Software Defined Networks – Technical Report

NATIONAL INSTITUTE OF TECHNOLOGY PUDUCHERRY
KARAIKAL – 609 609

Naghul Pranav A S (CS22B1037)	Tharaneeshwaran V U (CS22B1056)	Krishna J (CS22B1023)

Abstract

Software-Defined Networking (SDN) represents a revolutionary approach to network management by decoupling control and data planes, providing centralized control and automation. This paper discusses its advantages, including enhanced agility, scalability, and security, but also acknowledges the challenges it poses, such as security issues, scalability problems, and increased complexity. Strategies to mitigate these challenges involve implementing robust security measures, optimizing controller performance, and offering training. Overall, SDN offers a promising way to manage networks, but success hinges on careful planning and execution.

Shifting focus to Selenium, an open-source software testing framework tailored for automating web applications, this paper underscores its pivotal role in facilitating functional and regression testing across various programming languages. Selenium enables the creation of scripts that emulate user interactions with web applications, automating repetitive testing procedures.

Furthermore, the paper delves into the concept of brute force attacks used by cybercriminals to gain unauthorized access to systems, accounts, and encrypted data by systematically trying different password combinations. The paper acknowledges the various forms of brute force attacks, such as simple brute force, reverse brute force, dictionary attacks, and credential stuffing, emphasizing the need for preventative measures to thwart such attacks.

Lastly, the paper addresses SQL injection (SQLi) vulnerabilities, which allow attackers to manipulate database queries to access unauthorized data. It highlights the potentially devastating consequences of SQLi attacks, including data loss and application disruption, and outlines the foundation they provide for more complex Denial-of-Service (DoS) attacks. The paper introduces the principles of SQL injection attacks and outlines methods for preventing them.

In conclusion, this paper covers the pivotal topics of SDN, Selenium, brute force attacks, and SQL injection vulnerabilities, emphasizing the importance of proactive measures to enhance security and safeguard digital assets and data integrity.

Keywords: SQL Injection, Web Security, Selenium, SDN, Software Defined Networks, Brute Force, Cyberattack

Overview

Introduction:

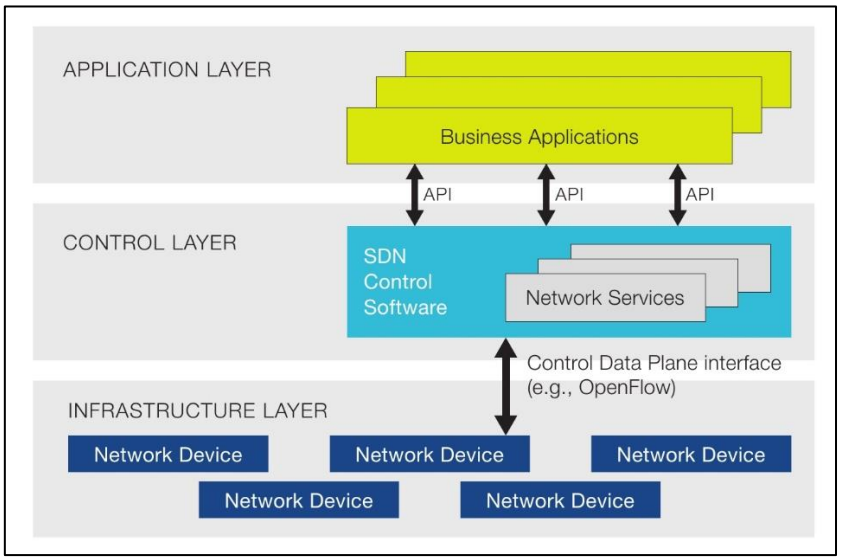
SDN, or Software-Defined Networking, is a transformative approach to network management. It replaces traditional hardware-based networks with a flexible software-driven model. Key components include a controller for centralized control, southbound APIs for communication with network devices, and northbound APIs for applications. SDN offers benefits like flexibility, centralized control, automation, optimization, and cost savings. It finds applications in data centers, wide area networks, network security, and 5G networks. Challenges include security concerns, interoperability, and a learning curve.

Key Components:

- **Control Plane:** This is where the decisions about how to route data and manage network policies are made. In SDN, this control plane is abstracted from the physical network devices and centralized in a software controller.
- **Data Plane:** The data plane is responsible for the actual forwarding of network traffic. It includes the physical network devices such as switches and routers.
- **Software Controller:** Think of the software controller as the brain of the SDN network. It manages and controls the network devices by sending instructions to them. These controllers can be implemented using various software platforms, some open-source and others proprietary.
- **OpenFlow Protocol:** OpenFlow is a standardized communication protocol that lets the controller interact with and manage the behaviour of network switches and routers. It provides a common way to program and manage network flows.
- **Network Programmability:** SDN allows network administrators to program and control network behaviour using software-based interfaces. This makes it easier to implement and manage complex networking tasks and policies.

Benefits of SDN:

- **Centralized Control:** SDN provides a central point of control for network traffic and policies, simplifying network-wide changes and updates.
- **Dynamic Configuration:** Network settings and policies can be adjusted swiftly without manual reconfiguration of individual devices.
- **Automation:** SDN automates many network management tasks, reducing manual work, improving efficiency, and lowering the risk of human errors.
- **Resource Optimization:** Centralized control and dynamic routing in SDN networks enhance resource utilization and efficient traffic routing, potentially leading to cost savings.
- **Network Virtualization:** SDN supports network virtualization, enabling multiple virtual networks to operate on the same physical infrastructure, particularly beneficial in cloud environments.
- **Enhanced Security:** SDN allows for more granular and adaptable security policies, enhancing the detection and response to security threats.
- **Quality of Service (QoS):** SDN can prioritize and manage different types of traffic, ensuring critical applications receive necessary resources for optimal performance.
- **Scalability:** SDN architectures are designed for easy scalability, allowing the addition of new devices or segments with minimal disruption through centralized programming.
- **Vendor Neutrality:** SDN's reliance on open standards and APIs enables organizations to avoid vendor lock-in, providing more flexibility and choice in technology adoption.
- **Alignment with Cloud Computing:** SDN is well-suited for dynamic and rapidly changing cloud environments, aligning with cloud computing principles and facilitating seamless integration between network and cloud services.



The image depicts 'SDN Architecture'

Challenges of SDN:

- **Security Concerns:** Centralized control in SDN creates a single point of failure and a potential target for cyberattacks, necessitating robust security measures for the controller and communication with network devices.
- **Controller Scalability:** As the network grows, ensuring the central controller's performance and scalability to handle a large number of devices and requests becomes crucial.
- **Network Reliability:** Controller unavailability can disrupt network services, requiring mechanisms to handle controller failures and maintain network continuity.
- **Migration and Integration:** Transitioning to SDN and integrating it with existing infrastructure requires careful planning to avoid disruptions.
- **Standardization and Interoperability:** Ensuring interoperability among different SDN components and vendor solutions can be challenging due to variations in implementations.
- **Vendor Lock-In:** Choosing a specific SDN solution may lead to vendor lock-in, limiting flexibility.
- **Network Complexity:** Initial SDN setup and configuration can be complex, demanding administrators with the necessary skills.
- **Latency and Performance:** Software-based control may introduce latency, impacting real-time applications.
- **Quality of Service (QoS):** Managing QoS policies becomes complex in dynamic SDN environments.
- **Lack of Standards:** While OpenFlow standardizes the data plane, there may be gaps in other SDN aspects like controller communication and APIs.
- **Operational Complexity:** SDN introduces new processes and workflows, requiring network administrators to adapt to a different management approach.
- **Regulatory and Compliance:** Compliance with industry-specific regulations and standards can complicate SDN implementation.
- **Training and Skill Gap:** Administrators need to acquire new skills and knowledge to effectively manage SDN environments, which may require time and resources.

Strategies to Improve:

- **Security:** Implement strong security measures for controllers and communication.
- **Controller Scalability:** Choose a scalable controller architecture, use load balancers, and optimize software.
- **Network Reliability:** Implement redundancy and disaster recovery plans.
- **Migration and Integration:** Start with pilots, develop migration plans, and consider network overlays.
- **Standardization:** Choose open standards and engage with industry groups.
- **Vendor Neutrality:** Evaluate multiple vendor solutions and plan for vendor transitions.
- **Network Complexity:** Provide training, hire skilled staff, and use automation tools.
- **Latency and Performance:** Optimize controller software and prioritize latency-sensitive traffic.
- **QoS Complexity:** Carefully plan QoS policies and adapt to changing conditions.
- **Lack of Standards:** Engage with standards organizations to promote standardization.
- **Operational Complexity:** Provide training and user-friendly management tools.
- **Regulatory Compliance:** Work with legal teams to ensure compliance.

Selenium – Automation Tool

Introduction:

Selenium stands as a robust open-source software testing framework specifically designed for automating web applications. Its primary role is to facilitate functional and regression testing, supporting various programming languages such as Java, Python, C#, and more. Selenium plays a crucial role in the automation of repetitive testing procedures by enabling the creation of scripts that simulate user interactions with web applications.

Historical Development:

Selenium's origins date back to 2004 when Jason Huggins initially developed it as an internal tool for web application testing at ThoughtWorks. He humorously named it "Selenium" after the element selenium, which is utilized in certain medicines for treating mercury poisoning. The software was open-sourced in the same year, gaining swift adoption and fostering a vibrant Selenium community. Over the years, Selenium has evolved, with various versions and components being introduced.

Challenges with Manual Testing:

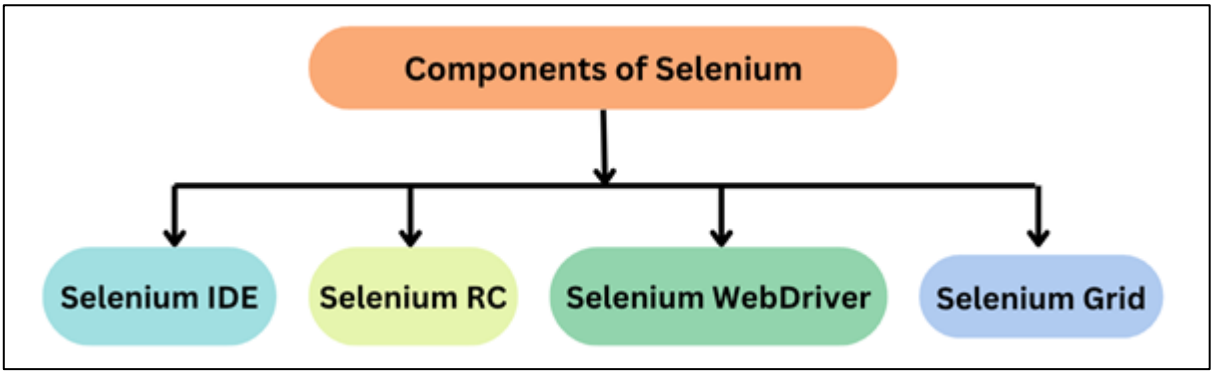
Manual testing is one of the primitive ways of software testing. It doesn't require the knowledge of any software testing tool and can practically test any application. The tester manually executes test cases against applications and compares the actual results with desired results. Any differences between the two are considered as defects and are immediately fixed. The tests are then re-run to ensure an utterly error-free application.

Manual testing has its drawbacks:

- It's extremely time-consuming and there's a high risk of error
- It requires the presence of a tester 24/7
- Requires manual creation of logs
- It has a limited scope

Components:

Selenium has a dedicated suite that facilitates easy testing of web applications.



Selenium IDE

It is a Chrome and Firefox plugin. The primary use of a Selenium IDE is to record user interactions such as clicks, selections etc in the browser and plays them back as automated test. It then generates the test script (of the automated tests) in programming languages like C#, Java, Python, and Ruby and Selenese (Selenium's scripting language).

It helps in:

- Creating automated test scripts and validating them at speed
- Identifying and highlighting errors during the replay of interactions
- Cross Browser Testing

Selenium RC

Selenium RC was built to automate the testing of web applications by simulating user interactions across different browsers and platforms. It provided a way to browser automation remotely and execute test scripts written in various programming languages. Limitations of Selenium RC:

- **Browser Limitations:** Selenium RC had to work with browsers using a JavaScript-based "proxy" mechanism, which introduced potential instability and limitations, especially when working with modern web applications.
- **Speed and Performance:** The use of a JavaScript proxy added overhead and affected the speed and performance of test execution.
- **Maintenance and Compatibility:** Selenium RC required separate "drivers" for each browser, making maintenance and compatibility challenging as browsers continued to update and evolve.
- **Synchronization Issues:** Selenium RC often faced synchronization problems, where test scripts had to wait for the browser to respond before proceeding to the next step.
- **Complex Setup:** Setting up Selenium RC involved multiple components, which could be complex and difficult to configure correctly.

Selenium WebDriver

It is a powerful and enhanced version of Selenium RC which was developed to overcome the limitations of Selenium RC. WebDriver communicates with browsers directly with the help of browser-specific native methods, thereby eliminating the need of Selenium RC. WebDriver works closely with Selenium IDE and Selenium Grid resulting in reliable test execution at speed and scale.

Selenium Grid

This is a smart proxy server that allows QAs to run tests in parallel on multiple machines. This is done by routing commands to remote web browser instances, where one server acts as the hub. This hub routes test commands that are in JSON format to multiple registered Grid nodes.

Practical usage & Applications:

Selenium's utility stems from its ability to automate web applications by emulating user actions. It empowers testers and developers to draft scripts that instruct Selenium on how to undertake tasks like clicking buttons, inputting text, navigating web pages, and verifying expected outcomes. Subsequently, Selenium executes these instructions on various web browsers.
Selenium finds extensive application within the software industry for the following purposes:

- Functional Testing: Guaranteeing that the application fulfils its intended functions accurately.
- Regression Testing: Verifying that new code changes do not disrupt existing functionality.
- Load Testing: Simulating multiple users to evaluate the application's performance under various conditions.
- Cross-Browser Testing: Ensuring uniform functionality across different web browsers.
- Automating Repetitive Tasks: Automation of monotonous tasks like data entry and report generation.

Advantages:

The advantages associated with Selenium encompass:

- **Open-Source Nature**: Selenium is freely available and benefits from an active and expansive community.
- **Multi-Browser Compatibility**: It extends support to a variety of web browsers, including Chrome, Firefox, Safari, and Edge.
- **Multilingual Support**: Test scripts can be authored in diverse programming languages.
- **Parallel Test Execution**: Selenium Grid permits concurrent test execution on multiple machines.

- ✧ **Integration with CI/CD**: It seamlessly integrates with Continuous Integration and Continuous Deployment pipelines.
- ✧ **Extensive Documentation and Support:** Selenium users have access to abundant resources and forums for guidance.

Limitations:

Despite its myriad advantages, Selenium presents some limitations:

- ✧ **Complex Setup**: The initial setup of Selenium can prove to be intricate.
- ✧ **Maintenance Requirements**: Frequent script updates are necessary due to changes in the application's user interface.
- ✧ **Limited Desktop Application Support**: Selenium primarily focuses on web applications and offers limited capabilities for testing desktop applications.
- ✧ **Reporting Challenges**: Selenium lacks built-in reporting capabilities, necessitating the integration of supplementary tools for comprehensive reporting.
- ✧ **Mobile Testing Limitations**: While Selenium WebDriver supports mobile app testing to some extent, its capabilities in this domain are not as extensive.

Conclusion:

Selenium emerges as a potent tool for automating web application testing. Its historical development, versatile components, and support for multiple programming languages make it invaluable for manual testers and automation engineers. Nevertheless, users should remain cognizant of its limitations and the challenges associated with maintaining test scripts, particularly in the context of continually evolving web applications. As the software industry progresses, Selenium retains its pivotal role in ensuring the quality and reliability of web applications.

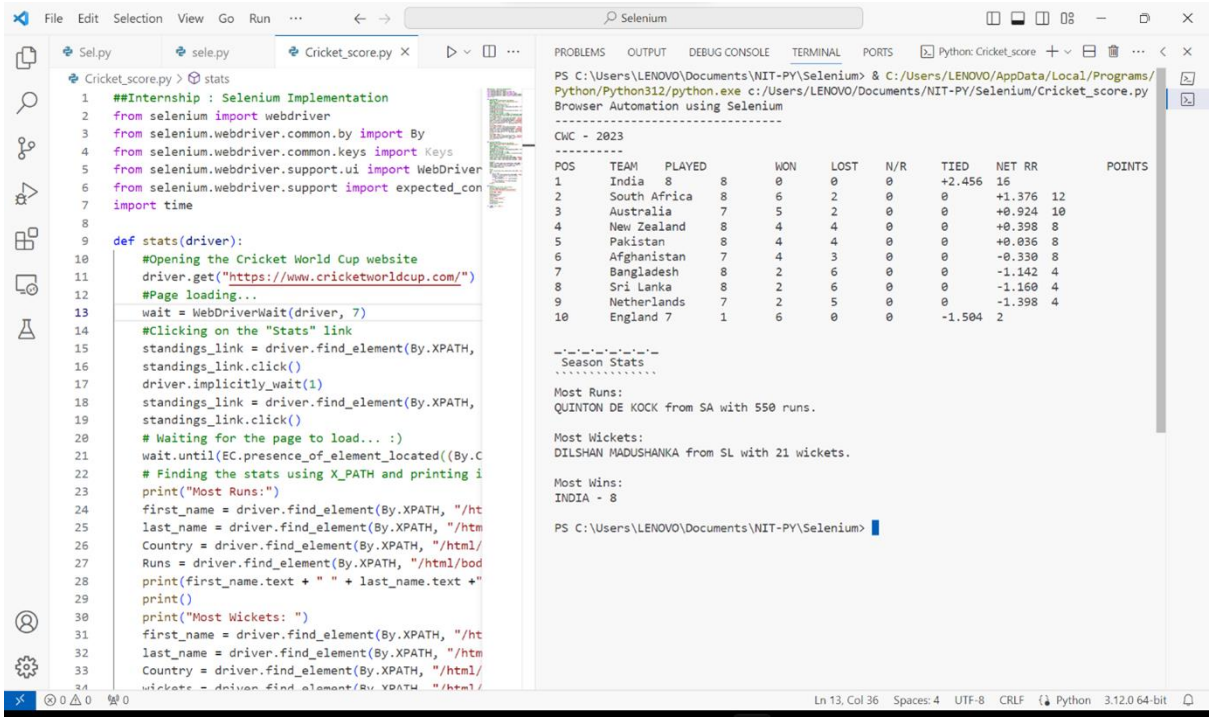
Code Implementations:

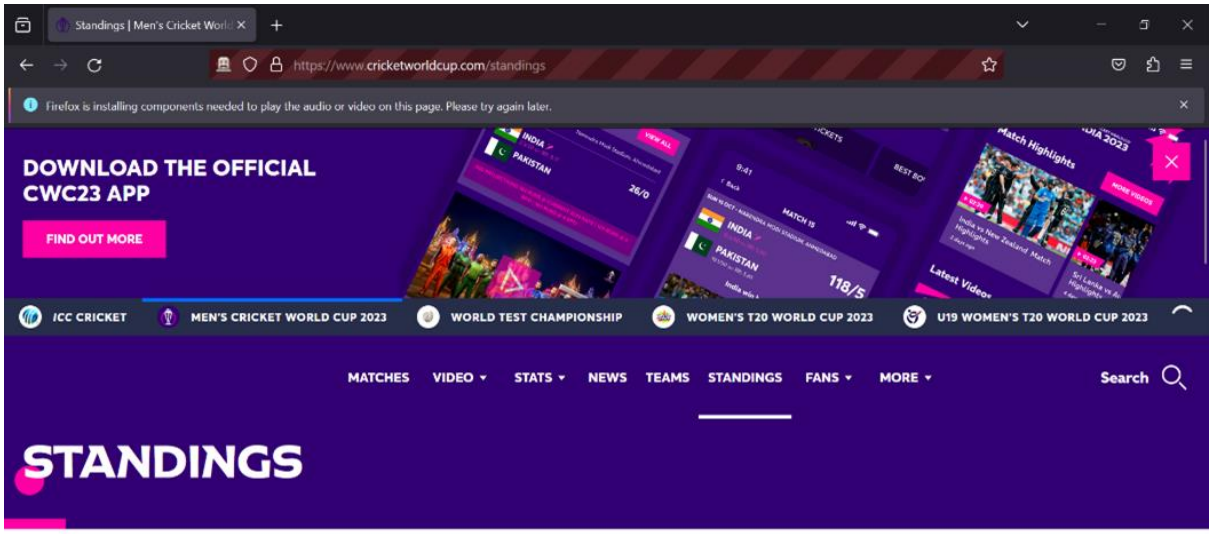
➠ *cricket_score.py*

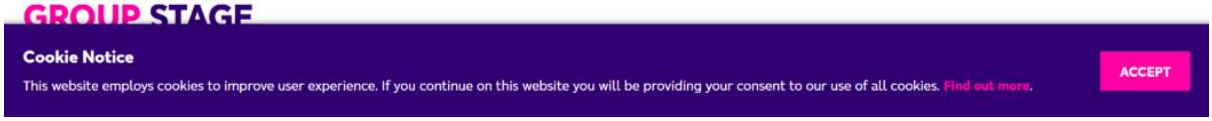
```
##Internship : Selenium Implementation
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time

def stats(driver):
    #Opening the Cricket World Cup website
    driver.get("https://www.cricketworldcup.com/")
    #Page loading...
    wait = WebDriverWait(driver, 7)
    #Clicking on the "Stats" link
    standings_link = driver.find_element(By.XPATH,
"/html/body/div[3]/div/nav[1]/div/div[3]/ul/li[3]/div[1]")
    standings_link.click()
    driver.implicitly_wait(1)
    standings_link = driver.find_element(By.XPATH,
"/html/body/div[3]/div/nav[1]/div/div[3]/ul/li[3]/div[2]/ul/li[1]")
    standings_link.click()
    # Waiting for the page to load... :)
    wait.until(EC.presence_of_element_located((By.CLASS_NAME,
"wrapper")))
    # Finding the stats using X_PATH and printing it(Most runs, wickets and
wins by a team)
    print("Most Runs:")
    first_name = driver.find_element(By.XPATH,
"/html/body/div[5]/main/div/div/div[2]/section[1]/div/div[1]/div/div/div[2]/
div[2]/a/span/span[1]")
    last_name = driver.find_element(By.XPATH,
"/html/body/div[5]/main/div/div/div[2]/section[1]/div/div[1]/div/div/div[2]/
div[2]/a/span/span[2]")
    Country = driver.find_element(By.XPATH,
"/html/body/div[5]/main/div/div/div[2]/section[1]/div/div[1]/div/div/div[2]/
div[2]/div[1]/span")
    Runs = driver.find_element(By.XPATH,
"/html/body/div[5]/main/div/div/div[2]/section[1]/div/div[1]/div/div/div[2]/
div[2]/div[2]")
    print(first_name.text + " " + last_name.text +" from "+ Country.text +"
with "+Runs.text+" runs.")
    print()
    print("Most Wickets: ")
    first_name = driver.find_element(By.XPATH,
"/html/body/div[5]/main/div/div/div[2]/section[2]/div/div[2]/div/div/div[2]/
div[2]/a/span/span[1]")
    last_name = driver.find_element(By.XPATH,
"/html/body/div[5]/main/div/div/div[2]/section[2]/div/div[2]/div/div/div[2]/
div[2]/a/span/span[2]")
    Country = driver.find_element(By.XPATH,
"/html/body/div[5]/main/div/div/div[2]/section[2]/div/div[2]/div/div/div[2]/
div[2]/div[1]/span")
    wickets = driver.find_element(By.XPATH,
"/html/body/div[5]/main/div/div/div[2]/section[2]/div/div[2]/div/div/div[2]/
div[2]/div[2]")
    print(first_name.text + " " + last_name.text +" from "+ Country.text +"
with "+wickets.text+" wickets.")
    print()
    print("Most Wins: ")
    Team_name = driver.find_element(By.XPATH,
"/html/body/div[5]/main/div/div/div[2]/section[3]/div/div[2]/div/div/div[2]/
div[2]/span/span/span")
    wins = driver.find_element(By.XPATH,
"/html/body/div[5]/main/div/div/div[2]/section[3]/div/div[2]/div/div/div[2]/
div[2]/div")
    print(Team_name.text + " - "+ wins.text)
    print()

def Standings(driver):
    #Opening the Cricket World Cup website
    driver.get("https://www.cricketworldcup.com/")
    #Page loading...
    wait = WebDriverWait(driver, 7)
    #Clicking on "Standings" link
    standings_link = driver.find_element(By.XPATH,
"/html/body/div[3]/div/nav[1]/div/div[3]/ul/li[6]/div")
    standings_link.click()
    #Waiting for that page to load
    wait.until(EC.presence_of_element_located((By.CLASS_NAME,
"wrapper")))
    driver.implicitly_wait(2)
    #Finding and printing the table data
    table = driver.find_element(By.CLASS_NAME, "table")
    table_head = table.find_element(By.TAG_NAME, "thead")
    table_body = table.find_element(By.TAG_NAME, "tbody")

    #headers
    headers = table_head.find_elements(By.CLASS_NAME, "table-
head__cell")
    header_text = [header.text for header in headers]
    header_formatted = "\t".join(header_text)
    print(header_formatted)
```

```
    #rows
    rows = table_body.find_elements(By.CLASS_NAME, "table-body")
    i=0
    for row in rows:
        cells = row.find_elements(By.CLASS_NAME, "table-body__cell")
        row_data = [cell.text for cell in cells]
        if(i==1 or i==10):
            row_formatted = "\t ".join(row_data)
        else:
            row_formatted = "\t".join(row_data)
        print(row_formatted)
    print()

def main():
    #Initiating the driver here in Main...
    driver = webdriver.Firefox()
    print("Browser Automation using Selenium")
    print("---------------------------------")
    print("CWC - 2023")
    print("----------")
    Standings(driver)
    time.sleep(1)
    print("_._._._._._._._._")
    print(" Season Stats")
    print("````````````````")
    stats(driver)
    # Closeing the browser
    driver.quit()

if __name__ == '__main__':
    main()
```

➠ ***Output:***







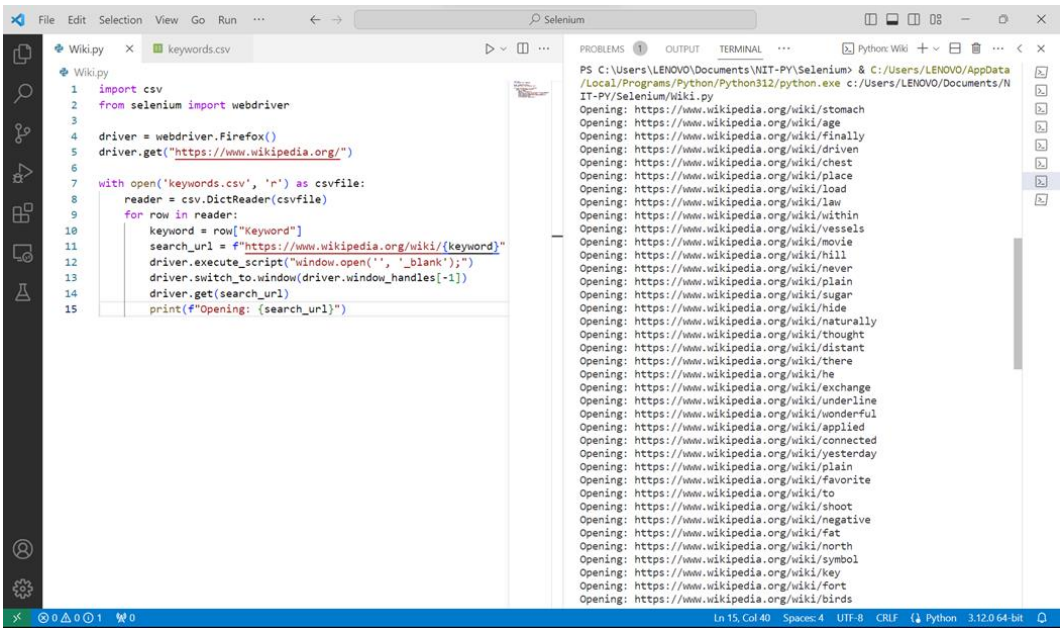
➠ *wik.py*

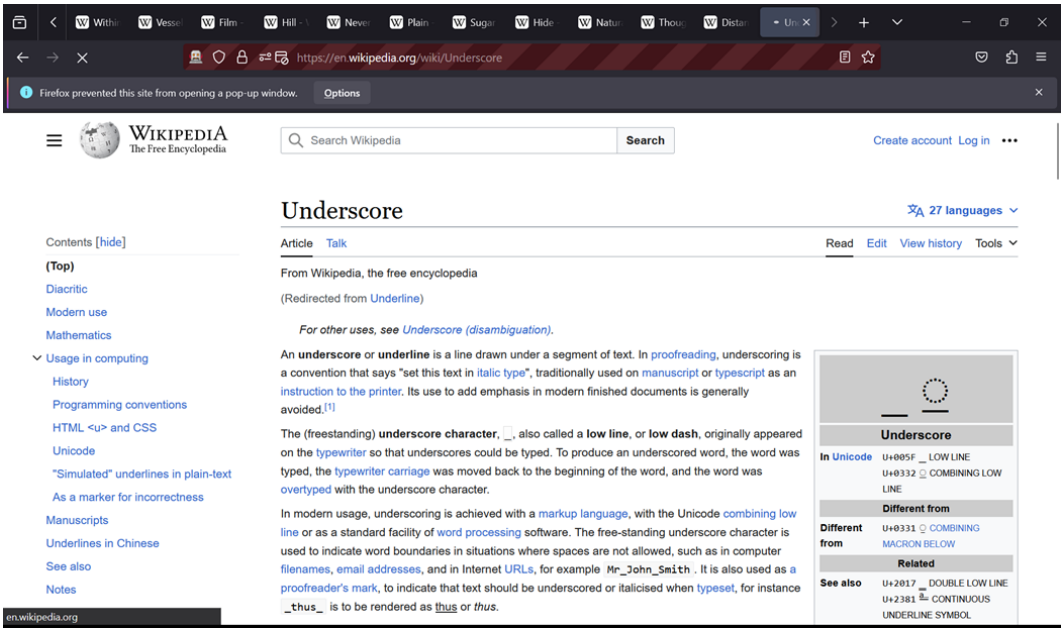
```
import csv
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("https://www.wikipedia.org/")

with open('keywords.csv', 'r') as csvfile:
    reader = csv.DictReader(csvfile)

    for row in reader:
        keyword = row["Keyword"]
        search_url = f"https://www.wikipedia.org/wiki/{keyword}"
        driver.execute_script("window.open('', '_blank');")
        driver.switch_to.window(driver.window_handles[-1])
        driver.get(search_url)
        print(f"Opening: {search_url}")
```

➠ *keywords.csv:*

Keyword	"within"	"thought"	"yesterday"	"key"	"package"
"stomach"	"vessels"	"distant"	"plain"	"fort"	"maybe"
"age"	"movie"	"there"	"favorite"	"birds"	"chest"
"finally"	"hill"	"he"	"to"	"everyone"	"island"
"driven"	"never"	"exchange"	"shoot"	"tight"	"tell"
"chest"	"plain"	"underline"	"negative"	"traffic"	"complex"
"place"	"sugar"	"wonderful"	"fat"	"cry"	
"load"	"hide"	"applied"	"north"	"curve"	
"law"	"naturally"	"connected"	"symbol"	"handsome"	

➠ *Output:*





Cyberattacks – Brute Force

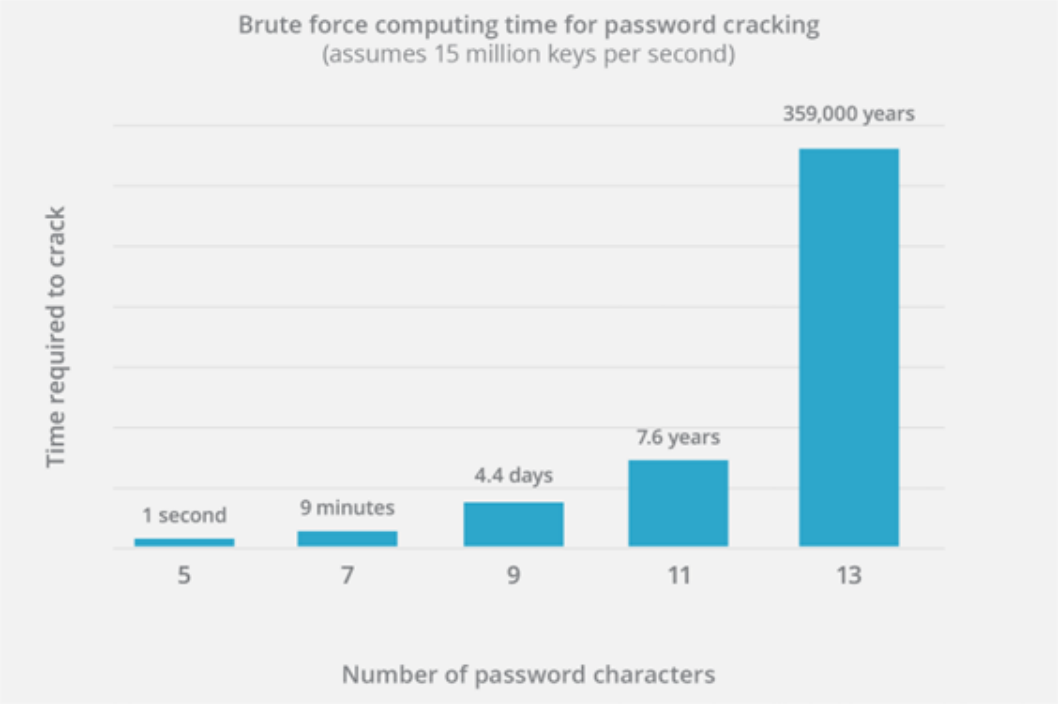
Introduction:

A brute force attack is a type of cyber-attack where an attacker systematically tries all possible combinations of characters to gain unauthorized access to a user account. The primary goal of a brute force attack is to gain access to sensitive information, such as login credentials or personal data. These attacks are often used to compromise websites, online services, or individual computers. There are two main types of brute force attacks:

- Online brute force attacks involve attempting password guesses while connected to the target system, such as trying to log in to a user account.
- Offline brute force attacks, on the other hand, involve obtaining a hashed password from the target system and then attempting to guess the original password offline.

The biggest advantages of brute force attacks are that they are relatively simple to perform and, given enough time and the lack of a mitigation strategy for the target, they always work. Every password-based system and encryption key out there can be cracked using a brute force attack. In fact, the amount of time it takes to brute force into a system is a useful metric for gauging that system's level of security.

On the other hand, brute force attacks are very slow, as they may have to run through every possible combination of characters before achieving their goal. This sluggishness is compounded as the number of characters in the target string increases (a string is just a combination of characters). For example, a four-character password takes significantly longer to brute force than a three-character password, and a five-character password takes significantly longer than a four-character password. Once character count is beyond a certain point, brute forcing a properly randomized password becomes unrealistic.



Impact of Brute Force Attacks:

The consequences of a successful brute force attack can be severe, as attackers may gain unauthorized access to sensitive data or critical systems. Some potential impacts include:

- Account takeover: Attackers could gain control of user accounts, allowing them to access sensitive information or even the account's associated devices.
- System compromise: If an attacker successfully gains access to a server or system, they could potentially compromise the security of the entire network or even other systems connected to it.
- Data breaches: A brute force attack that compromises a system could result in the exposure of sensitive data, potentially leading to further security incidents or legal consequences.
-

Countermeasures:

To protect against brute force attacks, organizations can implement several security measures, such as:

- Account lockout policies: These policies prevent further login attempts after a certain number of failed attempts, thereby slowing down brute force attacks.
- Password complexity requirements: Enforcing strict password complexity rules can make it more difficult for attackers to guess the correct password.
- Rate limiting: Limiting the number of logins attempts a user can make within a specific time frame can also help slow down brute force attacks.
- Multi-factor authentication (MFA): Implementing MFA can significantly increase the security of user accounts, as attackers would need to obtain not only a valid password but also access to a trusted device or application.

AI-Powered Security:

Artificial intelligence can play a crucial role in detecting and preventing brute force attacks. AI-powered solutions can:

- Analyse user behaviour patterns: By monitoring user behaviour, AI can identify suspicious patterns that may indicate a brute force attack, such as an unusually high number of failed logins attempts or a series of geographically dispersed login attempts.
- Identify known attack signatures: AI can automatically detect known attack signatures associated with brute force attacks, enabling faster identification and response.
- Provide adaptive defence: AI can adapt to new attack signatures and methods, enabling organizations to maintain a robust defence against brute force attacks even as they evolve.

Conclusion:

Brute force attacks remain a significant threat to cybersecurity, with attackers continuously finding new ways to compromise user accounts and access sensitive data. However, organizations can take several proactive measures to protect against brute force attacks, including implementing robust security policies, leveraging AI-powered security solutions, and staying informed about emerging threats and trends.

Code Implementation:

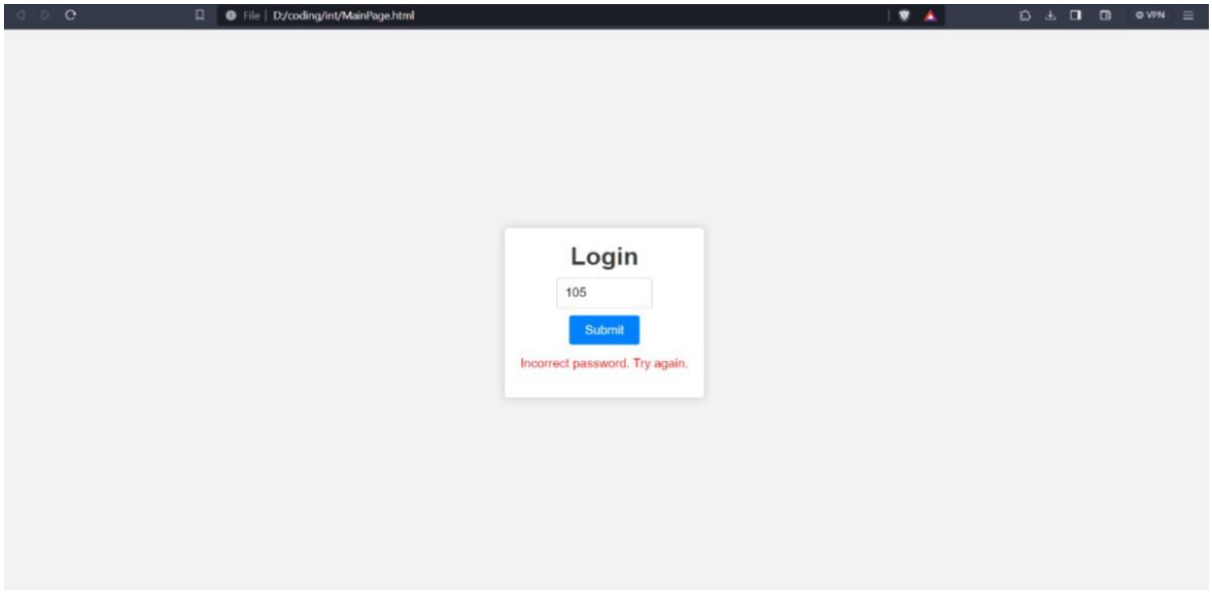
➠ MainPage.html

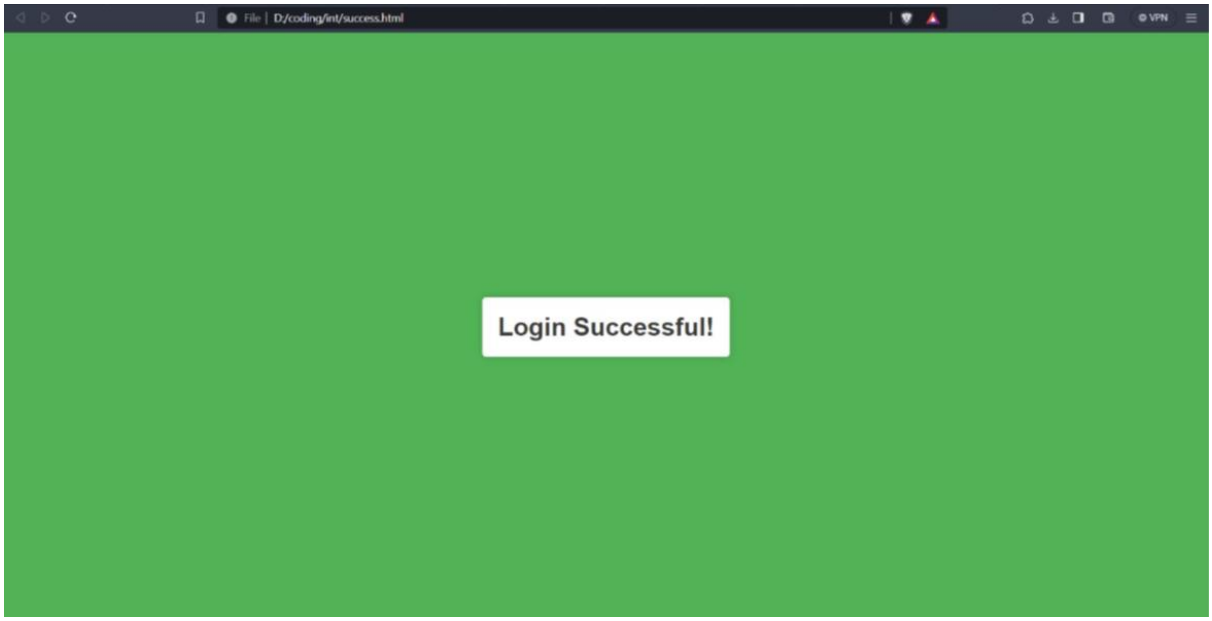
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login Page</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f1f1f1;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
    }
    .login-container {
      background-color: #fff;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.2);
      text-align: center;
    }
    h1 {
      margin: 0;
      color: #333;
    }
    input[type="text"] {
      padding: 10px;
      margin: 10px 0;
      border: 1px solid #ccc;
      border-radius: 3px;
      width: 100px;
      font-size: 16px;
    }
    button {
      background-color: #007BFF;
      color: #fff;
      border: none;
      border-radius: 3px;
      padding: 10px 20px;
      font-size: 16px;
      cursor: pointer;
    }
    button:hover {
      background-color: #0056b3;
    }
  </style>
</head>
<body>
  <div class="login-container">
    <h1>Login</h1>
    <input type="text" id="passwordInput" placeholder="Enter 3-digit password">
    <br>
    <button onclick="login()">Submit</button>
    <p id="message" style="color: red;"></p>
  </div>

  <script>
    function login() {
      var password = document.getElementById("passwordInput").value;
      if (password.length === 3 && !isNaN(password)) {
        if (password === "123") { //Password
          window.location.href = "success.html";
        } else {
          document.getElementById("message").textContent = "Incorrect password. Try again.";
        }
      } else {
        document.getElementById("message").textContent = "Please enter a 3-digit number.";
      }
    }
  </script>
  <script>
    let count = 1;
    let interval = setInterval(() => {
    if (count < 1000) {
      let passwordInput = document.getElementById("passwordInput");
      passwordInput.value = '';
      passwordInput.value = count;
      let loginButton = Array.from(document.querySelectorAll('button')).find(btn => btn.textContent === 'Submit');
      if (loginButton) {
        loginButton.click();
      }
      count++;
    } else {
      clearInterval(interval);
    }
    }, 100);
  </script>
</body>
</html>
```

➠ ***success.html***

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Success</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #4CAF50;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
    }
    .success-message {
      background-color: #fff;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.2);
      text-align: center;
    }
    h1 {
      margin: 0;
      color: #333;
    }
  </style>
</head>
<body>
  <div class="success-message">
    <h1>Login Successful!</h1>
  </div>
</body>
</html>
```

➠ ***Output:***





Cyberattacks – SQL Injection

Introduction:

SQL injections (SQLi) represent one of the most prevalent and concerning attack vectors against web applications. Even as recently as 2019, SQLi accounted for nearly two-thirds of web application attacks, underscoring their continued relevance and threat. What's surprising is that SQLi has been a part of the top 10 Common Vulnerabilities and Exposures (CVE) since 2003, and it has only grown in popularity in recent years. In this report, we will delve into the world of SQL injections, exploring their various forms and, more importantly, how to shield your applications from their potentially devastating impact. Furthermore, we'll emphasize the critical role of automated security testing in your DevOps pipeline to intercept these vulnerabilities before they can wreak havoc in a production environment.

Understanding SQL Injections:

SQL injections are web application attacks in which attackers cleverly "inject" SQL statements into user inputs. This manipulation can grant unauthorized access to application data, whether it's sensitive or publicly accessible. The vulnerability lies in the areas of web applications where user inputs are not thoroughly sanitized, allowing attackers to breach the application's database.

An Illustrative SQL Injection:

Attackers frequently infiltrate web applications through elements such as usernames, user IDs, or various form submissions. Imagine a typical scenario where a user provides their user ID, resulting in a legitimate SQL query like the following:

```
Sql> SELECT * FROM users WHERE id = '42'
```

However, the trouble begins when input sanitization is lacking. An attacker can input a malicious SQL statement, for instance:

```
Sql> '42' OR '1'='1'
```

In the absence of proper sanitization, this input would execute the following SQL query:

```
Sql> SELECT * FROM users WHERE id = '42' OR '1'='1';
```

The condition '1=1' is always true, causing the query to return all data fields for all users. This is a classic example of a SQL injection. Importantly, it's crucial to understand that attackers are not looking to disrupt the application itself but rather to exploit the existing functionality to access unauthorized data. When developing applications, it's vital to anticipate potential vulnerabilities and implement safeguards accordingly.

The Three Main Types of SQL Injections:

There are three primary types of SQL injections, each with its unique characteristics.

1.In-band SQLi:

- In-band SQLi occurs when an attacker initiates the SQL injection from the same location used to collect the output. This type of attack can be categorized further into two subtypes: Error-based SQLi and Union-based SQLi.
- Error-based SQLi: This type of SQL injection capitalizes on error messages produced by the database. Attackers use these error messages to gather information about the database's structure and elements before proceeding to more complex attacks.
- Union-based SQLi: Union-based SQLi employs the UNION operator to append additional data to a single result, often adding data to the visible table in the web application. Executing a Union-based SQLi successfully requires the attacker to possess key information about the database, including the table name, the number of columns in the query, and the data types. Information collected through Error-based SQLi can often be used in Union-based SQLi attacks.

2. Blind SQLi:

- Blind SQLi is a variant of In-band SQLi, with one crucial difference: responses from the web application do not reveal the results of the query or any database errors. Blind SQLi can manifest in two forms: Content-based Blind SQLi and Time-based Blind SQLi.
- Content-based Blind SQLi: In this approach, attackers utilize queries that prompt conditional responses rather than direct data outputs. These SQL queries ask the database true or false questions, allowing attackers to assess the responses and determine whether the application is vulnerable. This process can be quite tedious, which is why attackers sometimes resort to automation.
- Time-based Blind SQLi: Time-based Blind SQLi leverages time-intensive operations in the system. By causing the database to delay its response, attackers can gauge the vulnerability of the system.

3. Out-of-Band SQLi:

- Out-of-Band SQLi differs from In-band SQLi. In this case, attackers initiate the SQL injection from a different channel than the one used for gathering output. This method manipulates the database server to deliver data to an external server controlled by the attacker. It exploits features like Microsoft SQL Server's xp_dirtree command to achieve this objective.
- It's important to note that Out-of-Band SQLi attacks are relatively less common than other types, often contingent on the specific features enabled on the database server.

Preventing SQL Injections – Best Practices:

Mitigating the risks associated with SQL injections requires the implementation of several security measures:

- Sanitize User Inputs: Properly sanitizing and validating all user inputs is of paramount importance. This process should be conducted on the server-side, not solely on the client-side. Regular expressions can be employed to filter and accept only specific characters or patterns.
- Follow the Principle of Least Privilege: Limiting user access according to the principle of least privilege is essential. Grant users' access to only the privileges they truly require, reducing the potential impact of misuse.
- Separate Sensitive and Public Data: Sensitive data should be treated distinctly from public data. Store sensitive data only when absolutely necessary, and ensure it is encrypted. Take extra precautions to safeguard sensitive data and protect your users' privacy.
- Automate Testing for SQL Injection in the Build Pipeline: Integrating automated security testing into your DevOps pipeline is a critical step to identify and rectify vulnerabilities, such as SQL injections, before they can reach the production environment. Several tools, including StackHawk, are available to assist engineering teams in this endeavour.

Conclusion:

SQL injections continue to pose a persistent and evolving threat to web applications. As attackers increasingly exploit this vulnerability, developers and organizations must remain vigilant and take proactive steps to secure their applications. By adhering to best practices, thoroughly sanitizing user inputs, and implementing automated security testing, businesses can fortify their applications and protect their data from potential breaches.

Code Implementation:

➠ *success.html*

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Success</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #4CAF50;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
    }
    .success-message {
      background-color: #fff;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.2);
      text-align: center;
    }
    h1 {
      margin: 0;
      color: #333;
    }
  </style>
</head>
<body>
  <div class="success-message">
    <h1>Login Successful!</h1>
  </div>
</body>
</html>
```

➠ *test.sl3*

```
sqlite> CREATE TABLE password (passkey VARCHAR(5));
sqlite> INSERT INTO password VALUES (4421);
sqlite> SELECT * FROM password;
4421
```

➠ *MainPage.html*

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Login Page</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f1f1f1;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
    }

    .login-container {
      background-color: #fff;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.2);
      text-align: center;
    }

    h1 {
      margin: 0;
      color: #333;
    }

    input[type="text"] {
      padding: 10px;
      margin: 10px 0;
      border: 1px solid #ccc;
      border-radius: 3px;
      width: 100px;
      font-size: 16px;
    }
```

```
        button {
            background-color: #007BFF;
            color: #fff;
            border: none;
            border-radius: 3px;
            padding: 10px 20px;
            font-size: 16px;
            cursor: pointer;
        }

        button:hover {
            background-color: #0056b3;
        }
    </style>
</head>

<body>
    <div class="login-container">
        <h1>Login</h1>
        <input type="text" id="passwordInput" placeholder="Enter password">
        <br>
        <button onclick="login()">Submit</button>
        <p id="message" style="color: red;"></p>
    </div>
    <script>
        let bool1 = 0;
        window.addEventListener('load', function () {
            fetch('/getBool1?query=4421')
                .then(response => response.json())
                .then(data => {
                    bool1 = data.bool1;
                })
                .catch(error => {
                    console.error("Error fetching bool1:", error);
                });
        });

        function login() {
            let password = document.getElementById("passwordInput").value;
            if (!isNaN(password)) {
                if (password === bool1) { //Password
                    window.location.href = `http://localhost:8080/success.html`;
                    console.log("SUCCESS");
                } else {
                    document.getElementById("message").textContent = "Incorrect password. Try again.";
                }
            } else {
                document.getElementById("message").textContent = "Please enter a valid number.";
            }
        }

        let count = 1;
        let interval = setInterval(() => {
            if (count < 10000) {
                let passwordInput = document.getElementById("passwordInput");
                passwordInput.value = '';
                passwordInput.value = count;
                let loginButton = Array.from(document.querySelectorAll('button')).find(btn => btn.textContent === 'Submit');
                if (loginButton) {
                    loginButton.click();
                }
                count++;
            } else {
                clearInterval(interval);
            }
        }, 5);
    </script>
</body>

</html>
```

➡ ***script.js***

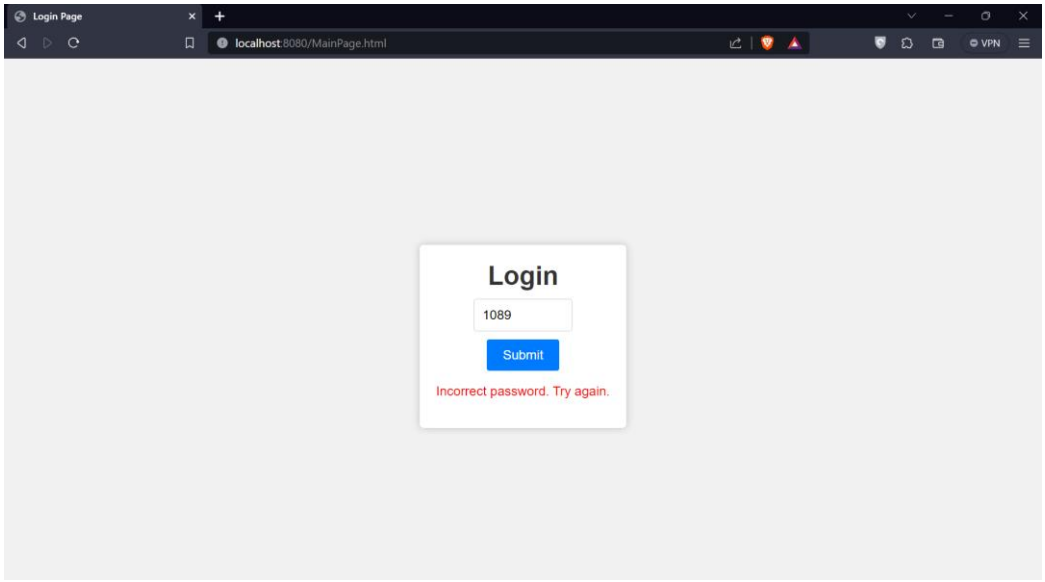
```
const http = require('http');
const fs = require('fs');
const sqlite3 = require('sqlite3');
const url = require('url');
const PORT = 8080;

// Define a function to get the password and call the callback function
function getPass(query, callback) {
    const sqlQuery = `SELECT passkey FROM password WHERE passkey = '${query}'`;

    const db = new sqlite3.Database('./test.sl3', sqlite3.OPEN_READWRITE, (err) => {
        if (err) {
            console.error('Database error:', err.message);
            return callback(err, null);
        }

        db.get(sqlQuery, function (err, row) {
            if (err) {
                console.error('Database error:', err.message);
                db.close();
                callback(err, null);
            } else {
                const bool1 = row ? row.passkey : null;
                db.close();
                callback(null, bool1);
            }
        });
    });
}

// Define a function to serve HTML files
function serveHTML(filename, response) {
    fs.readFile(filename, 'utf8', (err, data) => {
        if (err) {
            response.writeHead(500, { 'Content-Type': 'text/plain' });
            response.end('Internal Server Error');
        } else {
            response.writeHead(200, { 'Content-Type': 'text/html' });
            response.end(data);
        }
    });
}

// Create an HTTP server
http.createServer((request, response) => {
    const parsedUrl = url.parse(request.url, true);

    if (parsedUrl.pathname === '/getBool1') {
        const query = parsedUrl.query.query;

        if (query) {
            getPass(query, (err, bool1) => {
                if (err) {
                    response.writeHead(500, { 'Content-Type': 'text/plain' });
                    response.end('Internal Server Error');
                } else {
                    response.writeHead(200, { 'Content-Type': 'application/json' });
                    response.end(JSON.stringify({ bool1 }));
                }
            });
        } else {
            response.writeHead(400, { 'Content-Type': 'text/plain' });
            response.end('Query parameter "query" is missing in the URL.');
        }
    } else if (parsedUrl.pathname === '/MainPage.html') {
        serveHTML('./MainPage.html', response);
    } else if (parsedUrl.pathname === '/success.html') {
        serveHTML('./success.html', response);
    } else if (parsedUrl.pathname === '/MainPage_Query.html') {
        serveHTML('./MainPage_Query.html', response);
    } else {
        response.writeHead(404, { 'Content-Type': 'text/plain' });
        response.end('Not Found');
    }
}).listen(PORT);
```

➡ ***Brief Overview:***

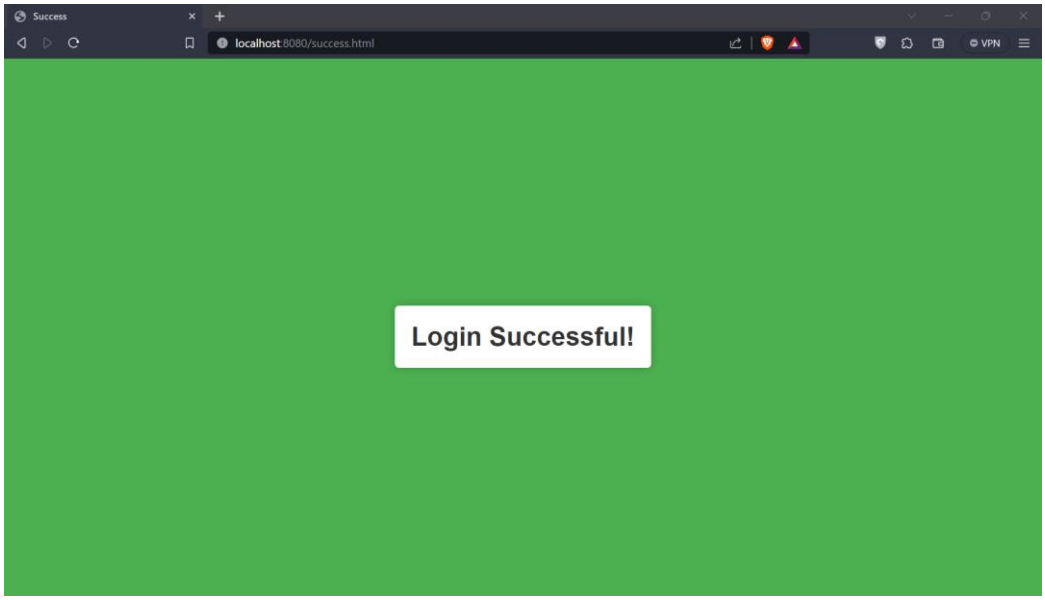
- `success.html` is a simple HTML document designed to display a success message when a user successfully logs in. It features a centered message with a green background, clear "Login Successful!" text, and a clean, minimalistic design. This page is intended to provide a positive user experience after a successful login attempt in the web application.
- `MainPage.html` features a login form centered on the screen. Users enter a password and click "Submit". JavaScript fetches a "bool1" value and checks if the entered password matches it. There's also a countdown loop for testing and automation of a simulated brute force attack. The design includes a white background, rounded corners, and a shadow effect.
- `script.js' is a Node.js script that creates an HTTP server to serve HTML files and handle requests for a specific endpoint that retrieves a password (bool1) from an SQLite database. Here's a brief overview:

  Dependencies: The script requires Node.js modules, including 'http', 'fs', 'sqlite3' and 'url'.
  Port Configuration: The server is set to listen on port 8080.
  getPass Function: This function queries an SQLite database for a password (bool1) based on a provided parameter. It uses a SQL query to retrieve the password and passes it to a callback function.
  serveHTML Function: This function reads and serves HTML files in response to client requests. It reads the specified file and sends it as a response.
  HTTP Server: An HTTP server is created using http.createServer. It listens for incoming requests and routes them to different endpoints based on the URL path.
  Request Handling:
  - If the path is '/getBool1', it extracts the 'query' parameter and calls the getPass function to retrieve the password. The result is sent as a JSON response.
  - If the path corresponds to specific HTML files ('/MainPage.html', '/success.html'), it uses the serveHTML function to serve the corresponding HTML file.
  - If the path doesn't match any known endpoints, it returns a 'Not Found' response with a 404 status.

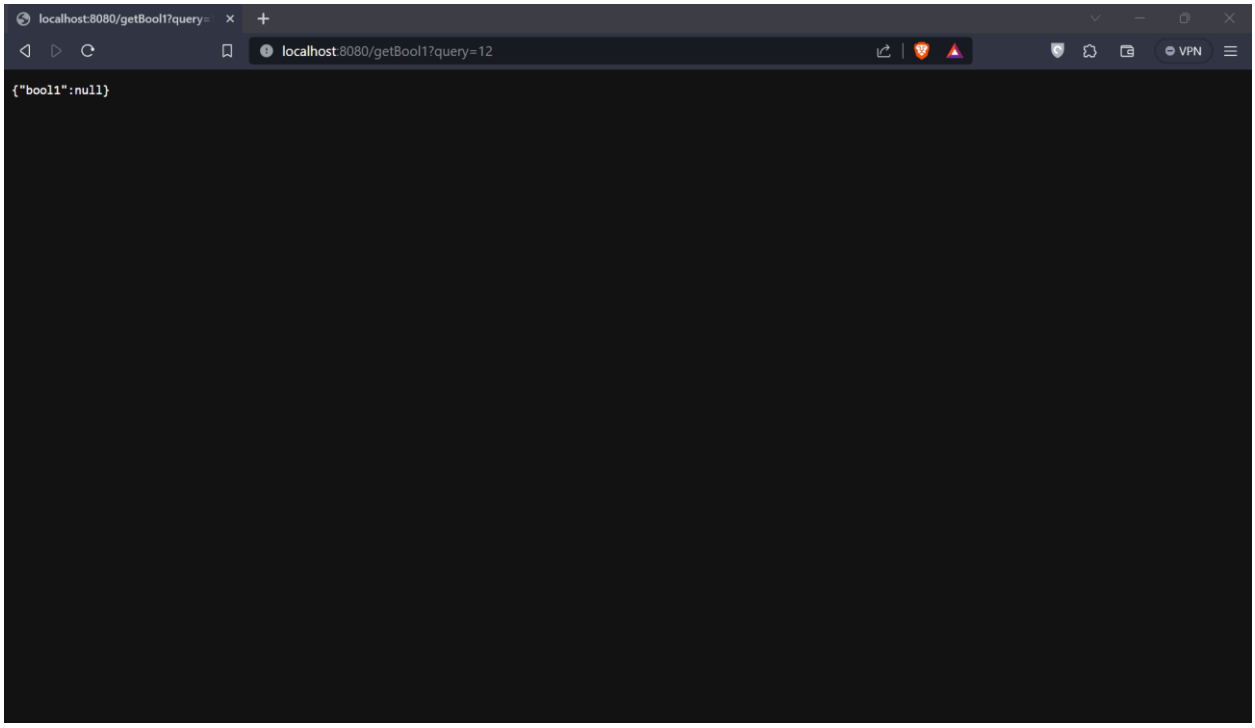
➡ ***Output:***



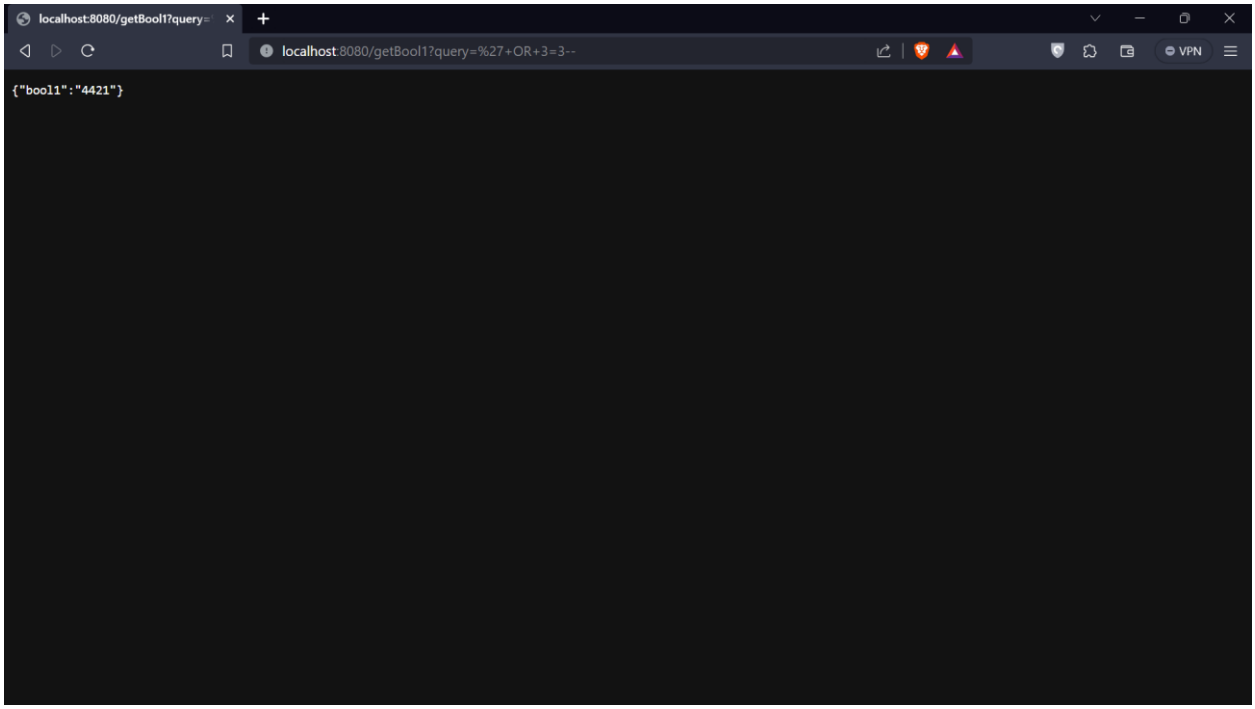
The image depicts `MainPage.html` being served by `script.js` in PORT 8080, when an incorrect password is tried.



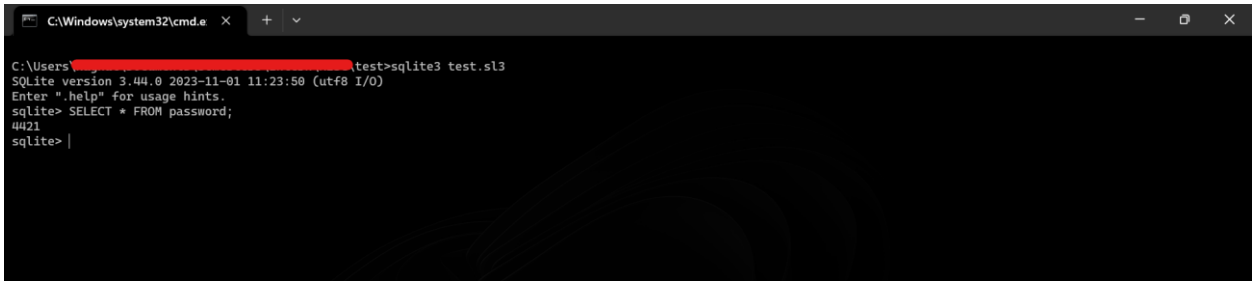
The image depicts `success.html` being served by `script.js` in PORT 8080, after a successful login attempt.



The image depicts `/getBool1` pathway output when queried with an incorrect password.



The image depicts `/getBool1` pathway during an SQLi attempt to retrieve the password from the database. The attempt makes use of the '3=3' condition being always true to negate the 'WHERE' clause using the 'OR' statement. '- -' is used to comment out the remaining portion of the query.



The image depicts the table of passwords stored in the sqlite database `test.sl3`.

References

- ✓ https://www.selenium.dev/
- ✓ https://www.webomates.com/blog/software-testing/selenium-testing/
- ✓ https://www.itview.in/blog/selenium-automation-testing-overview-and-its-benefits/
- ✓ https://www.browserstack.com/selenium
- ✓ https://www.simplilearn.com/tutorials/selenium-tutorial/what-is-selenium

- ✓ https://www.techtarget.com/searchnetworking/definition/software-defined-networking-SDN
- ✓ https://www.researchgate.net/publication/267339360_SoftwareDefined_Networking_Challenges_and_research_opportunities_for_Future_Internet
- ✓ https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4230813
- ✓ https://www.vmware.com/topics/glossary/content/software-definednetworking.html#:~:text=Software%2DDefined%20Networking%20(SDN),direct%20traffic%20on%20a%20network
- ✓ https://www.geeksforgeeks.org/software-defined-networking/
- ✓ https://www.cisco.com/c/en/us/solutions/software-defined-networking/overview.html

- ✓ https://www.spiceworks.com/it-security/cyber-risk-management/articles/what-is-brute-force-attack/
- ✓ https://ijrpr.com/uploads/V3ISSUE11/IJRPR7767.pdf
- ✓ https://www.strongdm.com/blog/brute-force-attack
- ✓ https://owasp.org/www-community/attacks/Credential_stuffing

- ✓ https://www.w3schools.com/
- ✓ https://medium.com/@codesprintpro/getting-started-sqlite3-with-nodejs-8ef387ad31c4
- ✓ https://medium.com/@adnanrahic/hello-world-app-with-node-js-and-express-c1eb7cfa8a30
- ✓ https://www.freecodecamp.org/news/module-exports-how-to-export-in-node-js-and-javascript/