, Malaysia*

ABSTRACT

An injection attack is a cyber-attack that is one of The Open Web Application Security Project Top 10 Vulnerabilities. These attacks take advantage of insufficient user input validation into the system through the input surface of a Web application as that user in the browser. The company's cyber security team must filter thousands of attacks to prioritize which attacks are considered the most dangerous to be mitigated first. This activity of filtering thousands of attacks takes much time because you have to check these attacks one by one. Therefore, a method is needed to assess how dangerous a cyber-attack is that enters an organization's or company's server. Injection attack detection can be done by analyzing the request data in the web server log. Our research attempts to perform quantification modeling of the variations of two types of injection attacks, SQL Injection (SQLi) and Cross-Site Scripting (XSS), using Common Vulnerability Scoring System Metrics (CVSS). CVSS metrics are generally used to calculate the level of dangerous weakness in the system. This metric is never used to calculate the level of how dangerous an attack is. The modeling that we have made shows that SQLi and XSS attacks have many variations in levels ranging from low to high levels. We discovered that when classified with Common Weakness Enumeration Database, SQLi and XSS attacks CVE values would have high-level congruence with almost 94% value between one another vector on CVSS.

Keywords: Common vulnerability scoring system, injection attack, metrics security risk level

ARTICLE INFO

Article history:
Received: 07 February 2022
Accepted: 24 May 2022
Published: 31 March 2023

DOI: https://doi.org/10.47836/pjst.31.3.07

E-mail addresses:
adkurniawan@binus.edu (Aditya Kurniawan)
yusof@tmsk.uitm.edu.my (Mohamad Yusof Darus)
mazizi@fskm.uitm.edu.my (Muhammad Azizi Mohd Ariffin)
ymuliono@binus.edu (Yohan Muliono)
chrisando.pardomuan@binus.edu (Chrisando Ryan Pardomuan)
* Corresponding author

ISSN: 0128-7680
e-ISSN: 2231-8526

INTRODUCTION

An injection attack on a website is one of the malicious hackers' most executed attack vectors. When an injection attack is executed, the hacker will enter malicious input into an application or program. The interpreter or compiler will process this input to get the expected output from the injection attack, such as data theft, denial of service, loss of data integrity, or bypassing system authentication. Malicious hackers exploit flaws and errors in validating user input on input surfaces such as forms or URLs (Kindy & Pathan, 2011).

Retrieving hidden data is a type SQLi that can modify SQL Query to return expected additional data results. Subverting application logic is a type SQLi that changes the query to interfere with the application's algorithm (Pramod et al., 2015). Aliero & Qureshi et al. (2020) conducted literature review research on SQL injection attacks published between 2006 and 2019. Their research found that of 82 papers filtered from 1261 papers, 85.4% or 70 papers proposed SQLIA mitigation & prevention methods and tools, 8.5% or seven papers proposed experimental evaluation methods, and 6.1% or five papers proposed analytical methods. The results of the study literature review by them have not found any research on assessing the dangerous level of an SQL injection attack. UNION attacks are a type SQLi that can retrieve data from different database tables. Examining the database is a type SQLi that can extract the structure of database information. Blind SQL injection is a type SQLi that use to test whether a website is SQLi vulnerable or not through query response that shows in the application's response (Aliero & Ghani, et al., 2020; Sadeghian et al., 2013). The malicious hacker usually executed SQLi attacks via automation tools such as NMAP (Alazmi & de Leon, 2022; Kieyzun et al., 2009).

Javascript code injection vulnerabilities have three variations: Reflected, Store, and Dom-based (Fogie et al., 2007). The Reflected type is an XSS vulnerability that originates from an HTTP request as a URL (Sarmah et al., 2018). Type Store is the type of XSS vulnerability coming from the JavaScript code that is deliberately inputted into the database by a malicious hacker with the purpose when the JavaScript code is read by the process of querying the database for display in the user interface web, executed during the process of web page loading and processing processes that malicious. The DOM-based type is an XSS attack that exploits JavaScript code that processes data from untrusted sources with insecure processing methods (Bisht & Venkatakrishnan, 2008). Several studies on detecting cross-site scripting vulnerabilities have been carried out on several types of cross-site scripting vulnerabilities (Sarmah et al., 2018). According to Sarmah et al., research on cross-site scripting is divided into two approaches, namely client-side and server-side approaches. Several studies using a client-side approach have been carried out using the static analysis method, as many as 10 studies, and the hybrid analysis method, as many as four studies. While the number of studies using the server-side approach with the static analysis method is as many as 12 studies, the dynamic analysis method has as many

as nine studies, and the hybrid analysis method has as many as two studies. This study shows that research in the server-side approach area with the static analysis method is the most studied. Reflected XSS detection has been widely studied using regular expression and string-matching methods (Bates et al., 2010; Gupta & Gupta, 2016; Pelizzi & Sekar, 2012; Rao et al., 2016; Wang & Zhou, 2016) (cite xx auditor, xss filt, no script, ie 2008, xss immune, xbuster, rule based). Meanwhile, for the detection of weaknesses in cross-site scripting types of DOM and Stored XSS, little research is done with string matching and string comparison.

Most injection attacks research only focuses on how to detect these injection attacks, filtering input as a defense mechanism, or also preventing the entry of injection attacks code by utilizing third-party software such as intrusion detection system (Bisht & Venkatakrishnan, 2008; Bozic & Wotawa, 2013; Gupta & Gupta, 2017; Kumar Singh & Roy, 2012). However, no research discusses how to assess the ability of malicious hackers to exploit injection vulnerabilities of the complexity of the injection attacks. Knowing the complexity of the attacks launched by these malicious hackers, the incident response team's resources can be focused on attacks that can be considered dangerous to the system (Athanasopoulos et al., 2010).

Measuring cyber security in a company's digital assets can help them define an organization's security posture. Appropriate cyber security measures can help companies (1) verify whether their control security has complied with a policy, process, or procedure, (2) help identify strengths and weaknesses and findings of exploitation of the company's digital assets, and (3) help identify trends in security, both in the internal and external environment of the organization (Voeller, 2008). Companies can monitor their cyber security and defense performance by keeping abreast of the latest cyber security trends. CVSS metrics help companies measure how big a weakness is in the system. In this study, we use CVSS metrics to measure how dangerous a cyber-attack is combined with NVD data.

The accuracy of the CVSS calculation score is influenced by the knowledge of the security researcher and its analysis. It contains an element of their subjectivity in determining each score of the CVSS variables. Security researchers worldwide will upload their calculations for every type of vulnerability they find in technology or system to organizations such as NVDs. NVD organizations use the Common Vulnerabilities and Exposures (CVE) standard for recording the uploaded weakness catalog (Aksu et al., 2018). NVD organizations modify CVE by providing a classification for each weakness known as Common Weakness Enumeration (CWE). This study uses CWE to classify weaknesses and insert some CWE values into the CVSS variables.

Several studies propose several uses of CVSS in certain cybersecurity fields. CVSS is used to analyze the risk of vulnerabilities in system assets as measured by the frequency and process of risk management and cybersecurity strategies in corporate and industrial

environments (Figueroa-Lorenzo et al., 2021; Houmb et al., 2010a). CVSS is also trying to be implemented to illustrate the graph of cyber-attacks (Gallon & Bascou, 2011a). The application of CVSS has also tried to be automated and use machine learning from the findings of weaknesses in the system (Beck & Rass, 2016; Elbaz et al., 2020; Minh Le et al., 2021; Radack & Kuhn, 2011). From a literature review study conducted regarding the application of CVSS, it was found that there was no CVSS automation application to measure the dangerous level of a cyber-attack, especially in SQL injection attacks and cross-site scripting attacks.

The paper is organized into the following section. Section 2 discussed the material and methodology of the CVSS method and its implementation on level attack scoring. Section 3 discussed the result and evaluation of level attack scoring CVSS on SQL injection and cross-site scripting attacks. Finally, we wind up in Section 4 with the conclusion of our research.

MATERIAL

Related Works

Common Vulnerability Scoring System. Common Vulnerability Scoring System is a system for conducting a quantitative assessment of a vulnerability by Common Weakness Enumeration, giving metrics to the properties of a vulnerability built and maintained by the Forum of Incident Response (Scarfone & Mell, 2009). Quantitative assessments are calculated based on formulas that depend on several metrics representing the ease of exploitation, the impact of mitigation, and the way the vulnerability is spread (Houmb et al., 2010b, 2010a). Quantitative assessment can be represented qualitatively (Low, Medium, High, and Critical) to help the organization assess and prioritize changes to their system.

CVSS has also become a standard system used by many agencies and companies. This research uses CVSS 3.0 version (Gallon & Bascou, 2011b; Houmb & Franqueira, 2009). CVSSv3 has three main metrics as follows:

(1) Base Metrics

This metric describes the vulnerability's characteristics and traits. This metric consists of the following:

(a) Exploitability Metrics

Metrics that represent the characteristics of a security hole, where vulnerable components will be given a score based on indications of a security hole that leads to the success of an attack. Exploitability metrics consist of Attack Vector (AV), Attack Complexity (AC), Privilege Required (PR), and User Interaction (UI).

(b) Scope

Scope metric (S) represents the scope of the impact of the security gap. Does

the security gap provide an opportunity for attackers to get coverage of access rights regulated by other access rights coverage. For example, the attacker can access and perform activities on the host OS by attacking VMware.

(c) Impact Metrics
Metrics that represent the impact of Vulnerability on the CIA triad (Confidentiality, Integrity, and Availability) of an infrastructure.

(2) Temporal Metrics
This metric reviews the current conditions based on exploitation techniques, the certainty of information about exploitation codes, or whether a solution is offered to correct the security hole itself.

(3) Environmental Metrics
This metric allows users or analysts to adjust Base Metrics to the organizational needs of the user itself, which will later be applied to the organization's infrastructure in terms of place security controls or CIA triads (Confidentiality, Integrity, and Availability).

This research uses Exploitability metrics, Scope metrics, and Impact metrics.

METHODOLOGY

The steps of this research methodology have been prepared and executed by processing the data in the National Vulnerability Database (NVD) and SQL Injection attack data and cross-site scripting. The following are the detailed steps of this research methodology:

(1) Data Pre-processing
  (a) Standard Deviation of CVSS for Constant Variable
  (b) CVSS Calculation Simulation on Injection Attacks
(2) Analysis & Design CVSS Formulation
  (a) Dynamic CVSS Vector Calculation
  (b) CVSS Base Score Calculation
(3) Algorithm Implementation
(4) Evaluation

Data Pre-Processing

Standard Deviation of CVSS for Constant Variable. NVD data is downloaded and updated periodically to perform a static calculation of scores of multiple CVSS vectors. The data is then used to determine the majority score of each vector generally assigned to a type of attack. Based on the calculation of standard error and standard deviation of the two types of attacks on NVD data, the following results were obtained in Tables 1 and 2. SQL Injection and Cross-site Scripting attacks have very low standard deviation and standard errors; These observations indicate that NVD data has a low variation. Therefore, by using

the majority value of each vector in the NVD data (the score of each vector is determined based on the highest score on the NVD data for each vector), the score for each vector in the newly detected attack can be determined only based on the type of attack detected.

Table 1
Results of SQL injection statistical calculations on NVD dataset (as of 23 March 2020)

Vector	stderr	mean	std
AV	0.000795	0.847520	0.029299
AC	0.001051	0.765386	0.038760
PR	0.004775	0.748227	0.176041
UI	0.000608	0.847800	0.022396
S	0.001040	1.001472	0.038348
C	0.555629	0.555629	0.067088
I	0.003348	0.587947	0.123428
A	0.003630	0.598764	0.133822

Table 2
Results of XSS injection statistical calculations on NVD dataset (as of 23 March 2020)

Vector	Stderr	Mean	std
AV	0.000145	0.849619	0.010214
AC	0.000281	0.768804	0.019834
PR	0.001666	0.764082	0.117414
UI	0.000273	0.621621	0.019243
S	0.001485	1.988925	0.104665
C	0.000754	0.225554	0.053156
I	0.000638	0.224591	0.044990
A	0.000668	0.995276	0.047085

CVSS Calculation Simulation on Injection Attacks. This research uses eight field parameter that showed in Table 3 as follow:

- URL: URL string read from the Apache engine.
- body: Name and parameter value that will be checked whether there is a string injection pattern or not
- source_address: The attacker's IP address
- has_sql: Did the request trigger a connection to the database
- source_port: The attacker's (inbound) port number
- arrived_at: Time the request was received
- audit_xss: XSS detection result data by SIEM
- inspect_sqli: Data from SIEM's SQL Injection detection results

Table 3
Data logs generated for SQL injection attacks

Attribute	Value
url	http://server.com/products.php?id=1' OR 1=1-- -
body	id=1' OR 1=1-- -
source_address	115.166.114.89
source_port	33527
has_sql	true
arrived_at	2021-04-21T13:24:22+00:00
audit_xss	Not Detected
inspect_sqli	Detected

Table 4
SQL injection NVD scores

SQL								
Score	**AV**	**AC**	**PR**	**UI**	**S**	**C**	**I**	**A**
0						12	101	129
00.02	1							
00.22						33	19	12
00.27			120					
00.44		19						
00.05			0					
00.55	6							
00.56						1314	1239	1218
0,043	4		298	13				
0,047			1					
0,053		1340						
0,059	1348		940	1346				
C					2			
U					1357			

The measurement variables contained in CVSS will be calculated statically and dynamically. Static calculations will be carried out to obtain PR (Privileges Required) vectors, AC (Attack Complexity), UI (User Interaction), S (Scope), C (Confidentiality), I (Integrity), and A (Availability) based on CVSSv3 data NVD published regularly. Meanwhile, a dynamic calculation will be carried out to get a vector score of AV (Attack Vector).

NVD data will be retrieved periodically to ensure the data used in the static calculation is the most up to date. The next step is that the system will filter the data for attacks that are out of context, leaving only NVD data for SQL Injection and Cross-site Scripting. Furthermore, from the log data provided by SIEM, it is known that the attack that occurred on the HTTP Request was a SQL Injection attack or Cross-site Scripting. Therefore, the NVD data used is SQL Injection attack data, as shown in Table 4. The NVD data used in this simulation is from 2017 to 2019, taken on 23 March 2020.

The value of each metric is taken based on the majority score for the attacks identified by SIEM. Based on the table, for example, it can be determined that the value for the AC metric in this HTTP Request is 0.77 because it is the majority value (1340 data on NVD shows SQL Injection attacks have an AC score of 0.77). The calculation results show the scores assigned to each CVSS vector on this HTTP as follows:

- AC: 0.77 (Low)
- PR: 0.85 (None)
- UI: 085 (None)

- S: Unchanged (U)
- C: 0.56 (High)
- I: 0.56 (High)
- A: 0.56 (High)

Analysis and Design CVSS Applied Formulation

Dynamic CVSS Vector Calculation. From the log data provided by SIEM, the attacker's IP is 115.166.114.89. Then, to perform AV calculations, the system needs to get the IP of the server it wants to protect (Host IP). Host IP can be retrieved automatically via the ipconfig command, which is run automatically or set manually by the administrator. As a simulation, it is assumed that the Host IP is 10.20.20.122 with a subnet of 255.255.0.0. Next, the system performs a subnet calculation against the Host IP to determine whether the attacker's IP is on the same subnet as the Host, with the following calculation in Table 5. From the results of these calculations, the attacker's IP (115.166.114.89) is on a different subnet from the Host's IP (10.20.20.122), where only IPs are in the range 10.20.0.1 to 10.20.255.254 are on that subnet. Because the two IPs are on different subnets, it can be concluded that the AV from the attack was Network with a score of 0.85.

Table 5
Subnet calculation results based on host IP

IP Address:	**10.20.20.122**
Network Address:	10.20.0.0
Usable Host IP Range:	10.20.0.1 - 10.20.255.254
Broadcast Address:	10.20.255.255
Total Number of Hosts:	65,536
Number of Usable Hosts	65,534
Subnet Mask:	255.255.0.0
Wildcard Mask:	0.0.255.255

Base Score CVSS Calculation. From the results of these calculations, the CVSS Vector String is obtained as follows:

CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

To determine the CVSS Base Score from this HTTP Request, the Exploitability Score and Impact Score were calculated; because the scope of this attack is Unchanged, the CVSS calculation formula used is Equation 1:

$$Impact = 6.42 \times ISS$$

$$ISS = 1 - ((1 - C) \times (1 - I) \times (1 - A))$$

$$Exploitability = 8.22 \times AV \times AC \times PR \times UI$$

$$BaseScore = Roundup(Min((Impact + Exploitability), 10)) \quad (1)$$

Based on this formula, the Exploitability Score is calculated as Equation 2:

$$Exploitability = 8.22 \times 0.85 \times 0.77 \times 0.85 \times 0.85 = 3.887 \quad (2)$$

Next, the Impact Score is calculated by calculating the Impact Sub Score (ISS) first, as Equation 3:

$$ISS = 1 - \big((1 - 0.56) \times (1 - 0.56) \times (1 - 0.56)\big) = 0.914 \quad (3)$$

Because the scope of this attack is Unchanged, the Impact Score (Equation 4):

$$Impact = 6.45 \times 0.914 = 5.895 \quad (4)$$

From these calculations, because the scope of this attack is Unchanged, the Base Score is obtained (Equation 5):

$$BaseScore = Roundup\Big(Min\big((3.887 + 5.895), 10\big)\Big) = 9.8 \quad (5)$$

From the calculation of the score is 9.8, it can be concluded that the severity level of the attack is Critical.

Algorithm Implementation

The implementation of this algorithm uses several server technologies, including Apache Server, Nginx, Redis, Elastic Kibana, and Logstash. The system diagram in Figure 1 illustrates the workflow of the system built in this study. Overall, there are five types of flow or data flow between modules, namely: initial flow, return flow, process flow, periodic flow, and concurrent flow, that described in Table 6.

The HTTP Sniffer module catches every HTTP packet that the system receives from the client, i.e., an HTTP request packet, when it is forwarded to the web server service (e.g., Apache, Nginx, among others). The HTTP Sniffer module normalizes the captured packets and sends them to the SQLi Feature Selector module for further processing. In addition, normalized HTTP packets are also sent to be stored on the Redis server, which is an in-memory data storage available in the system. This storage is done so that HTTP packets are still available and can be used in subsequent processes and so that they can be retrieved quickly. In the SQLi Feature Selector module, the input data received from

the user, whether it contains SQL syntax or not, is generalized into a token. The goal is to reduce the data's very high dimensionality or diversity to a simpler form. The results of the processing are then sent to the Audit Control module. From there, all received data will be used to detect SQLI and XSS attacks.

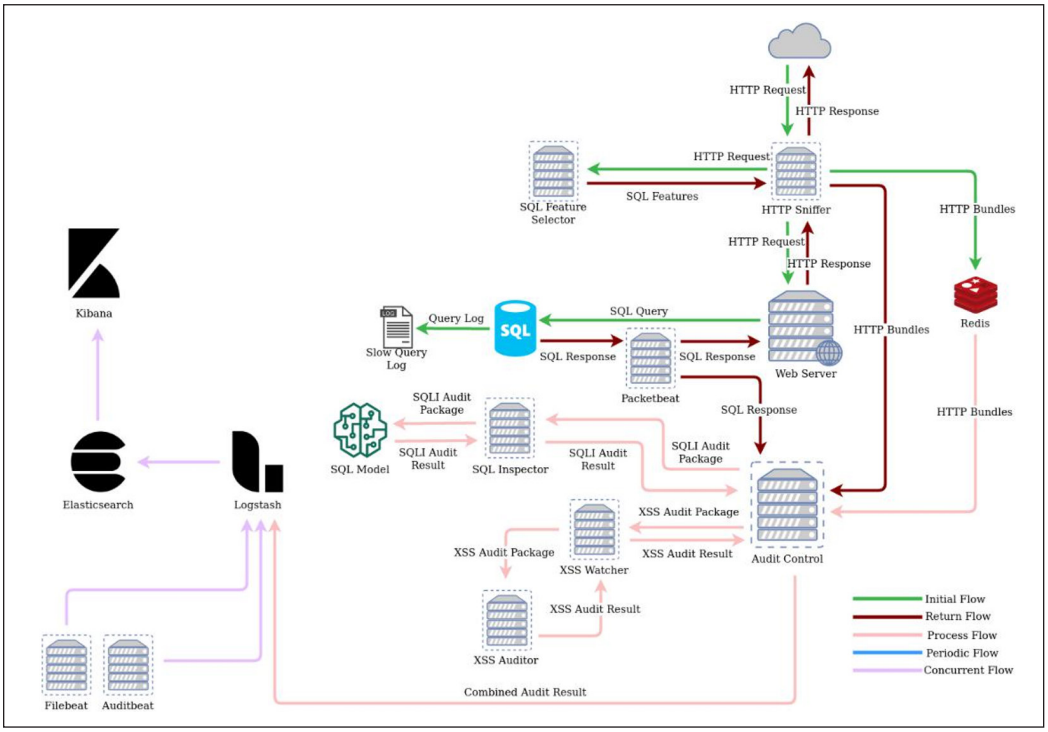

Figure 1. Inflow and outflow data traffic on system

Table 6
Data flow type description

Flow Type	Description	Example
Initial Flow	Occurs when a request comes from a client until before the response is returned	HTTP Request Packet Data
Return Flow	Occurs when a response has been received from client, processed, and returned to the client	HTTP Response Packet Data
Process Flow	Executed specifically by a module when all information needs have been met and are ready to be processed. It is on the fly and not tied to the initial flow or return flow	Data from the Redis server to the Audit Control module
Periodic Flow	Data flow is carried out periodically within a certain time	Data from Elastic Search to Kibana module
Concurrent Flow	The data flow always runs at any time, does not depend on a particular condition, and is not bound to other flows	Data from Filebeat and Auditbeat to Logstash

After the request from the client is processed by the web server and the response is sent back to the client in the form of an HTTP response packet, the packet is again captured by the HTTP Sniffer module and then forwarded to the Audit Control module along with the query result data from MySQL for XSS Auditing. The process results are returned to Audit Control and then sent to the Logstash server, which is also available in the system. Logstash servers receive, parse, and process information from various sources and then send it to the Elasticsearch servers as long-term storage.

Evaluation

Evaluation of the algorithm implementation uses a series of test procedures conducted against a server that runs a vulnerable web application.

RESULT

Result of Experiment

This research uses data feeds from the National Vulnerability Database (NVD) in 2017, 2018, and 2019. The table structure in the NVD will take the following data:

(1) CVE: CVE data has a \textbf{problemtype} column that has a CWE classification value
(2) Configuration: This section has the effect of weakness on the system
(3) Impact: This section has valuable information from CVSS
(4) Published Date: Publish date for a CVE record
(5) Last Modified: The last modified date for a CVE record

CWEs and CVEs data are important in assessing a software's vulnerability, and it provides an easy reference to understand existing vulnerabilities, which can ease software quality testing. Therefore, this research uses CWE as part of the classification of injection attacks. The CWE identifier is CWE-89 for SQL injection attacks and CWE-1033 for cross-site scripting injection attacks.

JSON data feeds from NVD and CVE have inconsistent scores, which can be seen in Figures 2, 3, and 4. Therefore, this research tries to find the congruence of each CVSS by using Equation 6:

$$c = \sum_{n=1}^{\infty} \frac{mode(t)}{y} \tag{6}$$

Where:
t = Each Vector's Possible values (Such as N, L, A, P for Attack Vector)
y = The total data gathered for such a vector.
c = Congruence

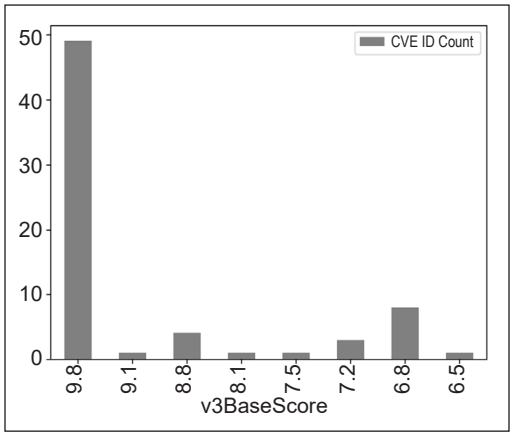

Figure 2. SQL injection CVSS3 scores in 2019

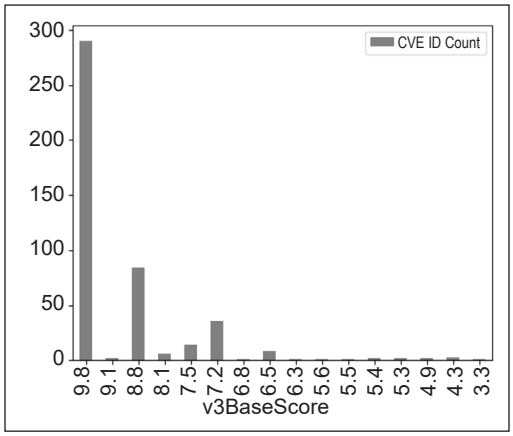

Figure 3. SQL injection CVSS3 scores in 2018

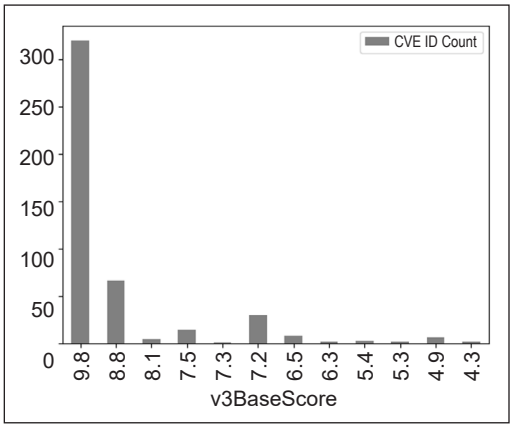

Figure 4. SQL injection CVSS3 scores in 2017

Table 7
Congruence values of CVE

Parameter	SQL	XSS
AV	99.3\%(N)	99.8\%(N)
AC	98.7\%(L)	99.6\%(L)
PR	73.0\%(N)	60.56\%(N)
UI	98.8\%(N)	99.2\%(R)
S	100\%(U)	98.8\%(C)
C	97.8\%(H)	98.7\%(L)
I	92.6\%(H)	98.8\%(L)
A	91.9\%(H)	98.9\%(N)
Total	985 CVEs	4681 CVEs

The calculation results of the above formula can be seen in Table 7. This research does not use the Attack Vector (AV) parameter in the calculation results because this parameter becomes invalid. After all, the attack surface can be accessed by hackers in a network interface protocol. Therefore, to check an attacker's IP address, whether it is still in one subnet or not, with the following algorithm:

Algorithm 1: AV String Vector Generation Algorithm

```
Result: Attack Vector (Network / Adjacent)
let compare = getInputIP() ;
let localip.prefix = getNetworkInterfaceInfo();
let subnets = GenerateSubnet(localip.prefix);
procedure findMatchingSubnet(ip)
for (subnet in subnets) do
```

```
    if IP is in the subnet, then
        return subnet
    else
        return -1
    end
end
if findMatchingSubnet(localip) is equal to findMatchingSubnet(compare) then
    return "A";
else
    return "N";
end
```

The Privileged Require (PR) vector is the lowest vector of all existing CVSS vectors. This vector can be assessed manually using the Principle of Least Privilege method, which uses restrictions on access to user roles or privileges.

Detection of user privileges, in general, can use the Cookies variable. This method is effective when the attacker is using a browser. Cookies have an interesting section containing a URL Path that can indicate the user's access rights. The Path variable contains information about where Cookie headers can be sent for authentication from the server.

This research uses the Path URL Cookies Header to identify whether an attacker has fully gained access to a specific Path. Since the Cookies header was sent to a specified path, we can conclude that if the attacker managed to access the specified path without generating an HTTP error, the attacker has essentially gained the privilege to access that Path and retrieved the server's corresponding resources. It can be done without forcefully checking the Cookie header from the incoming Request.

The privileges declared can vary; however, the Super User is one absolute privilege. Like the Linux counterpart, a superuser is allowed to execute any command in the server, giving them the highest level of privilege. If that is the case, the PR would be scored as "H" or High since the attacker has gained the highest privilege within the web server. Other privileges specified within the URL are relative to the use.

DISCUSSION

In this study, we used many log entries considered malicious and detected by the IDS or SIEM system. This study develops a log parser tool to convert logs from the detection system into a file with a .csv extension. The output taken from the log is IP Address; URL; Score; Vector String.

We would also use VSS, a Python library, to calculate the CVSS3.0 score. The script that implements the library takes an input of a CVSS:3.0 vector string and prints out the generated CVSS3.0 score.

Table 7 shows the results of the testing procedure. It all comprised SQL Injection attempts toward a victim web server located at 172.17.0.2 with a prefix of 16.

We can see from the table below that the score changes corresponding to the *URL* and *IP* parameter given.

There are a few things to take note of:

(a) We omit the queries of the URL string to shorten the sentence length in this paper.
(b) We omit the request method.

The following are the explanations of each row of the result table.

(1) In this case, the SIEM System recorded an attack in the first row with the following data.
    (a) IP: 171.25.193.77
    (b) URL: /
    (c) Timestamp: 1564284545
    (d) Is a threat: yes
    (e) SQLi type: tautology

Since the attacker is located outside the system's subnet, it is classified as a remote attack. The URL appears to be a non-privileged URL based on the configuration.

So, from there, the dynamically designed vector, which consists of *AV* and *PR*, would be scored *N* and *N*, respectively.

Since the vulnerability is classified as SQL Injection, the predetermined values would be **C:H/I:H/A:H/UI:N/S:U/AC:L**. Thus, the final CVSS vector string would be **C:H/I:H/A:H/AV:N/UI:N/S:U/PR:N/AC:L**, which is scored as *9.8* according to CVSS:3.0.

(2) In the second row, the SIEM System recorded an attack with the following data in this case.
    (a) IP: 171.25.193.235
    (b) URL: /index
    (c) Timestamp: 1564284545
    (d) Is a threat: yes
    (e) SQLi type: tautology

This row is very similar to the first row. The attacker is outside of the subnet the machine is in and has a URL with no privilege.

(3) In the third row, the SIEM System recorded an attack with the following data in this case.
    (a) IP: 172.17.0.9
    (b) URL: /2006
    (c) Timestamp: 1564284548
    (d) Is a threat: yes
    (e) SQLi type: tautology

This row features an adjacent attacker with an IP of *172.17.0.9*, which would modify the Attack Vector into *A* or Adjacent. Other than that, it is still the same as the previous row with a vector string of **C:H/I:H/A:H/ AV:A/UI:N/S:U/PR:N/AC:L** and a score of **8.8**.

(4) In this case, the SIEM System recorded an attack in the fourth row with the following data.
   (a) IP: 172.25.193.20
   (b) URL: /crack
   (c) Timestamp: 1564284548
   (d) Is a threat: yes
   (e) SQLi type: tautology

This row features an attack with a URL specified as having high privilege, which would modify the Privilege Required vector into *H* or Adjacent. Other than that, it is still the same as the previous row with a vector string of **C:H/I:H/A:H/ AV:A/UI:N/S:U/PR:H/ AC:L** and a score of **7.2**.

Table 8
Automatic summarizing (Victim IP is 172.17.0.2)

IP **URL** **timestamp** **category** **class**	**Vector**	**Score**
171.25.193.77 **/** **1564284545** **threat** **tautology**	C:H/I:H/A:H/AV:N/UI:N/S:U/PR:N/AC:L/	9.8
171.25.193.235 **/index** **1564284545** **threat** **tautology**	C:H/I:H/A:H/AV:N/UI:N/S:U/PR:N/AC:L/	9.8
172.17.0.9 **/2006** **1564284548** **threat** **tautology**	C:H/I:H/A:H/AV:A/UI:N/S:U/PR:N/AC:L/	8.8
171.25.193.20 **/crack** **1564284549** **threat** **tautology**	C:H/I:H/A:H/AV:N/UI:N/S:U/PR:H/AC:L/	7.2

EVALUATION

A series of test procedures were conducted against a server that runs a vulnerable web application to evaluate the overall performance of the methods. The server was assigned 10.20.153.52/20 as the IP address. Note that the /20 CIDR notation indicates that the IP is a Class C. The first test procedure involves a potential Cross-site scripting attack that has been detected, and the detection data is fed to the system. The data is shown in Figure 6.

After the summarizing process, data that indicates the severity measurement result (Figure 6, is emitted by the system. From the data, it can be concluded that the attack has a severity score of 6.1, denoted **medium** severity, with a likelihood percentage of 94%. The data also shows the vector string of the attack, which is very helpful for a vulnerability assessment and incident report.

The attack is analyzed based on the CWE tag. The IP and server prefix (CIDR) are obtained, then the system detects that the attack can be categorized as a Network Attack. P_ERRORS attribute in the output data indicates that the endpoint URL does not exist in the database. Hence, the system directly inserts the URL into the database so that future detections involving the same attack and endpoint will use the measured severity as precedence.

Same for the SQL Injection attack testing, the detection data (Figure 7, is fed to the system to be analyzed.

After the summarizing process, data in Figure 8, is emitted. From the data, it can be concluded that the attack also has a severity score of 6.1, denoted **medium** severity, with a likelihood percentage of 94%.

The system measures an attack with SQL Injection payload (CWE-89 based on

```
 {
"_id": "124",
"timestamp":str(datetime.datetime.now()),
"vul":"XSS",
"ip":"10.20.30.40",
"url":"/xss/stored"
}
```

Figure 5. Data input for cross-site scripting testing


```
{
   "_index" : "nethive-cvss",
   "_type" : "_doc",
   "_id" : "YhVwOG8BWtOlYx1xVPQM",
   "_\textit{score}" : 1.0,
   "_source" : {
     "corr_id" : "124",
     "timestamp" : "2020-01-23T03:26:56.147001",
     "SUMMARIZE_RESULT" : {
       "vector" : "AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N/",
       "\textit{score}" : 6.1,
       "severity" : "MEDIUM",
       "errors" : {
         "PR_ERRORS" : "2020/01/23 03:26:58 sql: no rows in
                        result set\n"
       },
       "likelyhood" : 94
     }
   }
 }
```


Figure 6. Data output for cross-site scripting testing

tag classification). The IP source of the attack indicates that the attack was carried out from a machine under the same network. Hence, the attack is categorized as an Adjacent attack.

Overall, 1359 SQL Injection and 4966 Cross-site Scripting attack simulations are conducted against the system. The accuracy of the automatic measurement is calculated, with human-assessed severity as the ground truth. The result is shown in Table 9.

Apart from the accuracy measurement, the *common error cause* is also measured for every vulnerability metric on SQL Injection and Cross-site Scripting attack data. The result is shown in Table 10. The *common error cause* illustrates the amount of attack data for each metric that are incongruent with the rest of the data.

```
{
    "_id": "124",
    "timestamp":str(datetime.datetime.now()),
    "vul":"SQLi",
    "ip":"10.20.153.54",
    "url":"/vulnerabilities/sqli"
}
```

Figure 7. Data input for SQL injection testing

```
{
   "_index" : "nethive-cvss",
   "_type" : "_doc",
   "_id" : "YxV20G8BWtOlYx1x5PQh",
   "_\textit{score}" : 1.0,
   "_source" : {
     "corr_id" : "124",
     "timestamp" : "2020-01-23T03:34:07.456804",
     "SUMMARIZE_RESULT" : {
       "vector" : "AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N/",
       "\textit{score}" : 6.1,
       "severity" : "MEDIUM",
       "errors" : { },
       "likelyhood" : 94
     }
   }
 }
```

Figure 8. Data output for SQL injection testing

Table 9
Accuracy percentage of SQL injection and cross-site scripting

Attack Type	String Accuracy %	Score Accuracy %
SQL Injection	62,61957321559971	62,61957321559971
Cross-site Scripting	59,30326218284333	59,28312525171164

Table 10
Common error cause breakdown for SQL injection and cross-site scripting

Attack Type	Common Error Cause							
	AV	**AC**	**PR**	**UI**	**S**	**C**	**I**	**A**
SQL Injection	7	19	419	13	2	45	120	7
Cross-site Scripting	4	18	1964	35	55	63	58	4

CONCLUSION

In this paper, the author designed a novel system to automate the calculation of CVSSv3.0 scores on SQL Injection and XSS vulnerabilities based on data from NVD and network logs. Dynamic analysis can be performed on AV Vector (Attack Vector) metric, while the PR (Privilege Required) metric is shown to be able to be calculated pseudo-dynamically, with a note that the accuracy rate is 59.28% for Cross-site Scripting attacks, and 62.62% for SQL Injection attack. For other vectors, the data from NVD has a high enough congruence that can be used as an element of static calculations.

Although, the authors have to admit that the methods we suggested are hardly flawless since it relies on the congruence of currently existing vector strings in NVD. If, in any case, the NVD data feeds become highly incongruent, then this paper becomes deprecated. Another point we should mention is that perhaps further research is required in detecting privilege. Since what we propose perhaps is effective, it is slightly hard coded. Though now it is statistically proven that a vulnerability's base vectors, most of the time, do have similar values, disregarding the software it is exploited in. Finally, the author suggests further researchers examine data from other sources, such as the Open Source Vulnerability Database (OSVDB) (Kouns, 2008), and extensively test the theory used in this paper.

ACKNOWLEDGEMENT

This work was supported by the Collaboration International Grant 2021 between Bina Nusantara University, Indonesia and Universiti Teknologi MARA (UiTM) Shah Alam, Malaysia.

REFERENCES

Aksu, M. U., Bicakci, K., Dilek, M. H., Ozbayoglu, A. M., & Tatli, E. I. (2018). Automated generation of attack graphs using NVD. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy* (pp. 135-142). ACM Publishing. https://doi.org/10.1145/3176258.3176339

Alazmi, S., & de Leon, D. C. (2022). A systematic literature review on the characteristics and effectiveness of web application vulnerability scanners. *IEEE Access*, *10*, 33200-33219. https://doi.org/10.1109/ACCESS.2022.3161522

Aliero, M. S., Ghani, I., Qureshi, K. N., & Rohani, M. F. (2020). An algorithm for detecting SQL injection vulnerability using black-box testing. *Journal of Ambient Intelligence and Humanized Computing*, *11*, 249-266. https://doi.org/10.1007/s12652-019-01235-z

Aliero, M. S., Qureshi, K. N., Pasha, M. F., Ghani, I., & Yauri, R. A. (2020). Systematic review analysis on SQLIA detection and prevention approaches. *Wireless Personal Communications*, *112*, 2297-2333. https://doi.org/10.1007/s11277-020-07151-2

Athanasopoulos, E., Pappas, V., Krithinakis, A., Ligouras, S., Markatos, E. P., & Karagiannis, T. (2010, June 23-24). *xJS: Practical XSS prevention for web application development* [Paper presentation]. USENIX Conference on Web Application Development, Boston, MA, USA.

Bates, D., Barth, A., & Jackson, C. (2010). Regular expressions considered harmful in client-side XSS filters. In *Proceedings of the 19th International Conference on World Wide Web - WWW '10* (pp. 91-100). ACM Publishing. https://doi.org/10.1145/1772690.1772701

Beck, A., & Rass, S. (2016). Using neural networks to aid CVSS risk aggregation - An empirically validated approach. *Journal of Innovation in Digital Ecosystems*, *3*(2), 148-154. https://doi.org/10.1016/j.jides.2016.10.002

Bisht, P., & Venkatakrishnan, V. N. (2008). XSS-GUARD: Precise dynamic prevention of cross-site scripting attacks. In D. Zamboni (Ed.), *Detection of Intrusions and Malware, and Vulnerability Assessment* (Vol. 5137, 23-43). Springer. https://doi.org/10.1007/978-3-540-70542-0_2

Bozic, J., & Wotawa, F. (2013). XSS pattern for attack modeling in testing. In *2013 8th International Workshop on Automation of Software Test (AST)* (pp. 71-74). IEEE Publishing. https://doi.org/10.1109/IWAST.2013.6595794

Elbaz, C., Rilling, L., & Morin, C. (2020). Fighting N-day vulnerabilities with automated CVSS vector prediction at disclosure. In *Proceedings of the 15th International Conference on Availability, Reliability and Security* (pp. 1-10). ACM Publishing. https://doi.org/10.1145/3407023.3407038

Figueroa-Lorenzo, S., Añorga, J., & Arrizabalaga, S. (2021). A survey of IIoT protocols: A measure of vulnerability risk analysis based on CVSS. *ACM Computing Surveys*, *53*(2), 1-53. https://doi.org/10.1145/3381038

Fogie, S., Grossman, J., Hansen, R., & Petkov, P. D. (2007). *XSS Attacks: Cross Site Scripting Exploits and Defense* (1st ed.). Syngres Media.

Gallon, L., & Bascou, J. J. (2011a). Using CVSS in attack graphs. In *2011 Sixth International Conference on Availability, Reliability and Security* (pp. 59-66). IEEE Publishing. https://doi.org/10.1109/ARES.2011.18

Gallon, L., & Bascou, J. J. (2011b). Using CVSS in attack graphs. In *2011 Sixth International Conference on Availability, Reliability and Security* (pp. 59-66). IEEE Publishing. https://doi.org/10.1109/ARES.2011.18

Gupta, S., & Gupta, B. B. (2016). XSS-immune: A google chrome extension-based XSS defensive framework for contemporary platforms of web applications. *Security and Communication Networks*, *9*(17), 3966-3986. https://doi.org/10.1002/sec.1579

Gupta, S., & Gupta, B. B. (2017). Cross-site scripting (XSS) attacks and defense mechanisms: Classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, *8*(Suppl 1), 512-530. https://doi.org/10.1007/s13198-015-0376-0

Houmb, S. H., & Franqueira, V. N. L. (2009). Estimating ToE risk level using CVSS. In *2009 International Conference on Availability, Reliability and Security* (pp. 718-725). IEEE Publishing. https://doi.org/10.1109/ARES.2009.151

Houmb, S. H., Franqueira, V. N. L., & Engum, E. A. (2010a). Quantifying security risk level from CVSS estimates of frequency and impact. *Journal of Systems and Software*, *83*(9), 1622-1634. https://doi.org/10.1016/j.jss.2009.08.023

Houmb, S. H., Franqueira, V. N. L., & Engum, E. A. (2010b). Quantifying security risk level from CVSS estimates of frequency and impact. *Journal of Systems and Software*, *83*(9), 1622-1634. https://doi.org/10.1016/j.jss.2009.08.023

Kieyzun, A., Guo, P. J., Jayaraman, K., & Ernst, M. D. (2009). Automatic creation of SQL Injection and cross-site scripting attacks. In *2009 IEEE 31st International Conference on Software Engineering* (pp. 199-209). IEEE Publishing. https://doi.org/10.1109/ICSE.2009.5070521

Kindy, D. A., & Pathan, A.-S. K. (2011). A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques. In *2011 IEEE 15th International Symposium on Consumer Electronics (ISCE)* (pp. 468-471). IEEE Publishing. https://doi.org/10.1109/ISCE.2011.5973873

Kouns, J. (2008). *Open source vulnerability database project*. TIM Review. https://timreview.ca/article/155

Le, T. H. M., Hin, D., Croft, R., & Babar, M. A. (2021). DeepCVA: Automated commit-level vulnerability assessment with deep multi-task learning. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 717-729). IEEE Publishing. https://doi.org/10.1109/ASE51524.2021.9678622

Pelizzi, R., & Sekar, R. (2012). Protection, usability and improvements in reflected XSS filters. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security - ASIACCS '12* (pp. 1-5). ACM Publishing. https://doi.org/10.1145/2414456.2414458

Pramod, A., Ghosh, A., Mohan, A., Shrivastava, M., & Shettar, R. (2015). SQLI detection system for a safer web application. In *2015 IEEE International Advance Computing Conference (IACC)* (pp. 237-240). IEEE Publishing. https://doi.org/10.1109/IADCC.2015.7154705

Radack, S., & Kuhn, R. (2011). Managing security: The security content automation protocol. *IT Professional*, *13*(1), 9-11. https://doi.org/10.1109/MITP.2011.11

Rao, K. S., Jain, N., Limaje, N., Gupta, A., Jain, M., & Menezes, B. (2016). Two for the price of one: A combined browser defense against XSS and clickjacking. In *2016 International Conference on Computing, Networking and Communications (ICNC)* (pp. 1-6). IEEE Publishing. https://doi.org/10.1109/ICCNC.2016.7440629

Sadeghian, A., Zamani, M., & Manaf, A. A. (2013). A taxonomy of SQL injection detection and prevention techniques. In *2013 International Conference on Informatics and Creative Multimedia* (pp. 53-56). IEEE Publishing. https://doi.org/10.1109/ICICM.2013.18

Sarmah, U., Bhattacharyya, D. K., & Kalita, J. K. (2018). A survey of detection methods for XSS attacks. *Journal of Network and Computer Applications*, *118*, 113-143. https://doi.org/10.1016/j.jnca.2018.06.004

Scarfone, K., & Mell, P. (2009). An analysis of CVSS version 2 vulnerability scoring. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement* (pp. 516-525). IEEE Publishing. https://doi.org/10.1109/ESEM.2009.5314220

Singh, A. K., & Roy, S. (2012). A network based vulnerability scanner for detecting SQLI attacks in web applications. In *2012 1st International Conference on Recent Advances in Information Technology (RAIT)* (pp. 585-590). IEEE Publishing. https://doi.org/10.1109/RAIT.2012.6194594

Voeller, J. G. (2008). *Wiley Handbook of Science and Technology for Homeland Security*. John Wiley & Sons, Inc. https://doi.org/10.1002/9780470087923

Wang, C. H., & Zhou, Y. S. (2016). A new cross-site scripting detection mechanism integrated with HTML5 and CORS properties by using browser extensions. In *2016 International Computer Symposium (ICS)* (pp. 264-269). IEEE Publishing. https://doi.org/10.1109/ICS.2016.0060