Input Validation Testing

Aim :

To familiarize sql injection testing for input validation using burpsuite

Description:

SQL injection testing checks if it is possible to inject data into the application so that it executes a user-controlled SQL query in the database. Testers find a SQL injection vulnerability if the application uses user input to create SQL queries without proper input validation. A successful exploitation of this class of vulnerability allows an unauthorized user to access or manipulate data in the database.

An SQL injection attack consists of insertion or "injection" of either a partial or complete SQL query via the data input or transmitted from the client (browser) to the web application. A successful SQL injection attack can read sensitive data from the database, modify database data (insert/update/delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file existing on the DBMS file system or write files into the file system, and, in some cases, issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

SQL Injection attacks can be divided into the following three classes:

- ✓ Inband: data is extracted using the same channel that is used to inject the SQL code. This is the most straightforward kind of attack, in which the retrieved data is presented directly in the application web page.
- ✓ Out-of-band: data is retrieved using a different channel (e.g., an email with the results of the query is generated and sent to the tester).
- ✓ Inferential or Blind: there is no actual transfer of data, but the tester is able to reconstruct the information by sending particular requests and observing the resulting behavior of the DB Server.

A successful SQL Injection attack requires the attacker to craft a syntactically correct SQL Query. If the application returns an error message generated by an incorrect query, then it may be easier for an attacker to reconstruct the logic of the original query and, therefore, understand how to perform the injection correctly. However, if the application hides the error details, then the tester must be able to reverse engineer the logic of the original query.

About the techniques to exploit SQL injection flaws there are five commons techniques. Also those techniques sometimes can be used in a combined way (e.g. union operator and out-of-band):

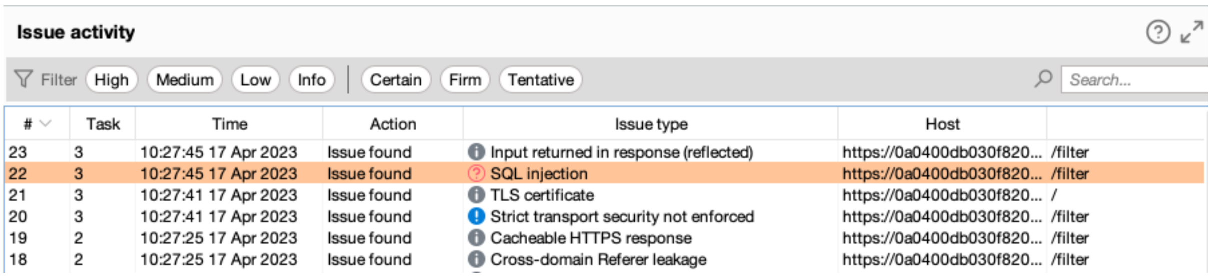
- ✓ Union Operator: can be used when the SQL injection flaw happens in a SELECT statement, making it possible to combine two queries into a single result or result set.
- ✓ Boolean: use Boolean condition(s) to verify whether certain conditions are true or false.
- ✓ Error based: this technique forces the database to generate an error, giving the attacker or tester information upon which to refine their injection.
- ✓ Out-of-band: technique used to retrieve data using a different channel (e.g., make a HTTP connection to send the results to a web server).
- ✓ Time delay: use database commands (e.g. sleep) to delay answers in conditional queries. It is useful when attacker doesn't have some kind of answer (result, output, or error) from the application.

Procedure:

Scanning for SQL injection vulnerabilities

If you're using Burp Suite Professional, you can use Burp Scanner to test for SQL injection vulnerabilities:

1. Identify a request that you want to investigate.
2. In **Proxy > HTTP history**, right-click the request and select **Do active scan**. Burp Scanner audits the application.
3. Review the **Issue activity** panel on the **Dashboard** to identify any SQL injection issues that Burp Scanner flags.

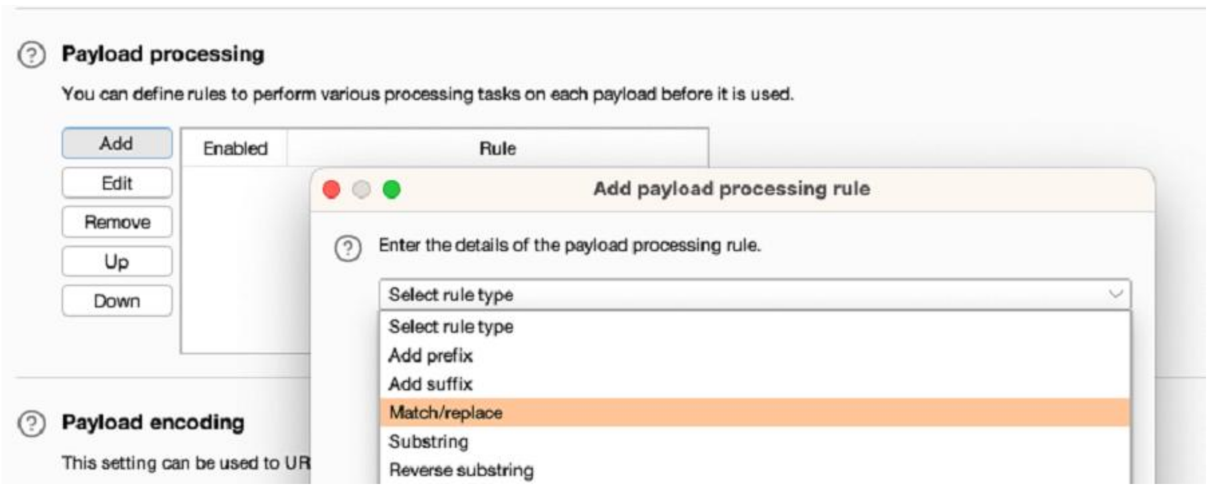


#	Task	Time	Action	Issue type	Host	
23	3	10:27:45 17 Apr 2023	Issue found	Input returned in response (reflected)	https://0a0400db030f820...	/filter
22	3	10:27:45 17 Apr 2023	Issue found	SQL injection	https://0a0400db030f820...	/filter
21	3	10:27:41 17 Apr 2023	Issue found	TLS certificate	https://0a0400db030f820...	/
20	3	10:27:41 17 Apr 2023	Issue found	Strict transport security not enforced	https://0a0400db030f820...	/filter
19	2	10:27:25 17 Apr 2023	Issue found	Cacheable HTTPS response	https://0a0400db030f820...	/filter
18	2	10:27:25 17 Apr 2023	Issue found	Cross-domain Referer leakage	https://0a0400db030f820...	/filter

Manually fuzzing for SQL injection vulnerabilities

You can alternatively use Burp Intruder to test for SQL injection vulnerabilities. This process also enables you to closely investigate any issues that Burp Scanner has identified:

1. Identify a request that you want to investigate.
2. In the request, highlight the parameter that you want to test and select **Send to Intruder**.
3. Go to the **Intruder > Positions** tab. Notice that the parameter has been automatically marked as a payload position.
4. Go to the **Payloads** tab. Under **Payload settings [Simple list]** add a list of SQL fuzz strings.
    1. If you're using Burp Suite Professional, open the **Add from list** drop-down menu and select the built-in **Fuzzing - SQL wordlist**.
    2. If you're using Burp Suite Community Edition, manually add a list.
5. Under **Payload processing**, click **Add**. Configure payload processing rules to replace any list placeholders with an appropriate value. You need to do this if you're using the built-in wordlist:
    1. To replace the {base} placeholder, select **Replace placeholder with base value**.
    2. To replace other placeholders, select **Match/Replace**, then specify the placeholder and replacement. For example, replace {domain} with the domain name of the site you're testing.


Payload processing

You can define rules to perform various processing tasks on each payload before it is used.

Add payload processing rule

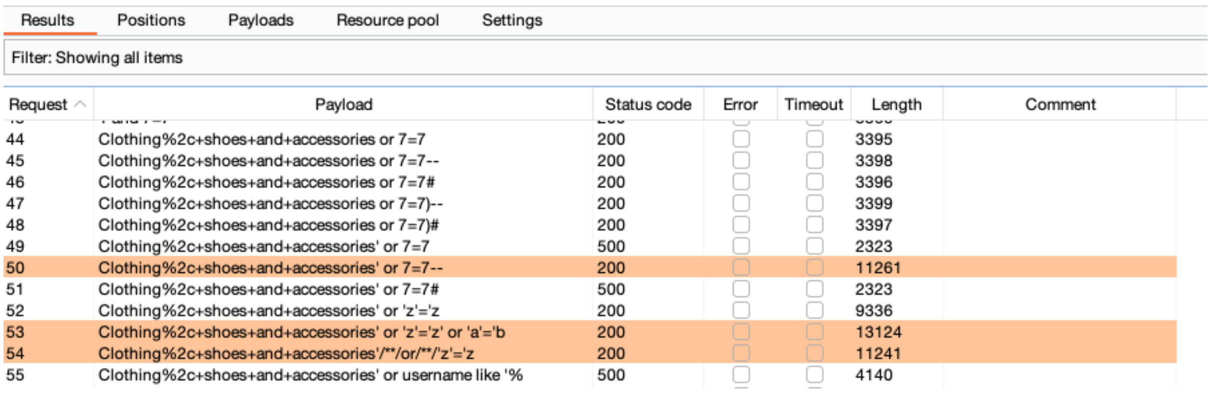
Enter the details of the payload processing rule.

Payload encoding

This setting can be used to UR

6. Click **Start attack**. The attack starts running in a new dialog. Intruder sends a request for each SQL fuzz string on the list.
7. When the attack is finished, study the responses to look for any noteworthy behavior. For example, look for:
    - Responses that include additional data as a result of the query.
    - Responses that include other differences due to the query, such as a "welcome back" message or error message.
    - Responses that had a time delay due to the query.

    If you're using the lab, look for responses with a longer length. These may include additional products.



Request	Payload	Status code	Error	Timeout	Length	Comment
44	Clothing%2c+shoes+and+accessories or 7=7	200	☐	☐	3395	
45	Clothing%2c+shoes+and+accessories or 7=7--	200	☐	☐	3398	
46	Clothing%2c+shoes+and+accessories or 7=7#	200	☐	☐	3396	
47	Clothing%2c+shoes+and+accessories or 7=7)--	200	☐	☐	3399	
48	Clothing%2c+shoes+and+accessories or 7=7)#	200	☐	☐	3397	
49	Clothing%2c+shoes+and+accessories' or 7=7	500	☐	☐	2323	
50	Clothing%2c+shoes+and+accessories' or 7=7--	200	☐	☐	11261	
51	Clothing%2c+shoes+and+accessories' or 7=7#	500	☐	☐	2323	
52	Clothing%2c+shoes+and+accessories' or 'z'='z	200	☐	☐	9336	
53	Clothing%2c+shoes+and+accessories' or 'z'='z' or 'a'='b	200	☐	☐	13124	
54	Clothing%2c+shoes+and+accessories'/**/or/**/'z'='z	200	☐	☐	11241	
55	Clothing%2c+shoes+and+accessories' or username like '%	500	☐	☐	4140	