**SQL Injection**

**SQL injection-Web application security vulnerability-Attacker is able to submit the SQL command that is executed by the web application to expose the database.**

**SQLi occur when the web application utilizes the user supplied data without proper validation or encoding as part of command or query.**

**Types of SQL Injection attack**

1. **First Order Attack**
2. **Second Order Attack**
3. **Blind SQLi**
4. **Time Based SQLi**
5. **Union Based SQLi**

**First Order Attack**

**In this attacker can simply inject a malicious string and cause the modified code to be executed immediately.**

**Second Order Attack**

**In this attacker inject into persistent storage which is considered as a trusted source. An attack subsequently executed by another activity.**

**Blind SQLi**

**In this attack that asks the database true or false questions and determines the answer based on the applications response.**

**This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.**

**Time Based SQLi**

**Used to achieve very basic test like determining if a vulnerability is present.**

**Injects a SQL segment which contains specific DBMS function or heavy query that generates a time delay.**

**Union Based SQLi**

**UNION-based attacks allows to easily extract information from the database.**

**UNION operator can only be used if both queries have the exact same structure.**

**Attacker must craft a SELECT statement similar to the original query.**

**To do this, a valid table name must be known but it is also necessary to determine the number of columns in the first query and their data type.**

**Prevention of SQL Injection**

* **Use of Prepared Statements (Parameterized Queries)**
* **Use of Stored Procedures**
* **Escaping all User Supplied Input**
* **White List Input Validation**

**Tornado Web application**

**Here in my tornado vulnerable web application Login page is vulnerable for sql injection.**

**localhost:7777/login**

**Here we can by pass the authentication by giving the by making the user name and password field true like this…’or’1’=’1**

**This authentication bypass is prevented in the tornado secured web application by using prepared statement.**

**Check the link.**

**http://localhost:8888/login**

**After login we can see link in tornado vulnerable we application for implementing the SQL injection.**

[**http://localhost:7777/SQLiload**](http://localhost:7777/SQLiload)

**By clicking the SQL injection example link it will redirect to other page. Where we can give a valid userid. If we gave an true statement along with the userid it will display all userid and username.**

[**http://localhost:7777/userdetails?uid=1002’or’1’=’1**](http://localhost:7777/userdetails?uid=1002'or'1'='1)

**Also in the same above link by attacker can steal the user credentials. By giving the union statement along with uid.**

**http://localhost:7777/userdetails?uid=1002' union select username,password from user where userid='1002**

**In Tornado secured web application the above SQL injection is prevented and authentication bypass also prevented**

[**http://localhost:8888/login**](http://localhost:8888/login)

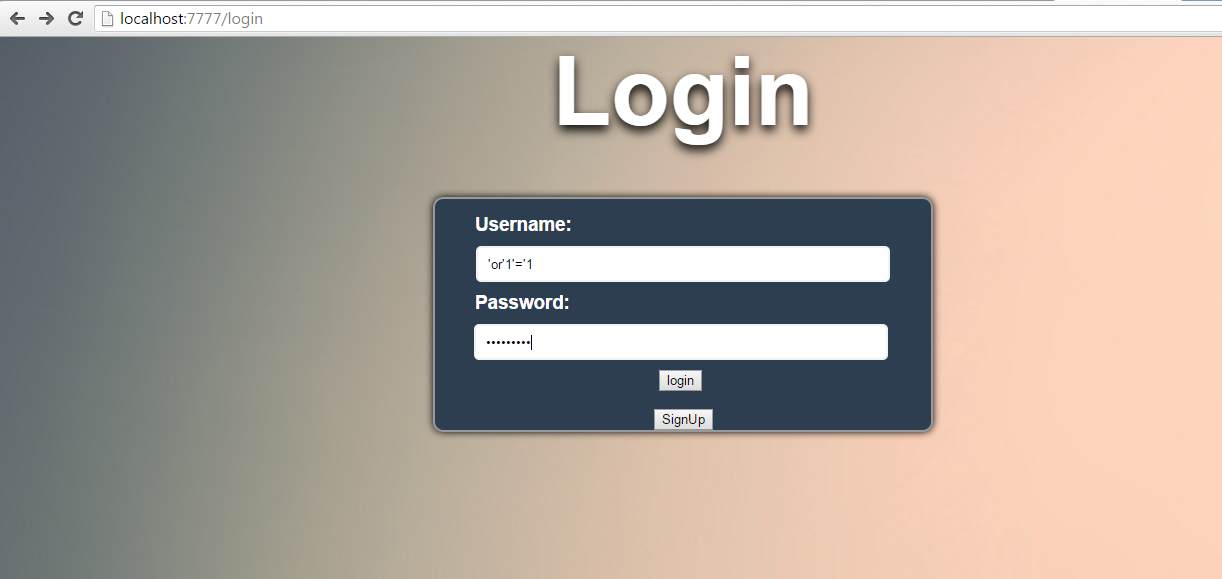
**Time based SQL Injection**

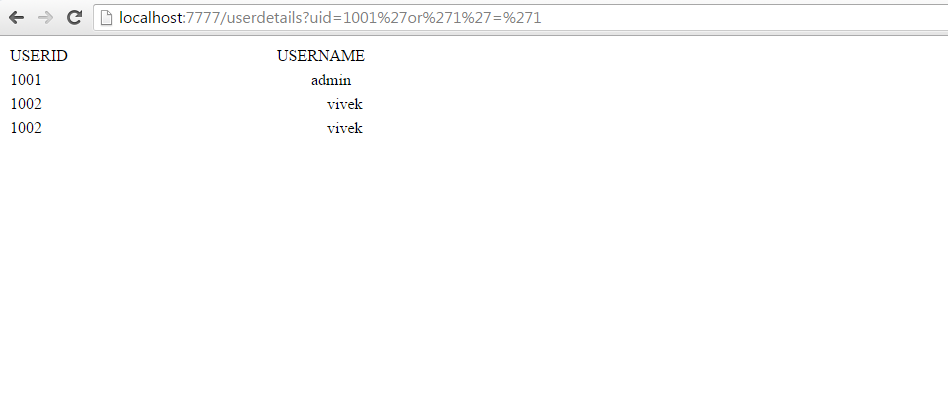
**In Tornado vulnerable web application Time based SQLi is present in the link.**

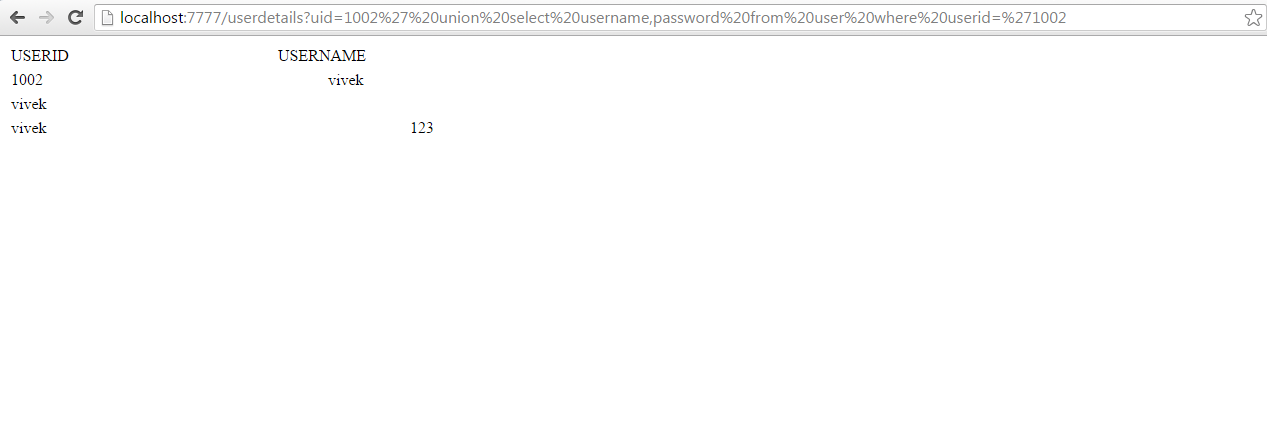
**http://localhost:7777/userdetails?uid=1001%27%20and%20sleep(10)%20or%271**

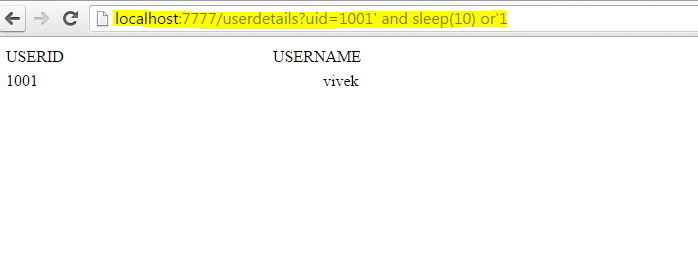
**In Tornado secured web application Time based SQLi is prevented in the link**

**http://localhost:8888/userdetails?uid=1001%27%20and%20sleep(10)%20or%271**

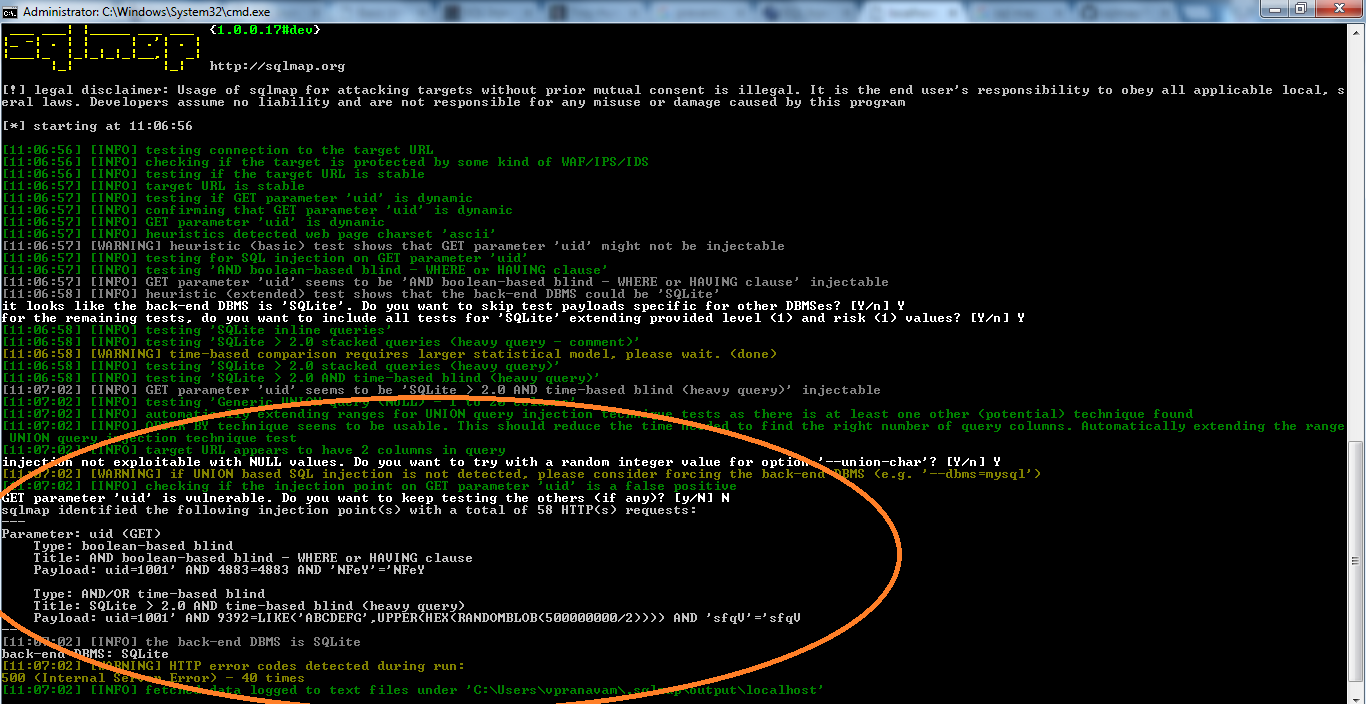
**Tornado vulnerable we application**

****

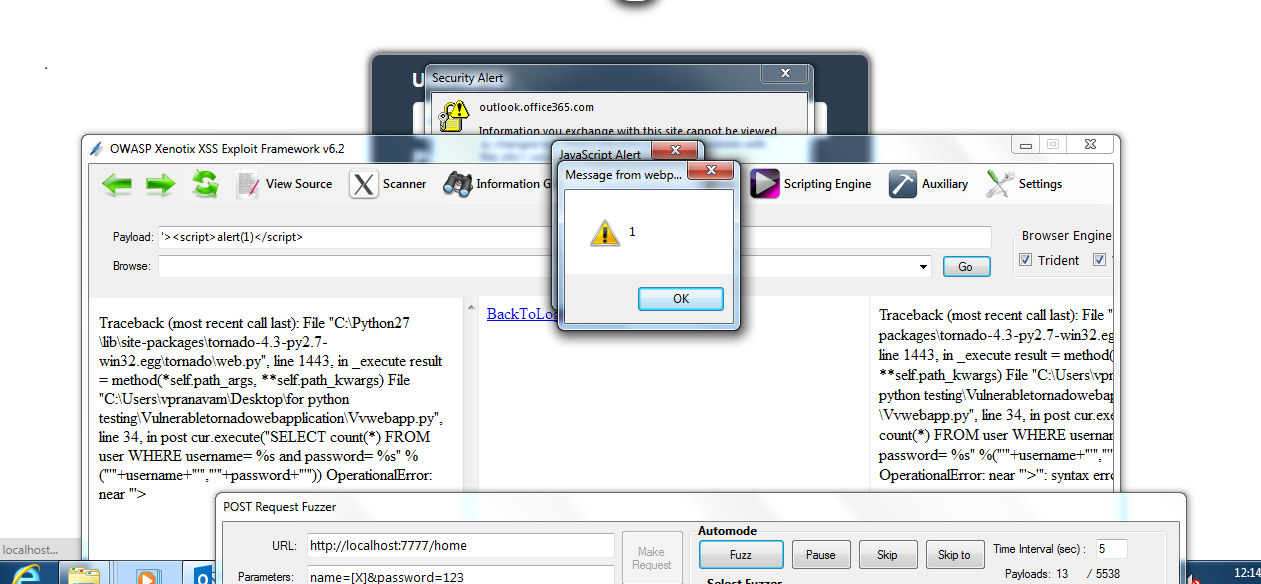
****

****

**Tested using SQLMap**

****

**XSS in vulnerable app - screen shot**

****