Finding SQL Injection vulnerabilities using polyglot payloads

Introduction

Looking for SQL Injection vulnerabilities in web applications can be a complicated task. Web applications are often implemented with complicated filters or web application firewalls which can block the use of basic queries such as ‘OR 1=1’ or other well used payloads. This can often trick a consultant into thinking that there is no SQL Injection vulnerability present in the tested parameter. This article will look at the use of a SQL Injection polyglot payload to identify SQL Injection vulnerability in web application filters.

Commented [kl1]: This can often introduce additional complexity…

Polyglot payloads are payloads that can be used in more than one context and it is still treated as valid data. This means that a user can use one payload to test for a vulnerability in different areas of the application with efficiency.

Commented [kl2]: An example or a reference are good to have. Someone not very familiar with the subject, but still technical, would like to know a bit more.

Technical Details

Since SQL Injection can occur in any statement that takes invalidated input; detection can be trivial. Furthermore, successful detection will depend on various factors such as escaping quotes, breaking out of nested statements and more.

To test the effectiveness of this technique, take a look at the following payload

```
IF(SUBSTR(@@version,1,1)<5,BENCHMARK(2000000,SHA1(0xDE7EC71F1)),SLEEP(5))/*'XOR(IF(SU
BSTR(@@version,1,1)<5,BENCHMARK(2000000,SHA1(0xDE7EC71F1)),SLEEP(5)))OR'|"XOR(IF(SUB
STR(@@version,1,1)<5,BENCHMARK(2000000,SHA1(0xDE7EC71F1)),SLEEP(5)))OR"*/
```

This payload was created by Mathias Karlsson and Fredrik Nordberg Almroth of Detectify. This SQL Injection query will work in many cases where a MySQL Injection vulnerability is present on the target. This is done by combing different queries that will work in different context to execute as one.

Commented [kl3]: Reference.

Commented [kl4]: For example the query above would likely remain undetected if the backend database would be running MySQL.

The Substring function is used to extract the version of the SQL server. If this condition is true, then the payload executes the benchmark and sleep function to wait for a specified time. Moreover, the payload uses binary functions (OR and XOR) to concatenate the strings without breaking the syntax. This will adapt the payload to use any quotation that is necessary in order to execute correctly. Furthermore, different queries from different context are combined together with the OR statement. More information regarding this research can be found at: labs.detectify.com/2013/05/29/the-ultimate-sql-injection-payload.

To test the effectiveness of this payload, the web for pentester SQL Injection challenges provided by Pentesterlab will be used as a testbed. Furthermore, the mentioned polyglot will be compared with the Burp Suite’s Active Scanner to see this technique’s strength and weaknesses. Lastly, the following caveats will also be followed as part of the test.

- No additional data will be or queries will be added with the Polyglot payload
- Active scan was reconfigured to scan only inserted points and test for SQL Injection vulnerabilities only
- The polyglot payload will not be modified in any way.

The results are presented in the below table.

Testbed	Context	Filter Used	SQLi payload	Burp Suite Scanner
Example 1	String	None	Success	Success
Example 2	String	No Spaces	Success	Success
Example 3	String	No Spaces, tabulations	Success	Success
Example 4	Integer	mysql_real_escape_string	Success	Success
Example 5	Integer	Regex to remove single/double quotes	Fail	Success
Example 6	Integer	Regex that ensures parameter ends with an digit	Fail	Fail
Example 7	Integer	Regex using multiline modifier	Fail	Fail
Example 8	String (Order by clause)	No Single quote, double quote	Fail	Fail
Example 9	String (Order by clause)	Regex to block back tick	Success	Success

According to the findings presented in the table, the Burp Suite Active scanner was able to detect more SQL injection vulnerabilities than the polyglot payload. But, it should be noted that the Burp scanner used multiple payloads to test for SQL Injection vulnerabilities within the given parameters. Furthermore, Burp Suite had to send a minimum of 40 - 60 requests to successfully identify a SQL injection vulnerability within the given parameter.

It should be noted than on some examples, it is necessary to balance out the initial query with additional quotes or Boolean operators before applying the polyglot payload. This was not done due to the nature of the test.

Conclusion

To conclude, Polyglot can be very useful when testing an application for SQL injection vulnerabilities while minimizing requests and additional payloads. Moreover, the mentioned polyglot payload can be make it easier for a consultant to check for SQL injection vulnerabilities with little understanding of the backend query. But, Polyglot payloads should not be considered as a one hit silver bullet for finding SQL Injection vulnerabilities.

References

https://labs.detectify.com/2013/05/29/the-ultimate-sql-injection-payload/