# A Detection and Prevention Technique on SQL Injection Attacks

Zar Chi Su Su Hlaing
*Faculty of Information Science*
*University of Computer Studies (Magway)*
Magway, Myanmar
zarchissh@gmail.com

Myo Khaing
*Faculty of Computer Science*
*University of Computer Studies (Maubin)*
Maubin, Myanmar
myokhaingucsm@gmail.com

***Abstract*—With the web advancements are rapidly developing, the greater part of individuals makes their transactions on web, for example, searching through data, banking, shopping, managing, overseeing and controlling dam and business exchanges, etc. Web applications have gotten fit to numerous individuals' day by day lives**

serious dangers to the security of backend database fror driven applications.

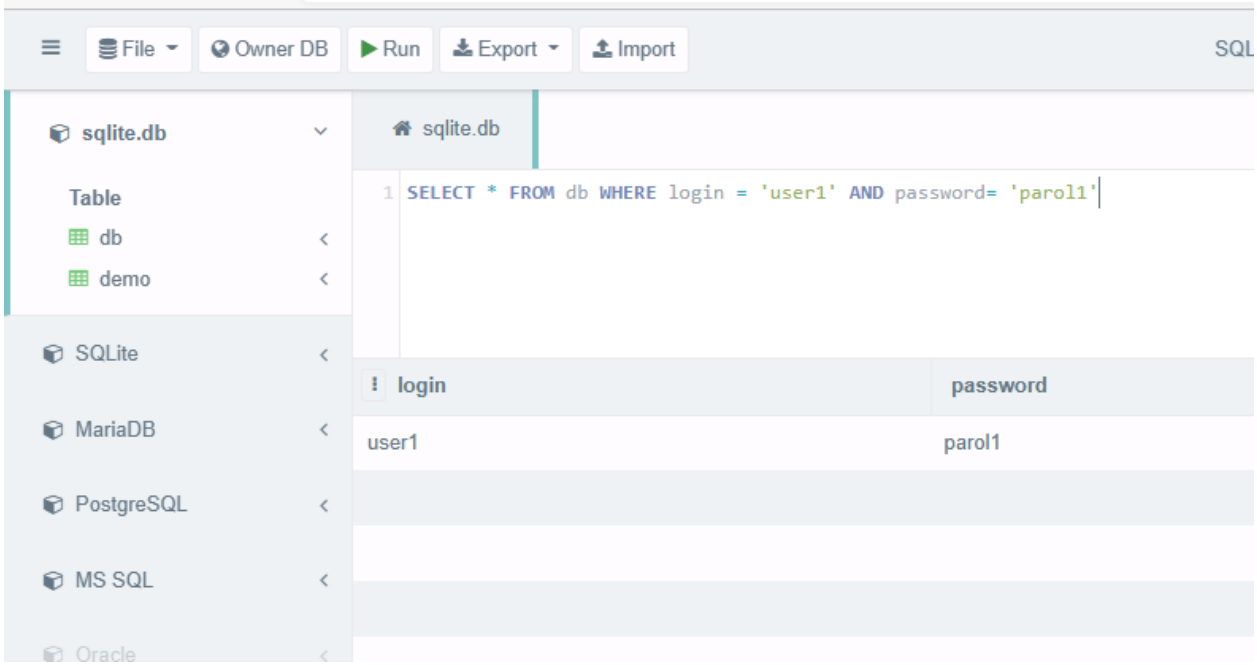
SQL injection is an assault method with negated SQI articulations used to abuse how site pages speak with back end databases. It can take a shot at defenseless website page

sqlite.db

```
1 SELECT * FROM db
```

| login | password |
|---|---|
| user1 | parol1 |
| user2 | parol2 |
| user3 | parol3 |
| user4 | parol4 |
| user5 | parol5 |
| user6 | parol6 |
| user7 | parol7 |
| user8 | parol8 |
| user9 | parol9 |
| user10 | parol10 |

```
In [4]: login = str(input('введите логиг:'))
        password = str(input('введите пароль:'))
        query_sql = f"select * from db where login = '{login}' and password= '{password}'"
        print('итоговый запрос:')
        print(query_sql)

        введите логиг:user1
        введите пароль:parol1
        итоговый запрос:
        select * from db where login = 'user1' and password= 'parol1'
```



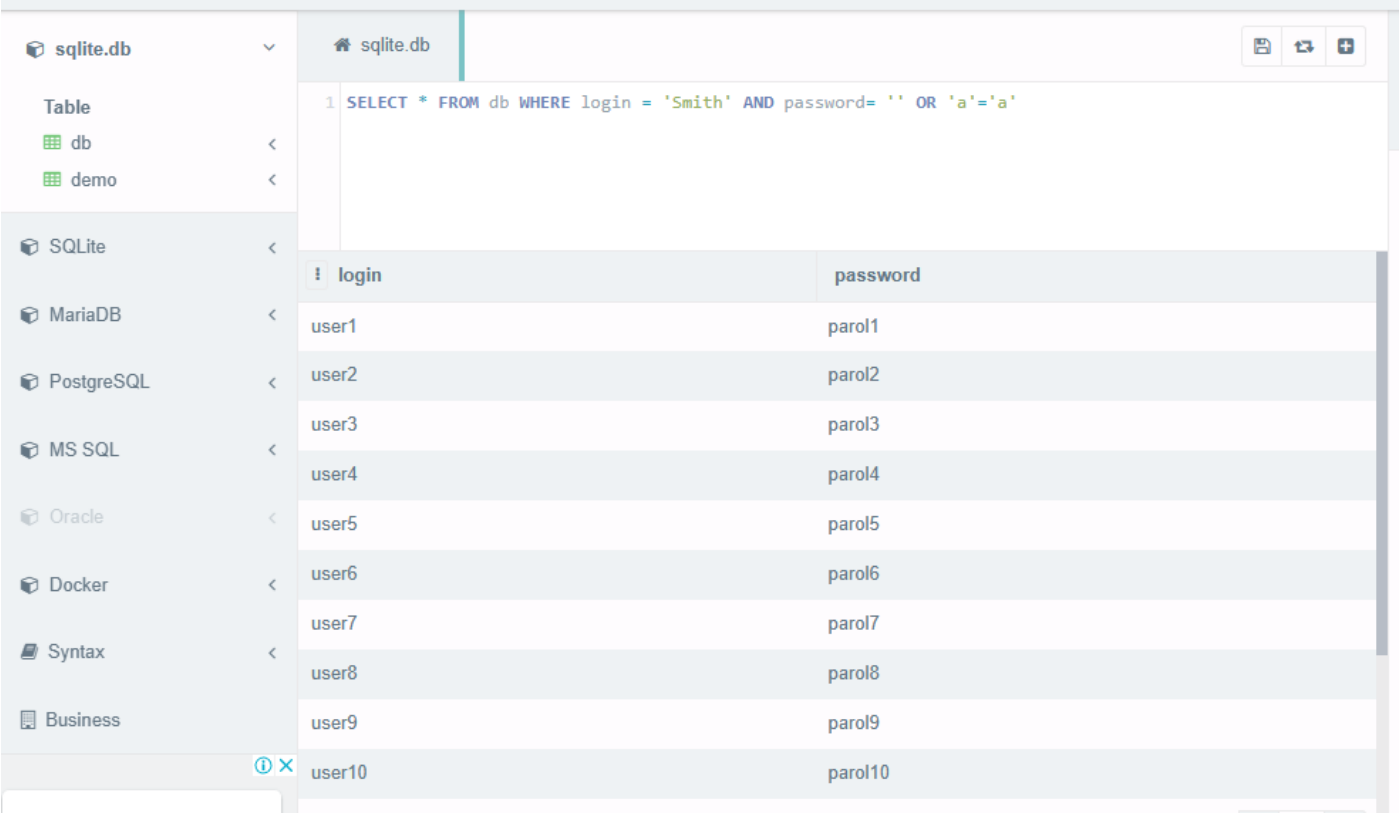

**Пример работы инъекций из статьи:**

| | |
|---|---|
| Smith<br>' OR 'a'='a | SELECT name FROM member WHERE username='Smith' AND password= ' ' OR 'a'= 'a' |
| ' OR 'a'='a<br>' OR 'a'='a | SELECT name FROM member WHERE username =' ' OR 'a'='a' AND password=' ' OR 'a'='a' |
| 'OR 'a'='a'<br>- - | SELECT name FROM member WHERE username =' ' OR 'a'='a' - - ' AND password=' ' |

**Проверка:**

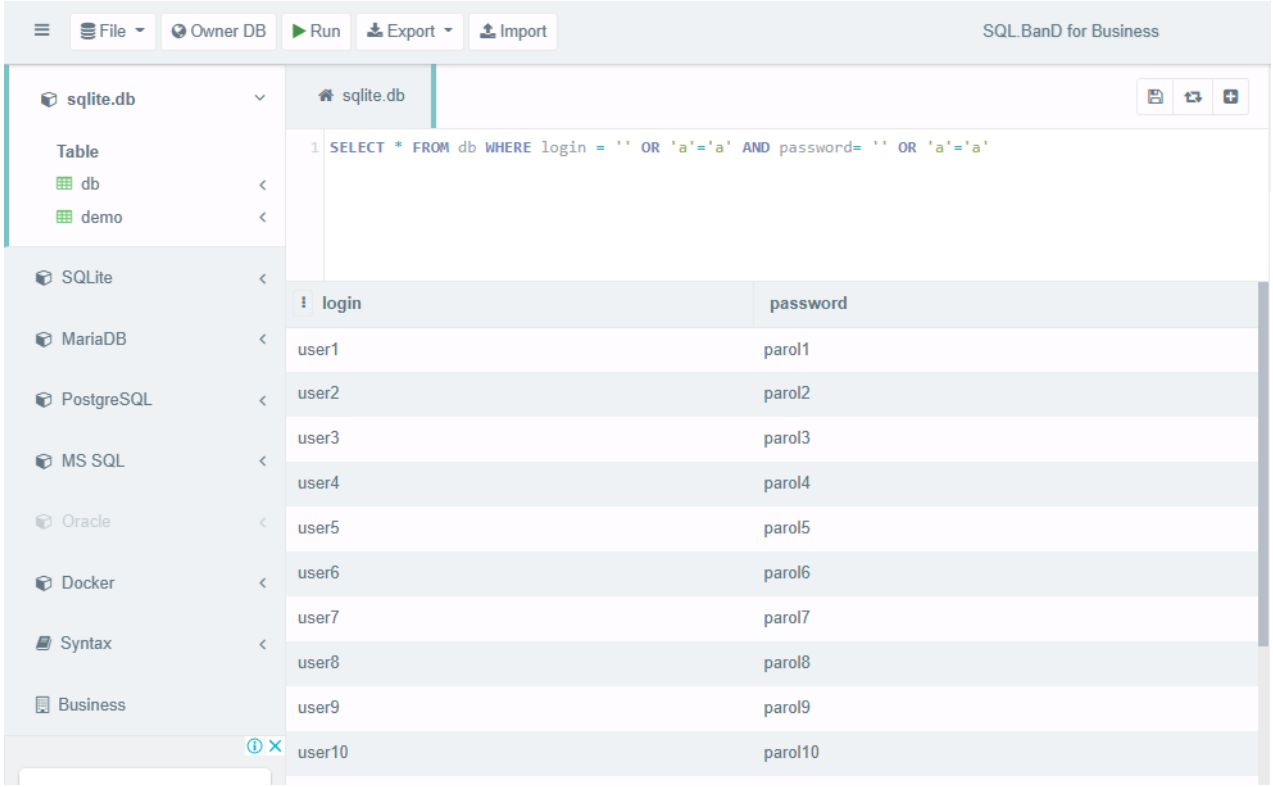
```python
login = str(input('введите логиг:'))
password = str(input('введите пароль:'))
query_sql = f"select * from db where login = '{login}' and password= '{password}'"
print('итоговый запрос:')
print(query_sql)
```

```
введите логиг:Smith
введите пароль:' OR 'a'='a
итоговый запрос:
select * from db where login = 'Smith' and password= '' OR 'a'='a'
```




```python
login = str(input('введите логиг:'))
password = str(input('введите пароль:'))
query_sql = f"select * from db where login = '{login}' and password= '{password}'"
print('итоговый запрос:')
print(query_sql)
```

```
введите логиг:' OR 'a'='a
введите пароль:' OR 'a'='a
итоговый запрос:
select * from db where login = '' OR 'a'='a' and password= '' OR 'a'='a'
```

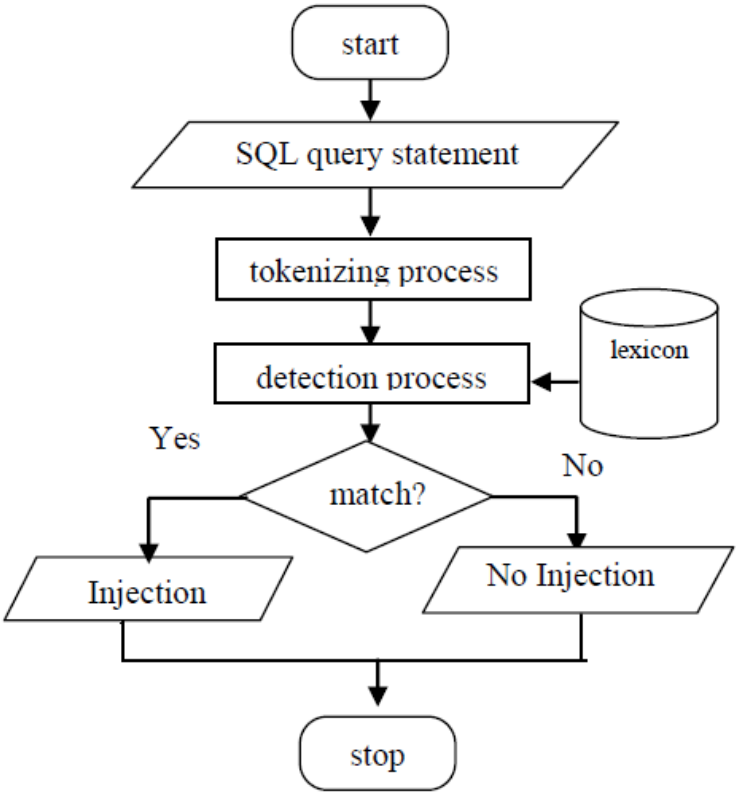



**Создание алгоритма защиты:**

**Суть алгоритма – не пропускать запросы, содержащие ключевые слова, которые являются командами SQL:**

| No | Injected command |
|---|---|
| 1 | alter |
| 2 | concat |
| 3 | drop |
| 4 | delete |
| 5 | execute |
| 6 | sleep |
| 7 | shutdown |

| | |
|---|---|
| 8 | union |
| 9 | or |
| 10 | if |




```python
import string
login = str(input())
password = str(input())
keyword = ['alter','concat','drop','delete','execute','sleep','shutdown','union','or','if']
query_sql = f"select * from db where login = '{login}' and password= '{password}'"
flag = False
rez = query_sql
for p in string.punctuation:
    if p in rez:
        rez = rez.replace(p, '')
for i in rez.split():
    for j in keyword:
        if i.lower() == j:
            flag = True
            break
if flag == True:
    print('Запрос опасен:\n',query_sql)
else:
    print('Запрос безопасен:\n',query_sql)
```

Проверка алгоритма:

TABLE III. " SQL QUERY STATEMENT

| input | SQL statement |
|---|---|
| smith<br>123 | SELECT * FROM member WHERE username ='smith' AND password ='123' |
| ' or '1=1<br>' or '1=1 | SELECT * FROM member WHERE username =' ' or '1=1' AND password =' ' or '1=1' |
| smith<br>' or 'a'='a | Select * from member where username ='smith' and password =' ' or 'a '=' a' |
| ' or "='<br>' or "=' | SELECT * FROM member WHERE username =' ' or ' '=' ' AND password =' ' or ' '=' ' |
| smith<br>' or "=' | SELECT * FROM member WHERE username ='smith' AND password =' ' or ' '=' ' |
| ' or '1=1'--<br>123 | SELECT * FROM member WHERE username =" or '1=1' --' AND password ='123' |
| "'; DELETE FROM member WHERE 1 or username = '"; | SELECT * FROM member WHERE username=' '; DELETE FROM member WHERE 1 or username = ' ' |
| "; SHUTDOWN; - - | SELECT name FROM member WHERE username="; SHUTDOWN; - - password=" |

```
smith
123
Запрос безопасен:
 select * from db where login = 'smith' and password= '123'
```

```
' or '1=1
' or '1=1
Запрос опасен:
 select * from db where login = '' or '1=1' and password= '' or '1=1'
```

```
 Smith
 ' or 'a'='a
 Запрос опасен:
  select * from db where login = 'Smith' and password= '' or 'a'='a'
```

```
' or ''='
' or ''='
Запрос опасен:
 select * from db where login = '' or ''='' and password= '' or ''=''
```

```
Smith
' or ''='
Запрос опасен:
 select * from db where login = 'Smith' and password= '' or ''=''
```

```
' or '1=1'--
123
Запрос опасен:
 select * from db where login = '' or '1=1'--' and password= '123'
```

```
"'; DELETE FROM
member WHERE 1 or username = '";
Запрос опасен:
 select * from db where login = '"'; DELETE FROM' and password= 'member WHERE 1 or username = '";'
```

```
 ''; SHUTDOWN; - -

 Запрос опасен:
  select * from db where login = '''; SHUTDOWN; - -' and password= ''
```

```
smith
''; DROP table users - -
Запрос опасен:
 select * from db where login = 'smith' and password= '''; DROP table users - -'
```

```
john
john123
Запрос безопасен:
 select * from db where login = 'john' and password= 'john123'
```

```
blake
blake123
Запрос безопасен:
 select * from db where login = 'blake' and password= 'blake123'
```

В итоге 3 теста из 10 являются безопасными, что соответствует итоговым результатам в статье:

| | | Prediction | | Total |
|---|---|---|---|---|
| | | *normal* | *injection* | |
| **Actual** | *normal* | 3 | 0 | 3 |
| | *injection* | 0 | 7 | 7 |
| | | 3 | 7 | 10 |

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

TABLE V. " EXPERIMENT OUTCOMES

| SQLIA Techniques | Proposed approach's outcomes |
|---|---|
| Tautologies | Successful prevention |
| Malformed queries | Successful prevention |
| Union queries | Successful prevention |
| Piggy-back queries | Successful prevention |
| Inference | Successful prevention |
| Stored procedure | Successful prevention |

proposed approach is used for the detection and preventio of SQL injection and also suitable the outcomes.


ACKNOWLEDGMENT

I would like to express my deepest thanks to all m teachers for their valuable advice, helpful comments, an precious time for this research. Most importantly, none c this would have been possible without the love and patienc of my family throughout the process. My heartfelt thank also extend to all my colleagues and friends for their hel interest and valuable hints for discussions about this.


REFERENCES


[1]" N. Lambert, K.S. Lin; "Use of Query tokenization to detect an prevent SQLinjection attacks", *Proceedings of the 3rd Internationa Conference on Computer Science and Information Technolog (ICCSIT),* Chengdu, China:IEEE (2010). pp: 438-440, 2010.

[2]" I. Balasundaram, E. Ramaraj, "An Efficient Technique for Detectio and Prevention of SQL Injection Attack using ASCII Based Strin Matching", International Conference on Communication Technolog and System Design, Prodedia Engineering, pp. 183-190, 2012.

[3]" Dr. R. Shettar, A. Ghosh, A. Mohan, A. Pramod, C. Raikar, " SQ

**Статья 2**

## IV. SQLI COUNTERMEASURES

*Disable unused features*: Disable all features that you do not use. It is not necessary to keep functions of your server that you do not use because they are potentially dangerous for you.

*Custom error message*: When running the query, run a test to see if it returns an error. If there is an error, put a custom error. SQL errors give too much information to a malicious user.

*Escape functions*: The escape functions are very easy to set up and allow you to quickly secure your server against most attacks.

```
SHUTWOWN
...
```

Fig. 5. Type of danger function

*Limit the size of the data*: You can limit the size of the data entered by the user. We saw that some injections required a certain number of characters (you can limit the ID numbers to five characters, a login to 15 for example).

```
<input type="number" name="id" min="1" max="99999" size="5" />
<input type="text" name="lastname" size="15" />
```

Fig. 7. HTML input with limitation of size

*Use the Prepare Statements*: Use the prepare statements. This solution remains the most effective to protect against SQL injections. It will ask you a little more time (not much more) but will effectively secure your server.

*Use the Prepare Statements*: Use the prepare statements. This solution remains the most effective to protect against SQL injections. It will ask you a little more time (not much more) but will effectively secure your server.

PHP

```
$query = "INSERT `teachers`(`CodeT`, `NameT`, `RankT`, `SpecialtyT`,
`StatusT`) VALUES(:code, :name, :rank, :specialty, :status)";
$datas = array (
            'code'=> $_POST['code'],
            'name'=> $_POST['name'],
            'rank'=> $_POST['rank'],
            'specialty'=> $_POST['specialty'],
            'status'=> $_POST['status']
);
$statement = $db->prepare($query);
$results=$statement->execute($datas);
```

Fig. 8. Example prepare statement using PHP

**Статья 3**

# A proposed approach for preventing Cross-Site Scripting


Twana Assad TAHA
Department of Software Engineering
Firat University
twana.assad@gmail.com

Murat KARABATAK
Department of Software Engineering
Firat University
mkarabatak@firat.edu.tr



***Abstract*— In this paper, the great threat Cross-Site Scripting (XSS) is introduced that faced with the web pages. Because of the impacts of such web threats during design and developing web pages, web developers must be aware and have adequate knowledge about varies type of web attacks and how to prevent or mitigate them. Web developers should have knowledge about how attackers attack websites and exploit weak points on websites during filling forms, registering and opening suspicious links or attachments in emails. The important of this subject is to provide great details and information about identifying**


web attackers have also become the biggest issue for web server, so web designers need to create these components, incorporating the applicable criteria that follow.

Cross-Site Scripting (XSS) threat is a code injection attack that lets an attacker to implement or execute harmful Java Script or other web script in browser of users. The gap (weak point) lets injection of inputs comprising HTML tags and client-side scripts code [1]. The code of XSS can be composed in any client-side scripting language. But JavaScript is used widely than other web scripts. This attack can likewise be positioned through a link in an Email or on a web page that

TABLE III. COMMON CHARACTERS REPLACING TO DEFEAT XSS

| Replace | With |
|---|---|
| < | &lt |
| > | &gt |
| ( | &#40 |
| ) | &#41 |
| # | &#35 |

XSS threats can be avoided by validating and checking data that are provided by users to ensure consistence with the required format for web applications, there are four suggested mechanisms for user input validation [10,11].

- **Replacement** is a way to search for dangerous user inputs then substitutes those dangerous codes with correct and true characters.
- **Removal** is a way also to find dangerous inputs but opposed to replacement by removing them.
- **Escaping** way changes (or marks) key characters of the data to avoid it from being interpreted in a dangerous code.
- **Restriction** way checks the user inputs to limited non-malicious.

OWASP's guide to secure development gives three rules for dealing with user data [12]:

- Accept only known valid data
- Reject recognized harmful data
- Clean harmful data

## V. APPROACHED SYSTEM TO PREVENT XSS

XSS is to be stated how to happen. it takes place when web forms receive malicious scripting code that has been injected to the victim computer then the web browser will execute. In this approached system, secure code PHP functions are proposed to detect and prevent form XSS attack by using two methods, the first one is to use regular expression to validate data from web forms that has been entered by the user, and the second one is

HTML starting tags and ending tags <> with any text inside

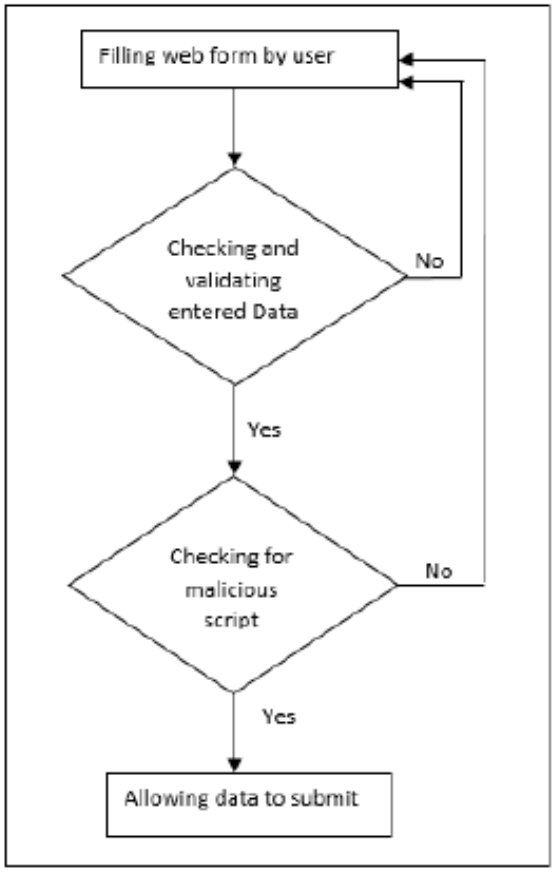
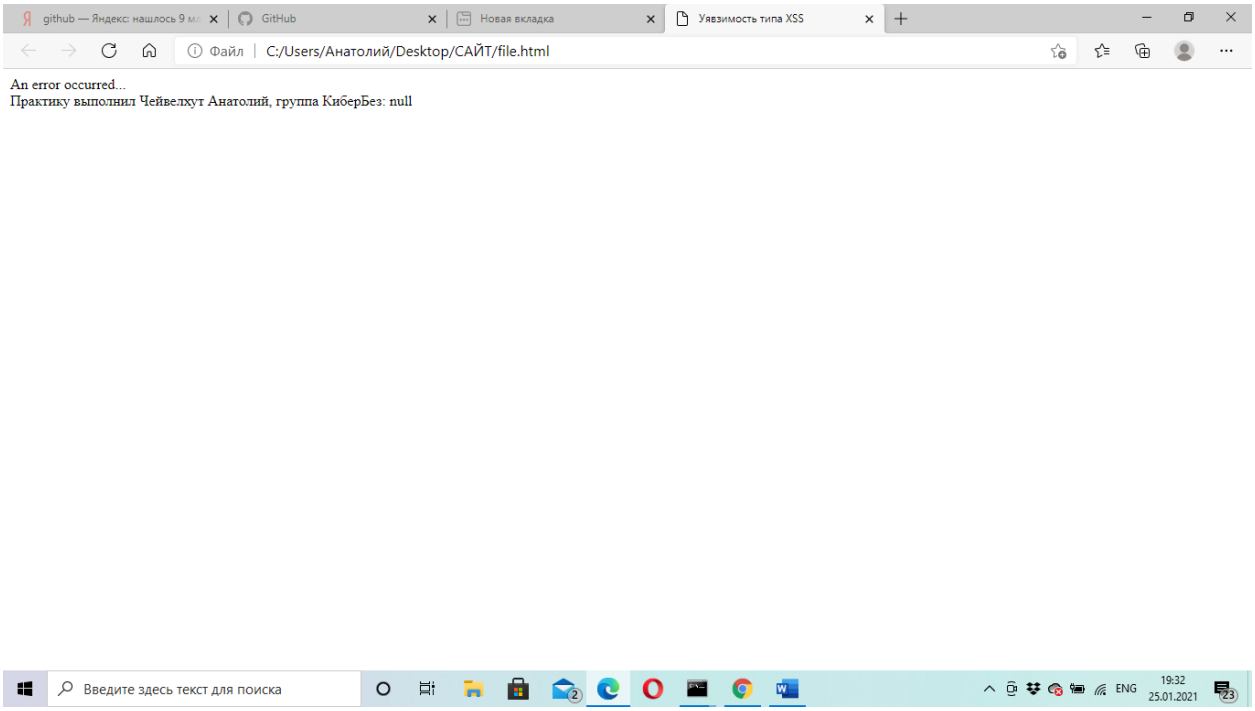



Figure 2. approached system to prevent XSS flowchart

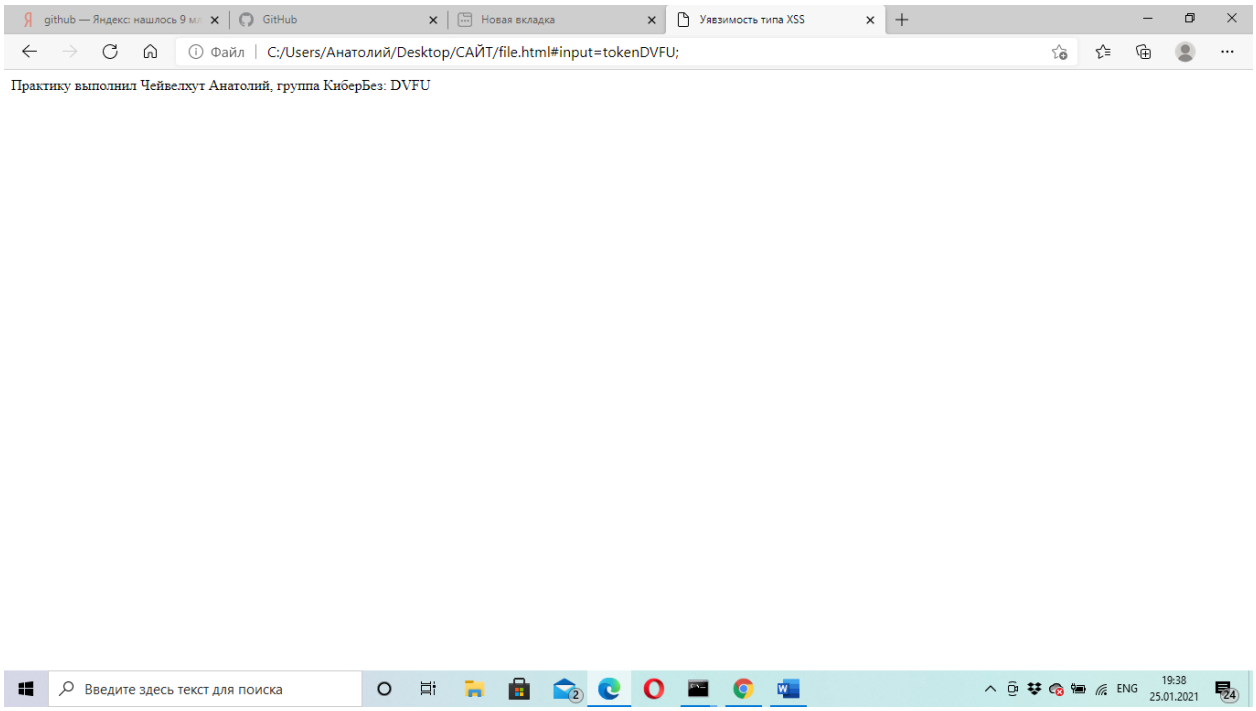
Algorithm (1): The Proposed System to prevent XSS Attack Process

```
Initialize AllowList_RegExp
[
```

Практическая работа. XSS уязвимости на примере html-документа

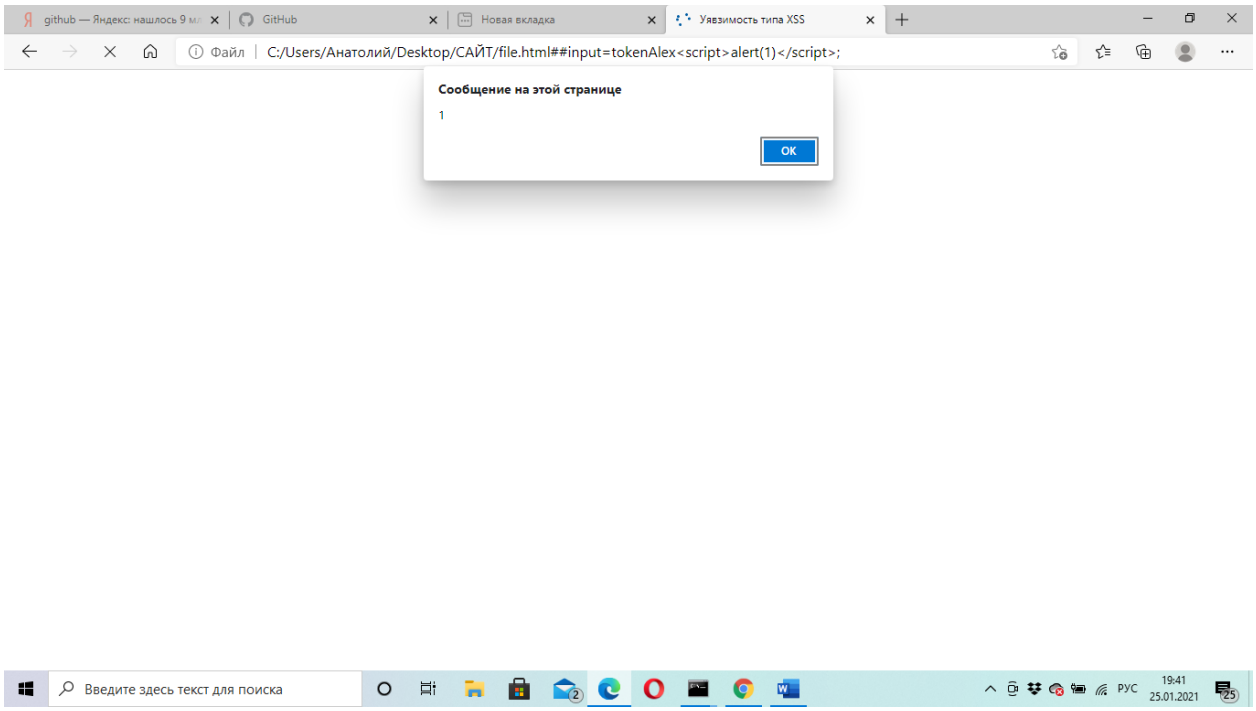


Добавим в адресную строку к текущему адресу следующее значение: **#input=tokenDVFU;**




**При обновлении адресной строки значение, которое находилось в адресной строке присвоилось переменной внутри скрипта, которое впоследствии вывелось на экран.**

**Затем заменим предыдущую команду на следующую:**
`#input=tokenAlex<script>alert(1)</script>;`



**Данная последовательность символов содержала в себе команду alert, цель которой – вывод сообщения на экран. Таким образом, это не единственная команда, которая может вводиться через данную уязвимость. Таким способом можно воровать cookie посетителей и прочую информацию.**

## Вставка функции фильтрации запросов

### Исходные код HTML с кодом JavaScript

file.html – Блокнот

Файл Правка Формат Вид Справка

```
<!DOCTYPE html>
<html>
    <head>
        <title>Уязвимость типа XSS</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div id="default"> An error occurred...</div>

        <script>
            function OnLoad() {
                var foundFrag = get_fragment();
                return foundFrag;
            }

            function get_fragment() {
                var r4c = '(.*?)';
                var results = location.hash.match('.*input=token(' + r4c + ');');

                if (results) {
                    document.getElementById("default").innerHTML = "";
                    return (unescape(results[2]));
                } else {
                    return null;
                }
            }

            display_session = OnLoad();
            document.write("Практику выполнил Чейвелхут Анатолий, группа КиберБез: " + display_session + "<br><br>")
        </script>

    </body>
</html>
```

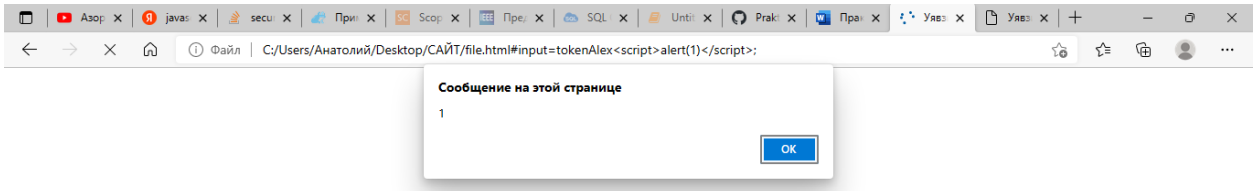
### Копируем данный HTML документ, вставляем туда функцию фильтрации

file — копия.html – Блокнот

Файл Правка Формат Вид Справка

```
    <body>
        <div id="default"> An error occurred...</div>

        <script>
            function OnLoad() {
                var foundFrag = get_fragment();
                return foundFrag;
            }
function escapeOutput(toOutput){
    return toOutput.replace(/\&/g, '&amp;')
        .replace(/\</g, '&lt;')
        .replace(/\>/g, '&gt;')
        .replace(/\"/g, '&quot;')
        .replace(/\'/g, '&#x27')
        .replace(/\//g, '&#x2F');
}

            function get_fragment() {
                var r4c = '(.*?)';
                var results = location.hash.match('.*input=token(' + r4c + ');');

                if (results) {
                    document.getElementById("default").innerHTML = "";
                    return escapeOutput(unescape(results[2]));
                } else {
                    return null;
                }
            }

            display_session = OnLoad();
            document.write("Практику выполнил Чейвелхут Анатолий, группа КиберБез: " + display_session + "<br><br>")
        </script>

    </body>
</html>
```

## Старая версия HTML документа




## Обновленная версия