***University of Missouri-Kansas City***

***Subject- Information Security and Assurance***

***PROJECT (PHASE 1, ATTACK)***

***Submitted to:***

***Dr. Mahesh Maddumala (Information Security and Assurance Professor)***

***Submitted by:***

***MIHIR PITALE***

***PLAMOND COLAS0***

***MERWAN ABDELMAJED***

Description- Sql injection is a mixture of a code which is injected and invokes the security and vulnerability which is happening in the database system and also exploits and finds hidden information and the capability to change the data, which otherwise would only be changed by admin.

* It finds the specific url’s for missing SQL statements and attacks them and penetrate through them for violating through commands and has the capability of changing the admin database through root login.
* Information disclosure, Compromised Data integrity, and remote\_Cmdshell are most common losses faced due to SQL injection.
* Attacker can thereby easily skip the authentication process.
* Mostly the sites which run with prefix http//: are checked to be seen if vulnerable.
* Attacker searches for vulnerable SQL queries url’s.
* This attack techniques allows malicious users to attack the system using various illogically
* incorrect SQL queries. On the front-end user enters inputs and the resulting web page pops up. In
* the same way malicious users give input which looks invalid or inappropriate bur are database
* wise correct to retrieve more data from database or to bypass authentication
* We have used Tautology sql injection for this project
* Tautology: It injects code in the conditional statements such that it always evaluates to true ‘ or ‘1’=’1
* There are various types of SQL injections

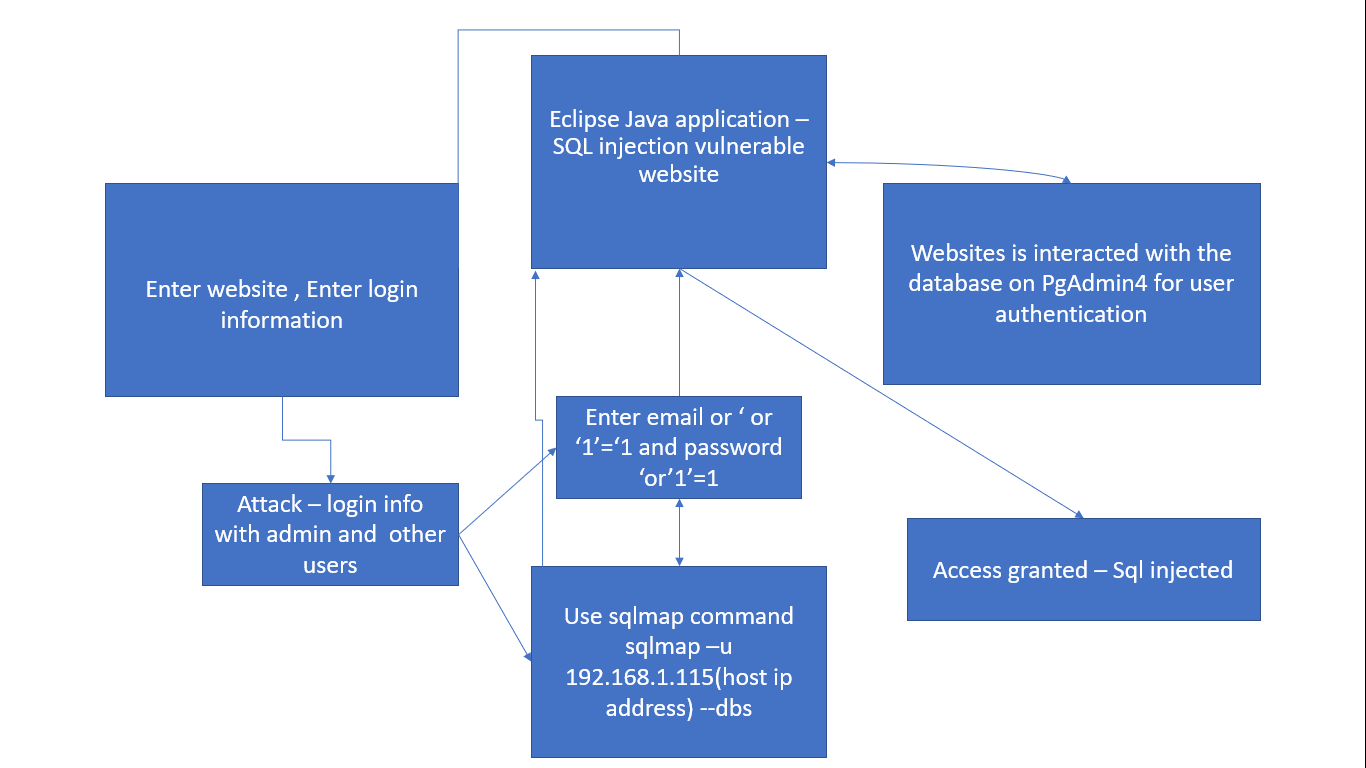
1)Out band

2) Inferential based

3) IN-band

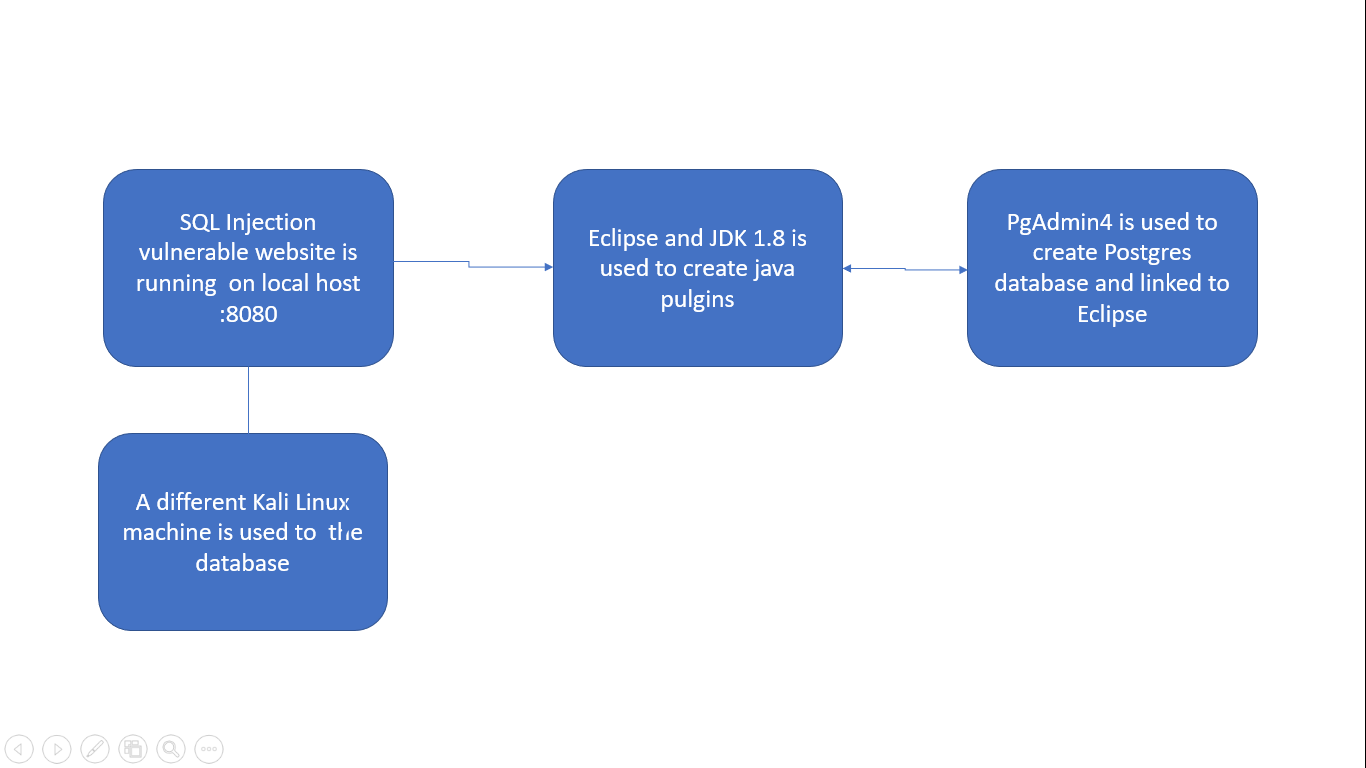
* In-band has more inter-types mainly error-based and Union based
* Inferential has more inter-types mainly, Boolean based and time-based.
* In our SQL injection attack, we have used Boolean based SQL injection for our reference.
* We first created our own website and made it prone to SQL attacks by setting the permissions required.
* Secondly, we created our own database and linked that database to the website named SQLINJECTION VULNERABLE WEBSITE
* Thirdly, we created a source code where only admin has the permission to access and see the emails and passwords of other users, but is kept vulnerable for SQL attacks at localhost:8080/users#
* The database of admin has same password as of that of website thereby granting attacker upper-hand to access the database and modify the database as required.

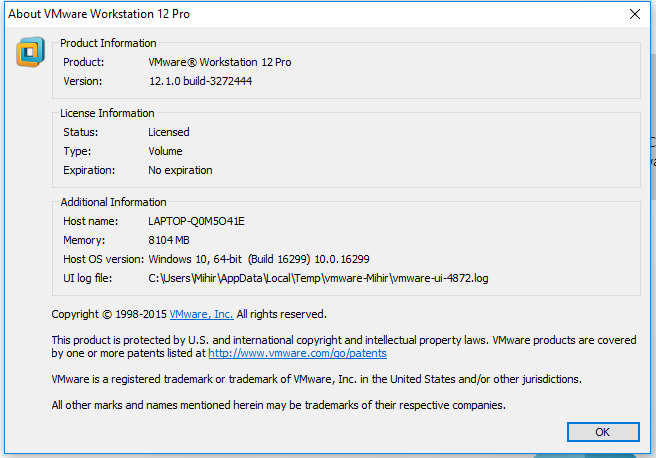
***ATTACK FLOW DIAGRAM***



Protocol information – GET and POST parameters, http protocol, ICMP protocol

***PROJECT FLOW SETUP DIAGRAM***



Virtual Machine Information- 

Source code (Plain text) please refer the attached files to run on eclipse with java jdk v 1.8 and run as maven build.

1. WELCOME CONTROLLER (java file)

package com.umkc.edu.isa;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

@Controller

public class WelcomeController {

// inject via application.properties

@Value("${welcome.message:test}")

private String message = "Hello World";

@RequestMapping("/")

public String welcome(Map<String, Object> model) {

model.put("message", this.message);

return "welcome";

}

@RequestMapping("/login")

public String login(Model model,@RequestParam(value="userName")String userName,@RequestParam("password")String password) throws SQLException {

DBConnection.stablishConnection("jdbc:postgresql://localhost:5432/sql", "postgres", "password");

String query = "Select \* from users where user\_name='" + userName + "' and password='" + password + "'";

ResultSet result = DBConnection.executeQuery(query);

while(result.next()){

model.addAttribute("user",result.getString(2));

return "home";

}

return "error";

}

@RequestMapping("/logout")

public String logout(Map<String, Object> model) {

//return to the welcome page

return "welcome";

}

@RequestMapping("/users")

public String users(final Model model) throws SQLException {

String query = "Select \* from users";

ResultSet result = DBConnection.executeQuery(query);

List<UserDTO> users = new ArrayList<>();

UserDTO user;

while(result.next()){

user = new UserDTO();

user.setId(result.getString(1));

user.setUserName(result.getString(2));

user.setPassword(result.getString(3));

users.add(user);

}

model.addAttribute("users",users);

return "home";

}

}

1. SPRINGBOOTWEBAPPLICATION.JAVA

package com.umkc.edu.isa;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringBootWebApplication {

public static void main(String[] args) throws Exception {

SpringApplication.run(SpringBootWebApplication.class, args);

}

}

1. DBC Connections

package com.umkc.edu.isa;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class DBConnection {

public static Connection con ;

public static Statement statement;

public static Connection stablishConnection(String databaseURL, String userName, String password) throws SQLException{

con = DriverManager.getConnection(databaseURL,userName,password);

return con;

}

private static Statement getStatement() throws SQLException{

statement = con.createStatement();

return statement;

}

public static ResultSet executeQuery(String sql) throws SQLException{

return getStatement().executeQuery(sql);

}

}

1. USERDTO.JAVA
2. **package** com.umkc.edu.isa;
3. **public** **class** UserDTO {
4. String id;
5. String userName;
6. String password;
7. **public** String getUserName() {
8. **return** userName;
9. }
11. **public** String getId() {
12. **return** id;
13. }
14. **public** **void** setId(String id) {
15. **this**.id = id;
16. }
17. **public** **void** setUserName(String userName) {
18. **this**.userName = userName;
19. }
20. **public** String getPassword() {
21. **return** password;
22. }
23. **public** **void** setPassword(String password) {
24. **this**.password = password;
25. }

28. }

HTML CODE

/\*

\* Specific styles of signin component

\*/

/\*

\* General styles

\*/

**body,** **html** {

height: *100%*;

background-repeat: *no-repeat*;

background-image: *linear-gradient(rgb(104,* *145,* *162),* *rgb(12,* *97,* *33))*;

}

*.card-container.card* {

max-width: *350px*;

padding: *40px* *40px*;

}

*.btn* {

font-weight: *700*;

height: *36px*;

-moz-user-select: *none*;

-webkit-user-select: *none*;

user-select: *none*;

cursor: *default*;

}

/\*

\* Card component

\*/

*.card* {

background-color: *#F7F7F7*;

/\* just in case there no content\*/

padding: *20px* *25px* *30px*;

margin: *0* *auto* *25px*;

margin-top: *50px*;

/\* shadows and rounded borders \*/

-moz-border-radius: *2px*;

-webkit-border-radius: *2px*;

border-radius: *2px*;

-moz-box-shadow: *0px* *2px* *2px* *rgba(0,* *0,* *0,* *0.3)*;

-webkit-box-shadow: *0px* *2px* *2px* *rgba(0,* *0,* *0,* *0.3)*;

box-shadow: *0px* *2px* *2px* *rgba(0,* *0,* *0,* *0.3)*;

}

*.profile-img-card* {

width: *96px*;

height: *96px*;

margin: *0* *auto* *10px*;

display: *block*;

-moz-border-radius: *50%*;

-webkit-border-radius: *50%*;

border-radius: *50%*;

}

/\*

\* Form styles

\*/

*.profile-name-card* {

font-size: *16px*;

font-weight: *bold*;

text-align: *center*;

margin: *10px* *0* *0*;

min-height: *1em*;

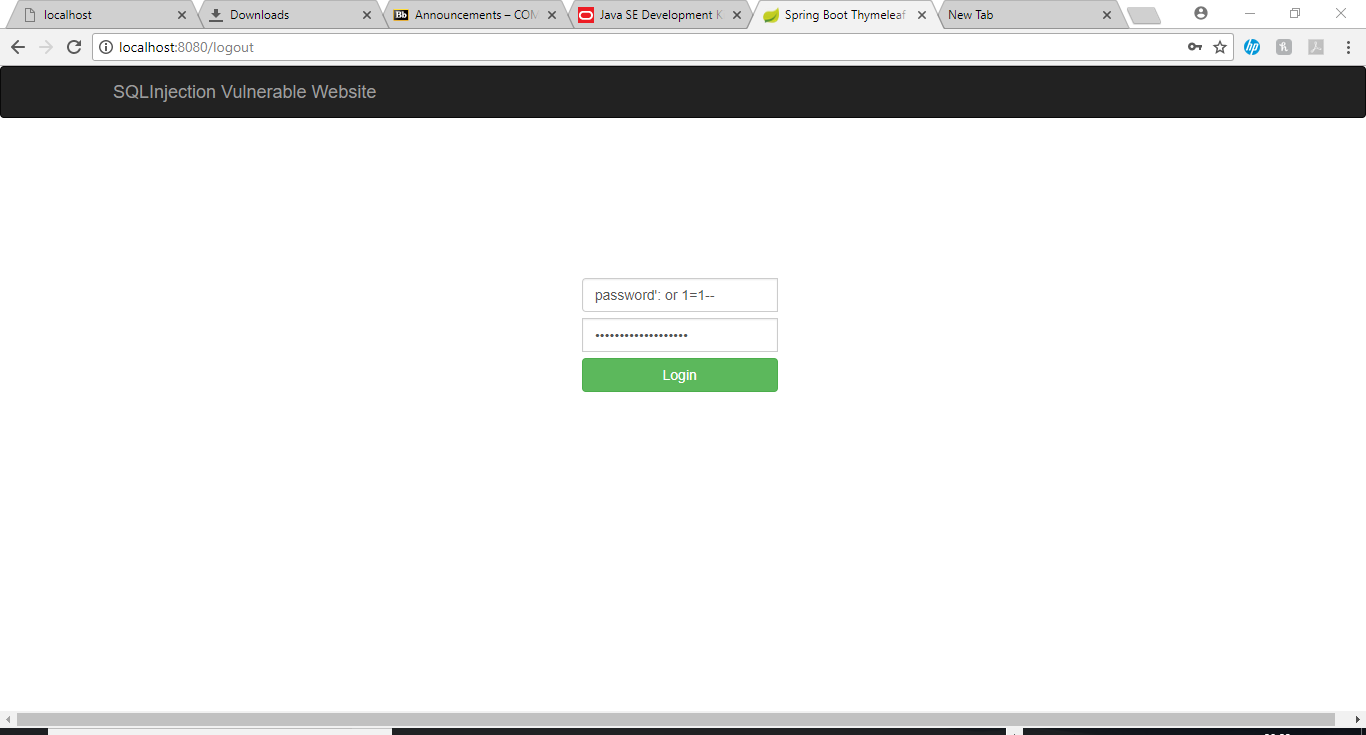
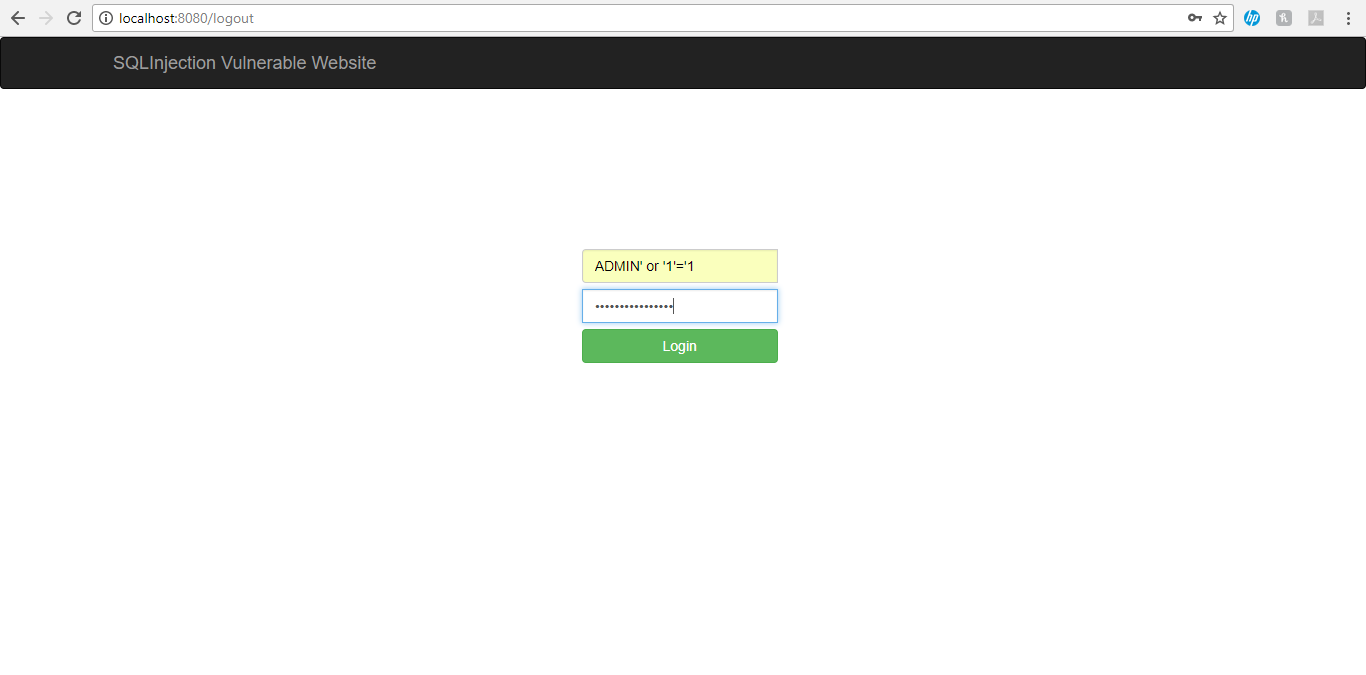
----------------------------------------------------------x—0---x-------------------------------------------------------------------------------------

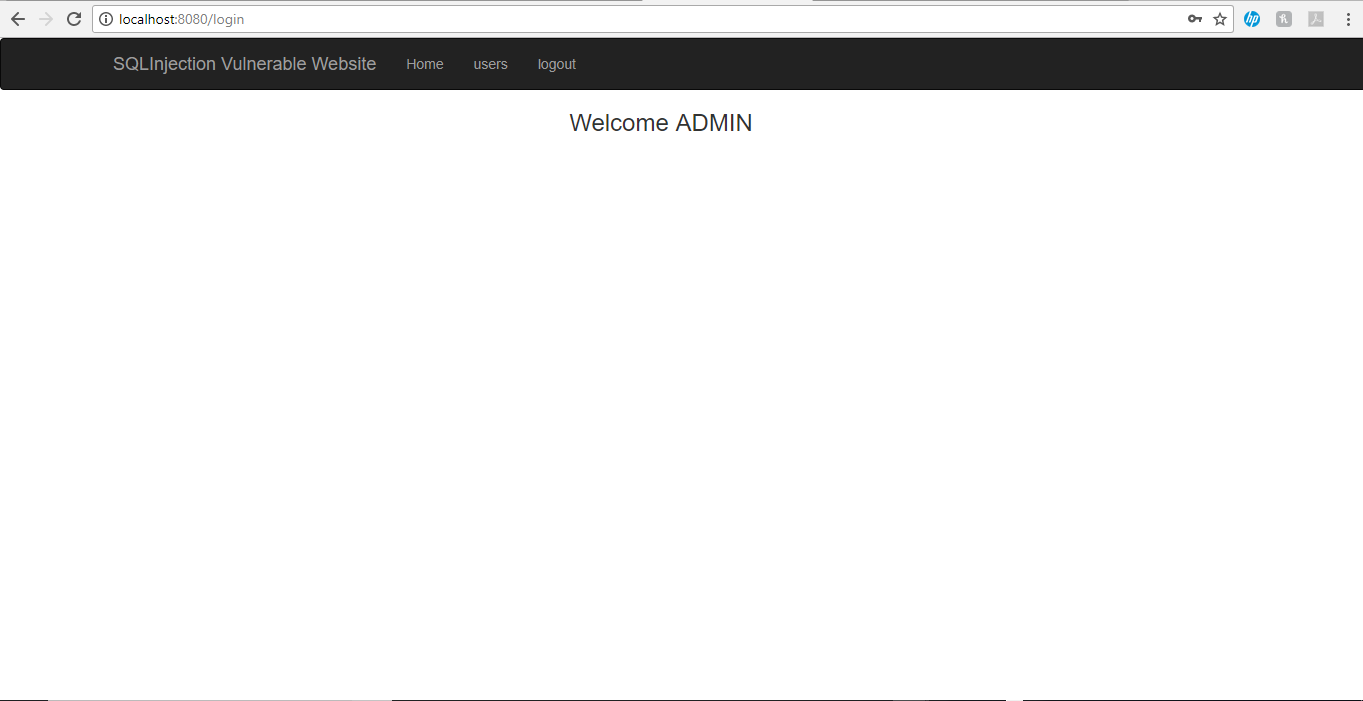
Instructions to execute the code-

1. Firstly, run the java source code in java compiler such as eclipse or InelliJ.
2. Secondly, compile it and run the code for vulnerable website as a maven file
3. Open the local web browser and type localhost:8080 for accessing the website.
4. Attack by SQL injection commands and modify the database.

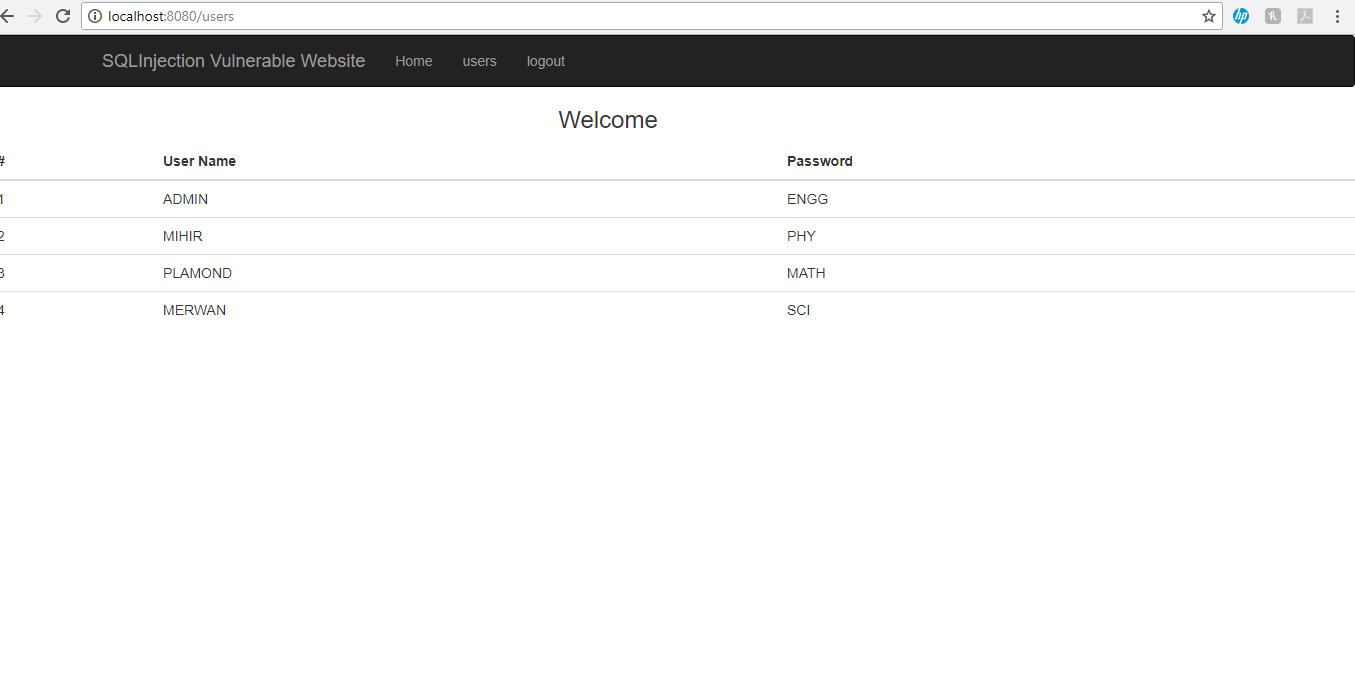
***OUTPUT SCREENS***

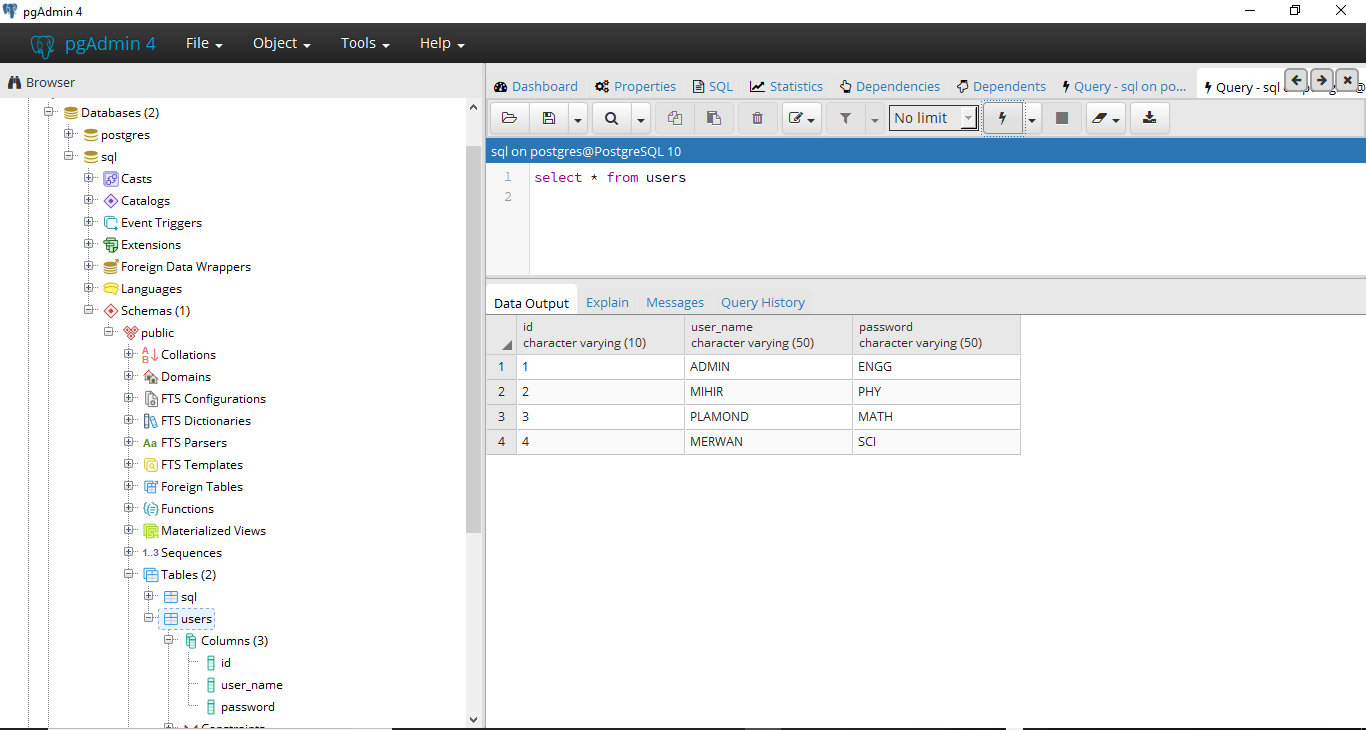
1. First login user page appears of website-





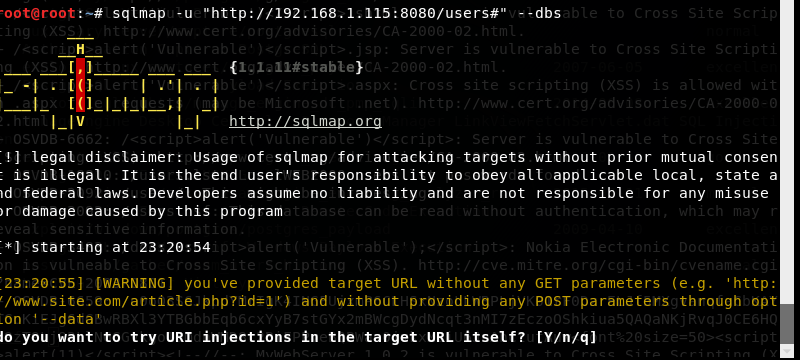
Access and modification of database

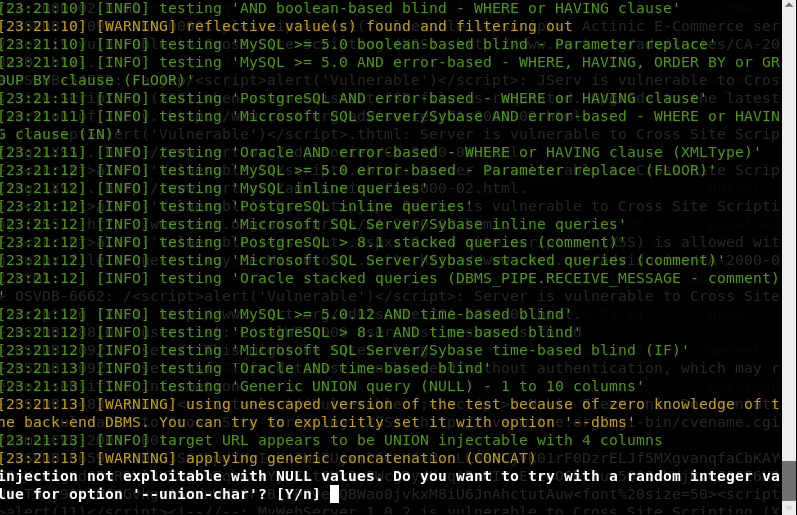


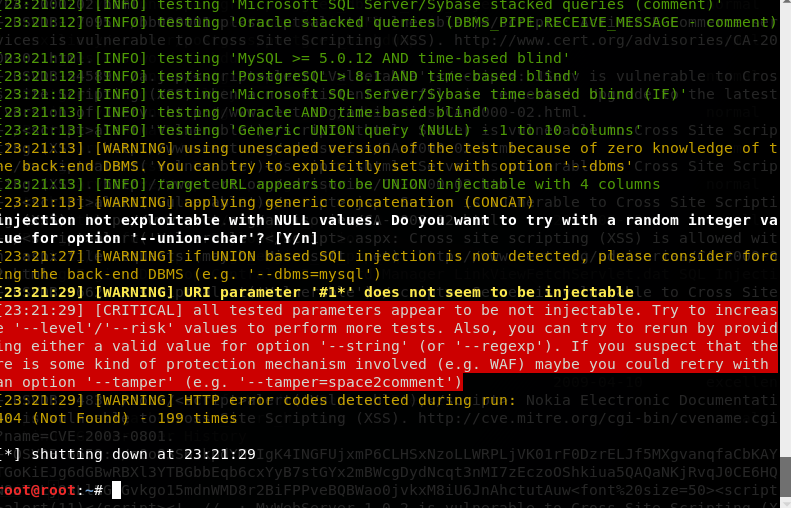


But, if guest user logins he won’t be able to see admins file and other users password.

In Kali Linux-



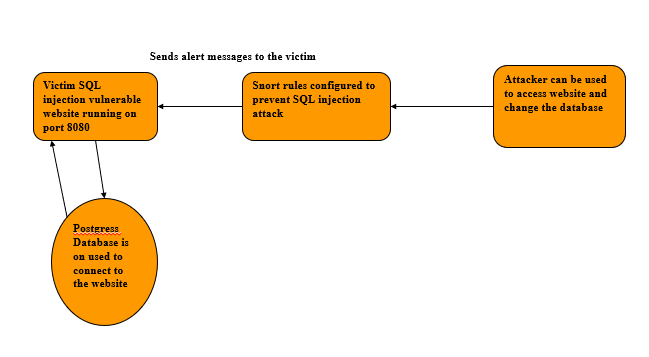




As we see, the attack is firstly not injectable, but after sql queries and commands we csn easily access the database.

**Phase 2 Defending the SQL Injection attack (SNORT)**

**Flow Chart**



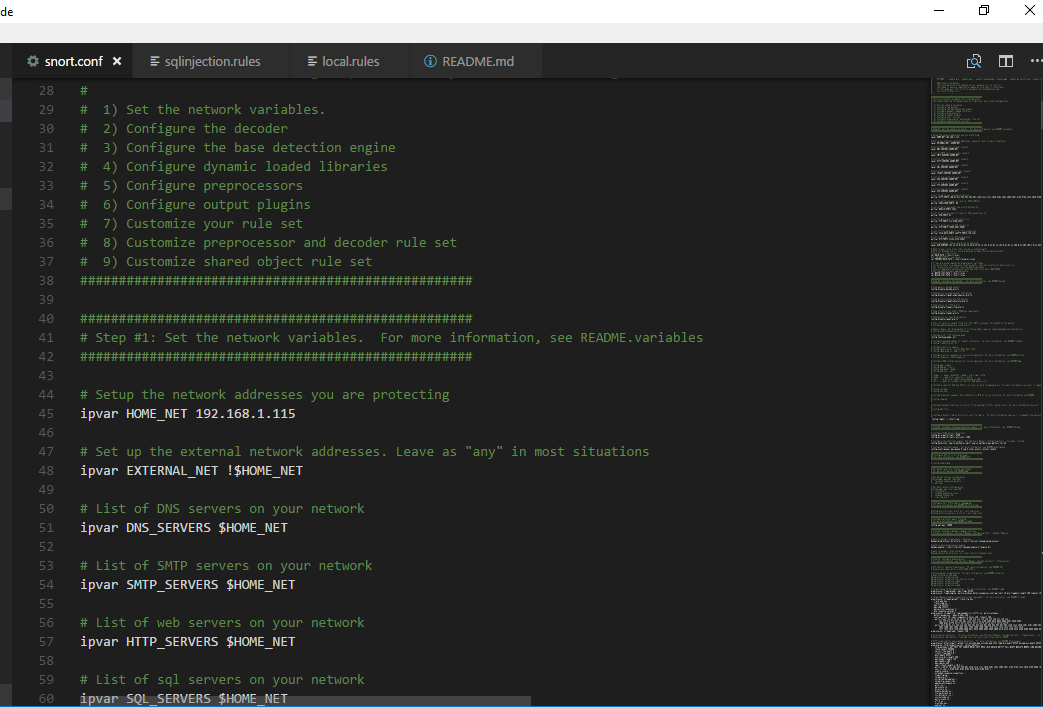
**Protocols used to prevent the attack are**

**ICMP, TCP, UDP, HTTP**

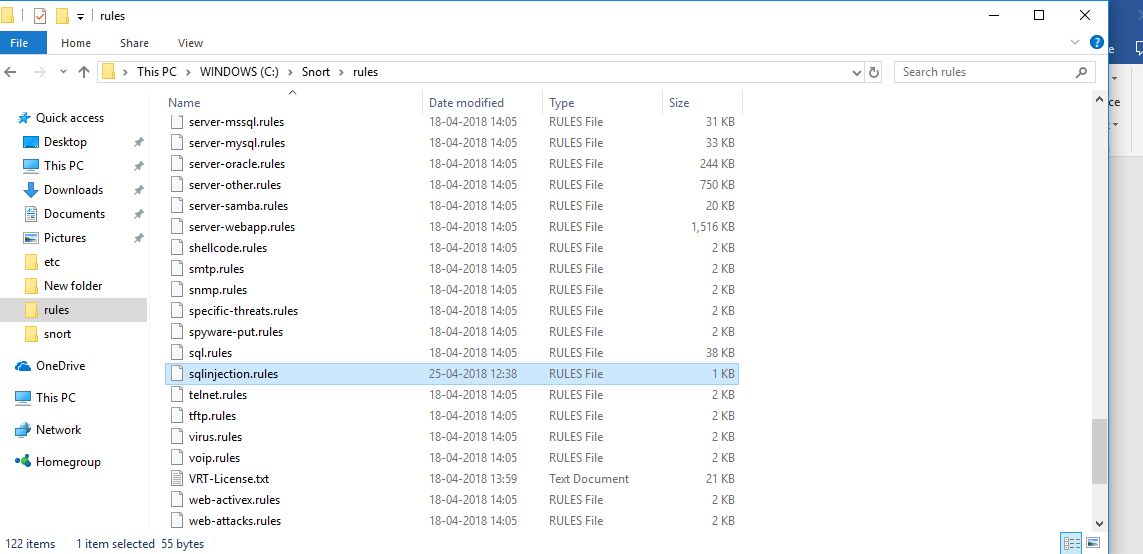
Instructions to run the attack:-

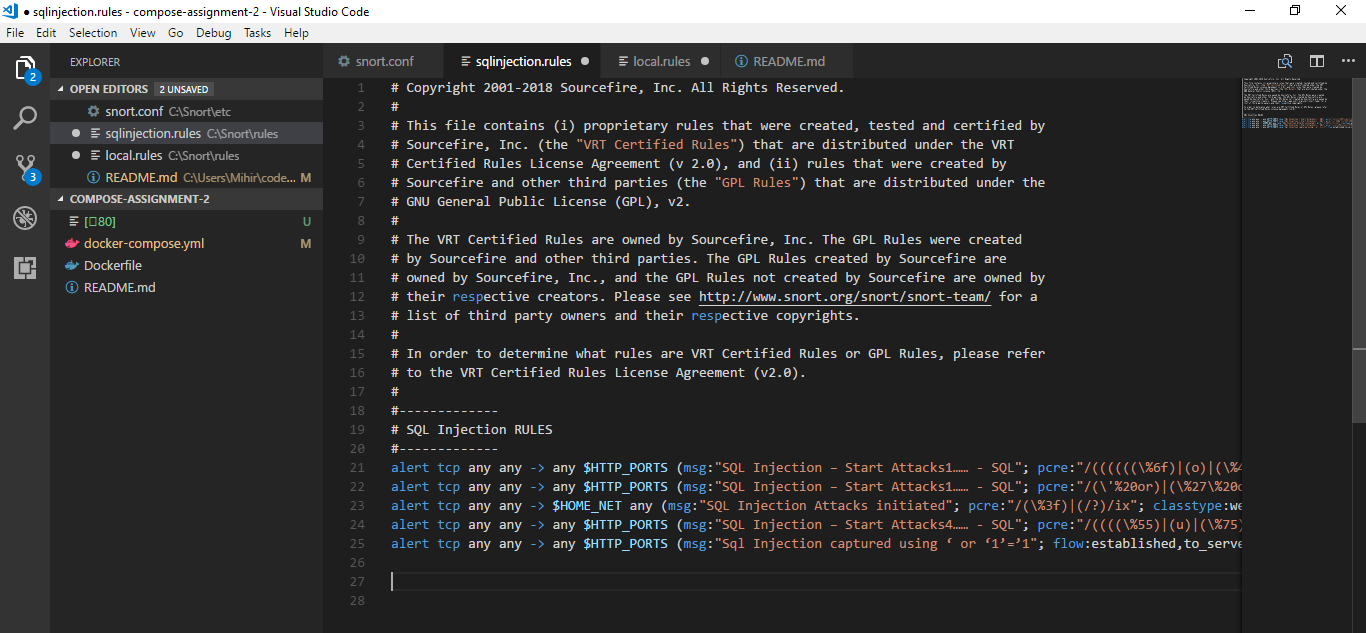
1. Import the project in Eclipse -> File-> Import -> From existing sources
2. Perform the SQL injection as above
3. Use the customized Snort rules as shown in the below screenshot
4. Log messages will be generated according to the alert

Source Code while installing snort and the configuration changes for the home network and the destination network screenshot (snort.conf)

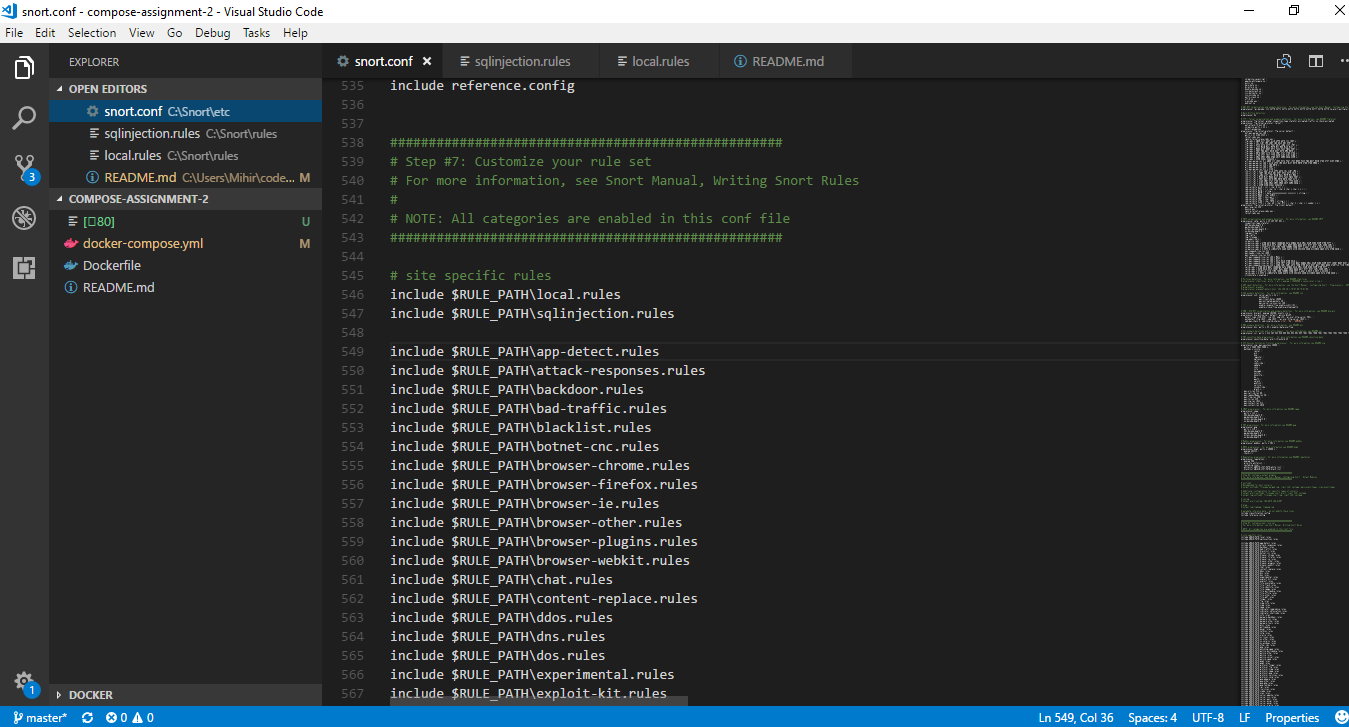


SQL injection detection rule file which has been created to detect attacks

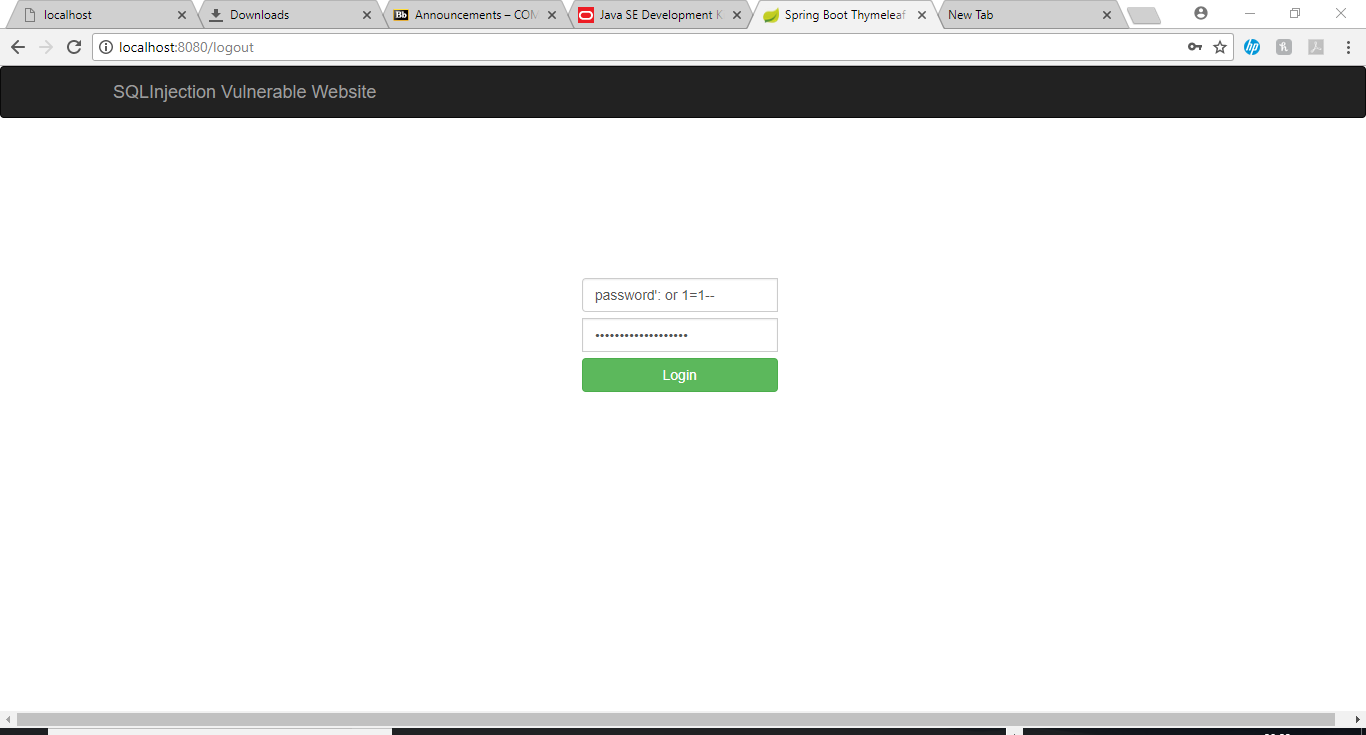
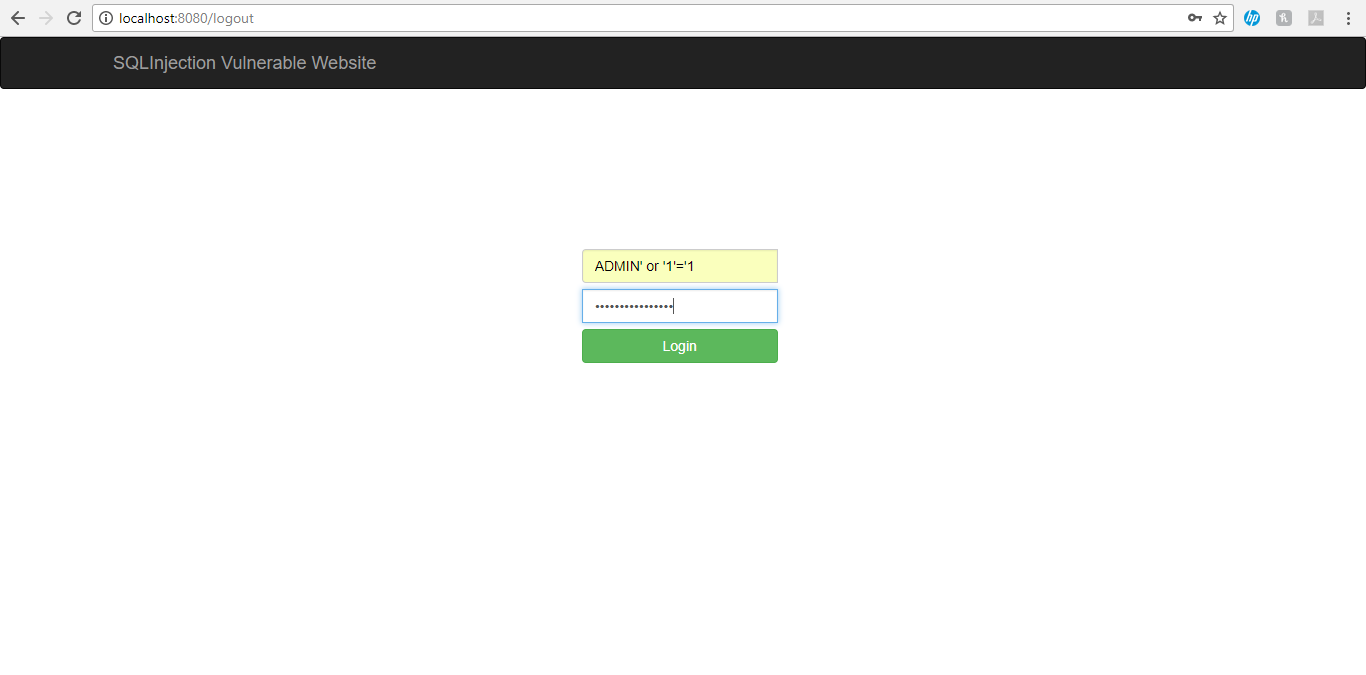


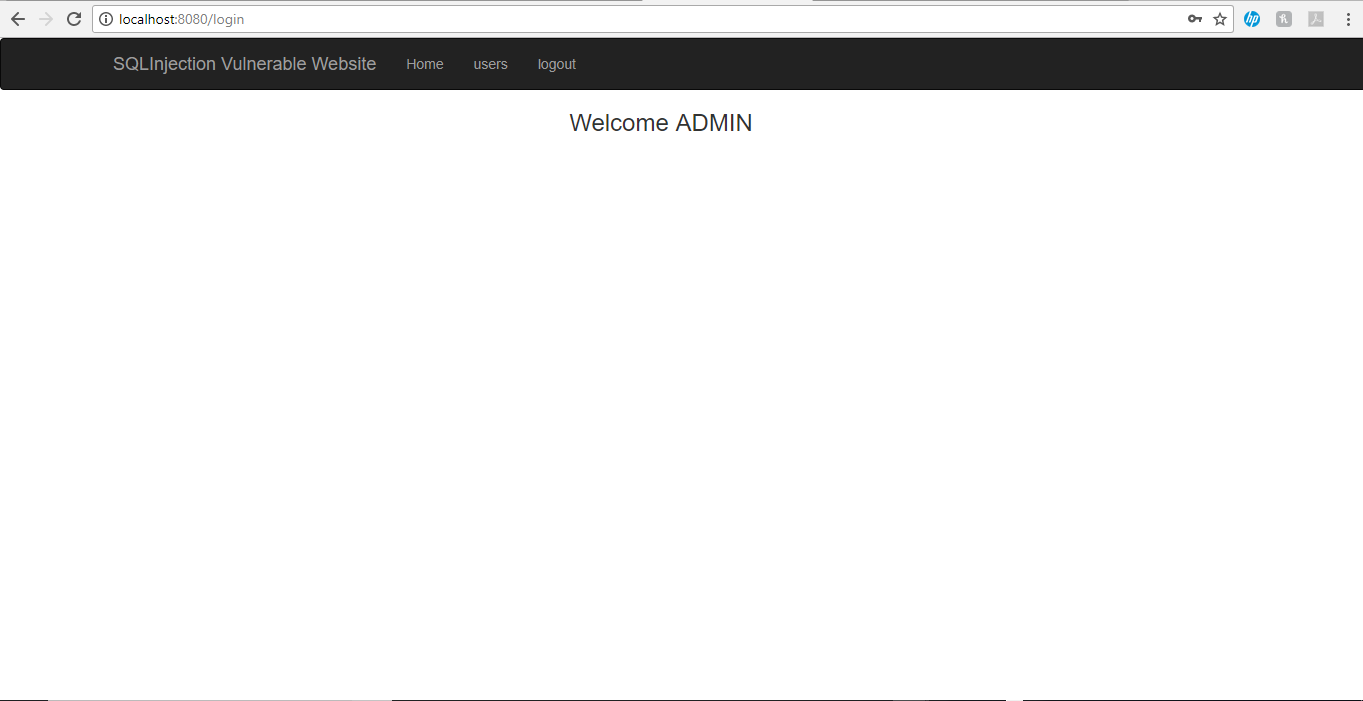


Including these rules in the rule path file while initializing Snort on the interface

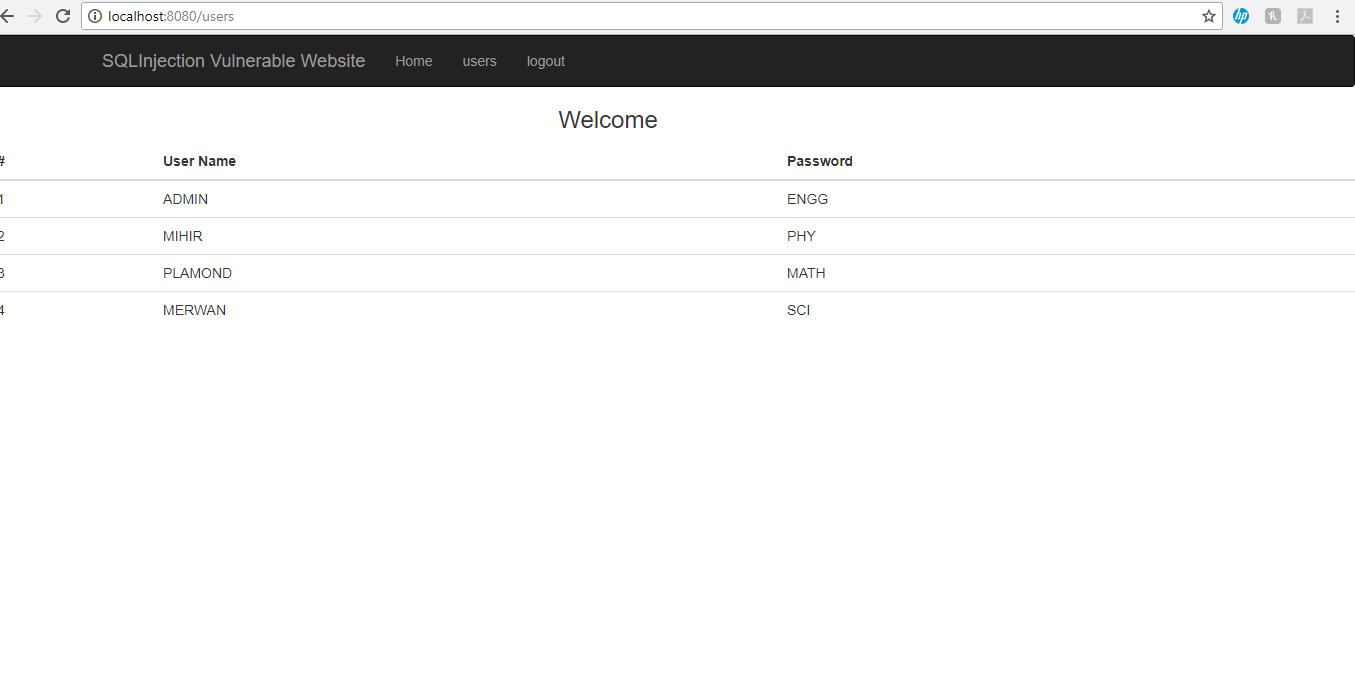


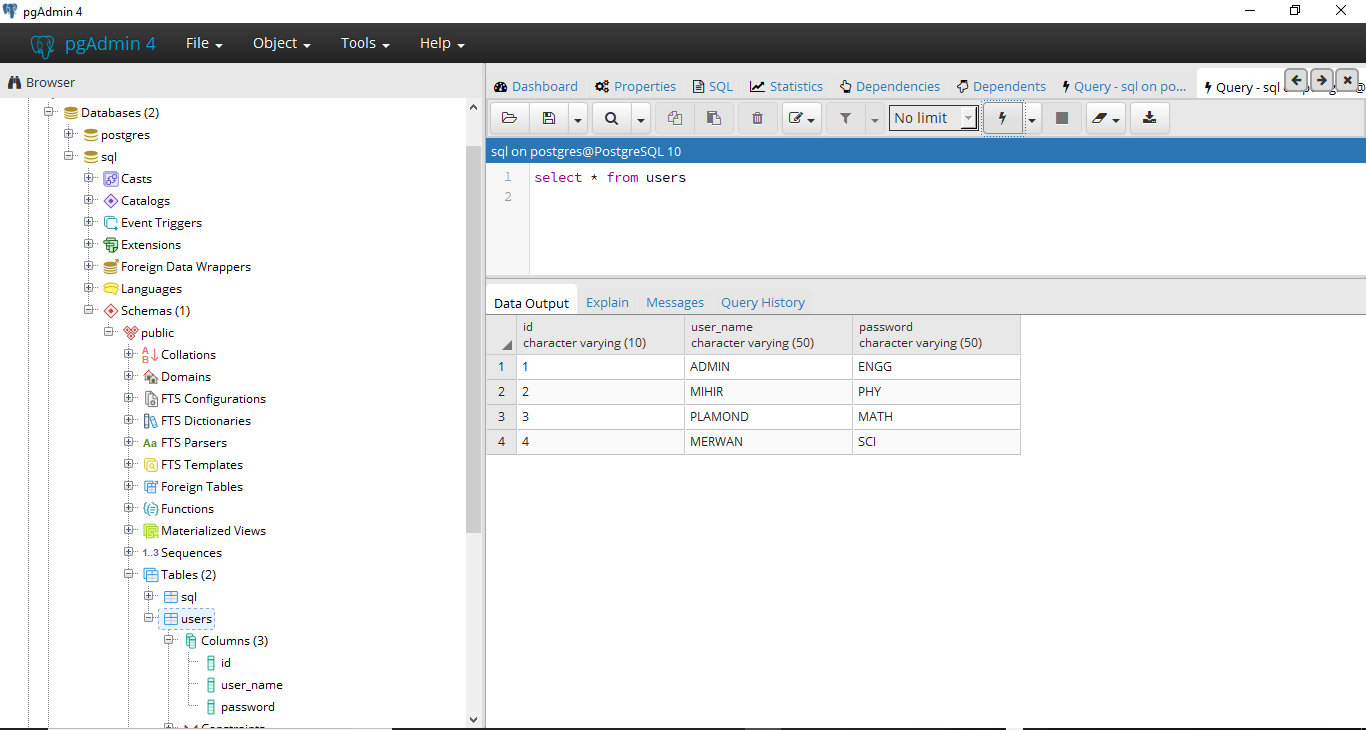
Initializing the attack



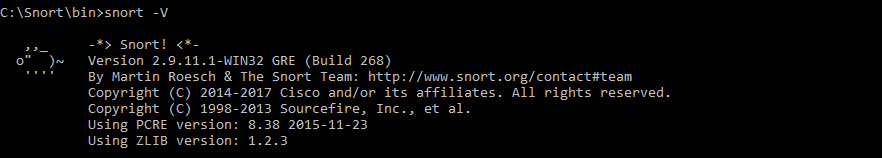


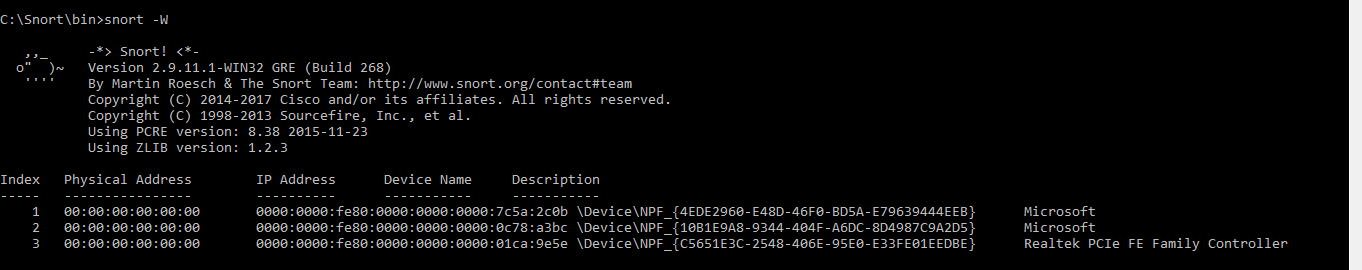
Access and modification of database



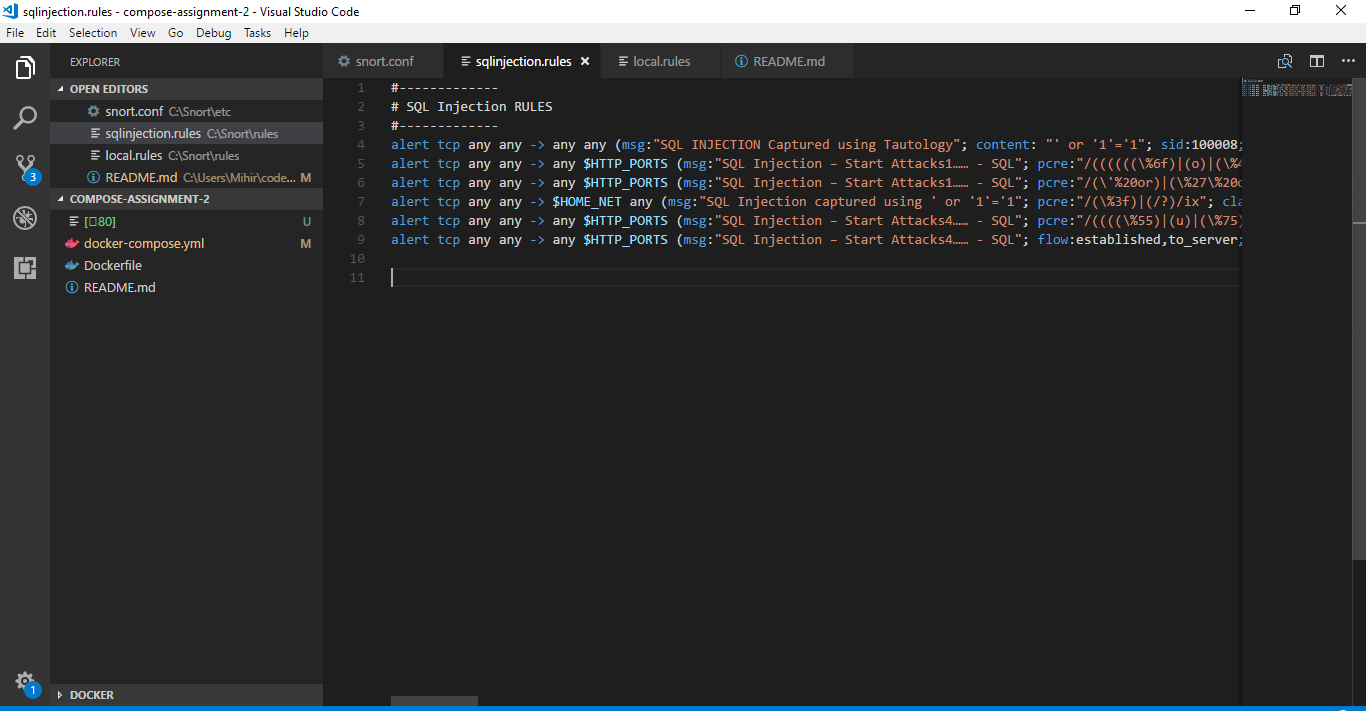


Defending this attack using Snort rules





Now after performing SQL injection the logs will be captured in the console and the log files as well.



#-------------

# SQL Injection RULES

#-------------

#-------------

#-------------

alert tcp any any -> any $HTTP\_PORTS (msg:"SQL Injection – Start Attacks1…… - SQL"; pcre:"/((((((\%6f)|(o)|(\%4f))((\%52)|(r)|(\%72))(%20))|(((\%6f)|(o)|(\%4f))((\%52)|(r)|(\%72))((%2b)|(\+)))) |((\%0a))|(((((\%6f)|(o)|(\%4f))((\%72)|(r)|(\%52))((\/)|(\%2f))((\\*)|(\%2a)))))|(((((\%4c)|(l)|(\%6c))((\%69)|(i )|(\%49))((\%6b)|(k)|(%4b))((\%65)|(e)|(\%45)))))|(((((\%63)|(c)|(\%43))((\%6f)|(o)|(\%4f))((\%6e)|(n)|(\%4 e))((\%63)|(c)|(\%43))((\%61)|(a)|(\%41))((\%74)|(t)|(\%54))((\%76)|(v)|(\%56))((\%65)|(e)|(\%45))((\%72)| (r)|(\%52))((\%73)|(s)|(\%53))((\%69)|(i)|(\%49))((\%6f)|(o)|(\%4f))((\%6e)|(n)|(\%4e)))|(((\%68)|(h)|(\%48) )((\%6f)|(o)|(\%4f))((\%73)|(s)|(\%53))((\%74)|(t)|(\%54))((\%6e)|(n)|(\%4e))((\%61)|(a)|(\%41))((\%6d)|(m) |(\%4d))((\%65)|(e)|(\%45)))|(((\%55)|(u)|(\%75))((\%55)|(u)|(\%75))((\%49)|(i)|(\%69))((\%44)|(d)|(\%64))) |(((\%64)|(d)|(\%44))((\%61)|(a)|(\%41))((\%74)|(t)|(\%54))((\%61)|(a)|(\%41))((\%64)|(d)|(\%44))((\%69)|(i )|(\%49))((\%72)|(r)|(\%52)))))))/i"; classtype:Web-application-attack; sid:9383; rev:19;)

alert tcp any any -> any $HTTP\_PORTS (msg:"SQL Injection – Start Attacks1…… - SQL"; pcre:"/(\'%20or)|(\%27\%20or)|(\%27\%20\%6f%72)|(\%27\%20\%6f%52)|(\%27\%20\%4f%72)|(\%27\% 20\%4f%52)|(\%27\%20\O%52)|(\%27\%20\%4fR)|(\%27\%20\%6fR)|(\%27\%20\O%72)|(\%20or)|(\%20\ %6f%72)|(\%20\%6f%52)|(\%20\%4f%72)|(\%20\%4f%52)|(\%20\O%52)|(\%20\%4fR)|(\%20\%6fR)|(\% 20\O%72)|(\%0aor)|(\%0a\%6f%72)|(\%0a\%6f%52)|(\%0a\%4f%72)|(\%0a\%4f%52)|(\%0a\O%52)|(\%0a \%4fR)|(\%0a\%6fR)|(\%0a\O%72)|(\\*%2for)|(\\*/\or)|(\%2a/\or)|(\%2a%2for)|(\%2a%2f\%6f%72)|(\%2a%2f\%6f%52)|(\%2a%2f\%4f%72)|(\%2a2f\%4f%52)|(\%2a%2f\O%52)|(\%2a%2f\%4fR)|(\%2a%2f\%6fR)|( \%2a%2f\O%72)|(like)|(\%6c\%69\%6b\%65)|(\%4c\%49\%4b\%45)|(concat)|(\%63\%6f\%6e\%63\%61\ %74)|(\%43\%4f\%4e\%43\%41\%54)|(version)|(\%76\%65\%72\%73\%69\%6f\%6e)|(\%56\%45\%52\%5 3\%49\%4f\%4e)|(hostname)|(\%68\%6f\%73\%74\%6e\%61\%6d\%65)|(\%48\%4f\%53\%54\%4e\%41\ %4d\%45)|(uuid)|(\%55\%55\%49\%44)|(\%75\%75\%69\%64)|(datadir)|(\%64\%61\%74\%61\%64\%69\ %72)|(\%44\%41\%54\%41\%44\%49\%52)//i"; classtype:Web-application-attack; sid:9383; rev:19;)

alert tcp any any -> $HOME\_NET any (msg:"SQL Injection Attacks initiated"; pcre:"/(\%3f)|(/?)/ix"; classtype:web-application-attack; sid:1100; rev:21; )

alert tcp any any -> any $HTTP\_PORTS (msg:"SQL Injection – Start Attacks4…… - SQL"; pcre:"/((((\%55)|(u)|(\%75))((\%4e)|(n)|(\%6e))((\%69)|(i)|(\%49))((\%6f)|(o)|(\%4f))((\%4e)|(n)|(\%6e)))[^\ n]\*(((\%73)|(s)|(\%53))((\%65)|(e)|(\%45))((\%6c)|(l)|(\%4c))((\%65)|(e)|(\%45))((\%63)|(c)|(\%43))((\%74)|(t)|(\%54))))/i"; classtype: Web-application-attack; sid:9397; rev:28;)

alert tcp any any -> any $HTTP\_PORTS (msg:"Sql Injection captured using ‘ or ‘1’=’1"; content: ‘ or ‘1’=’1;flow:established,to\_server; pcre:"/((\?)[^\n]\*(\=)[^\n]\*((\%55)|(u)|(\%75))((\%4e)|(n)|(\%6e))((\%69)|(i)|(\%49))((\%6f)|(o)|(\%4f))((\%4e)|(n)|(\%6e)))/i"; classtype: Web-application-attack; sid:9397; rev:28;)

**Explanation of snort rules-**

* The above Snort rule for SQL injection states that alert the snort IDS when there is an incoming query form any TCP network to any incoming networks including the home network at $ means all iof the http ports including message stated in commencing packet processing as Sql injection captured using ‘ or ‘1’=1’ flow.
* PCRE engine is being used for regular expression matching on the payload, each rule in snort is translated by PCRE compiler for regular expression rule defined and followed by specific set of variables.
* Class type is mentioned as web-application attack in any server on pc, and service id (Sid) is 9397
* Revision number is how many times the rule has being changed or manipulated.
* The attack traffic is caught on content mentioned if the query is made in use on any of the bad traffic rather than normal traffic, it is filtered out and using decoder and detection engine of snort.
* Read/Apply are used between rule file and output file to check the bad potential traffic.

**Flowchart of the Snort rules**

|  |
| --- |
| **1)Snort captures the packet** |
| 1. **Use the decoder to decode the packet** |
| 1. **Preprocessor the packet** |
| **4)Detection engine** |
| **5) If a rule has been matched then triggers the message to the victim** |
| 1. **Output plug** |
| 1. **Outputs the messages in the log files , sockets and console** |
| 1. **The whole process continues for the network traffic of the interface selected to rule the snort software tool** |

**Snort command and explanation-**

**snort -I 1 -c c:\Snort\etc\\snort.conf -A console -I c:\snort\log\**

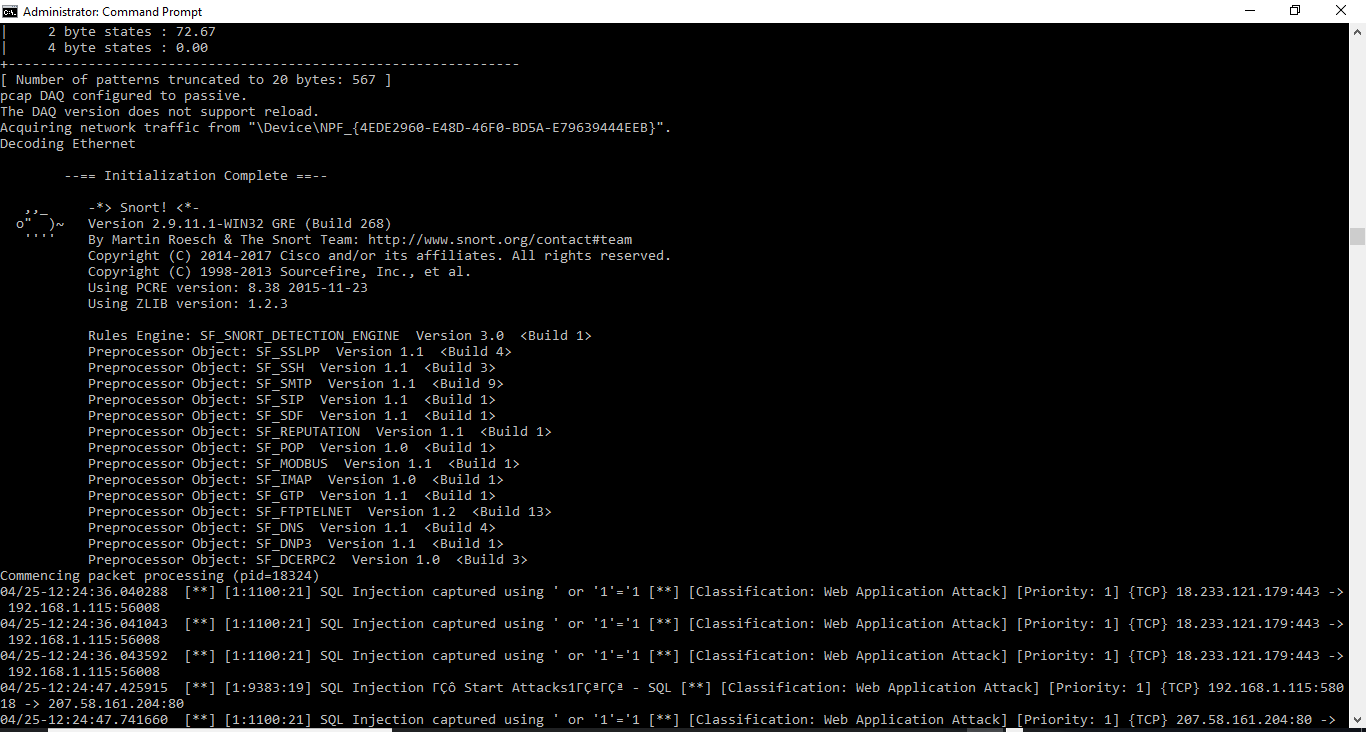
**The above command means -I being the interface through which the snort would perform its IDS functions allotted in my PC at 1 (network adapter of Microsoft), maybe be different in other user cases.**

**-c is used for designationg the configuration file, in my case the config file is saved in c:\Snort\etc\\\snort.conf**

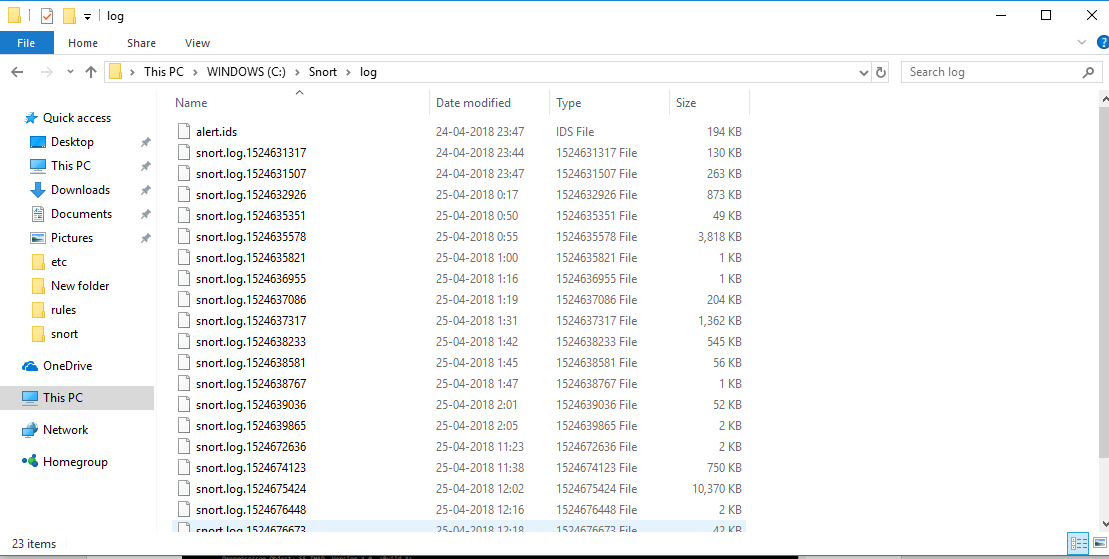
**-A console provides direct output to the cmd**

**C:\snort\log\ simultaneously the log files are created in log file in snort destination of the OS**

**alerts.ids file output**



Logs files have been generated



Knowledge gained

* How to configure snort config file which can be useful for preventing for lot of attacks in future and install it with the log files and interfaces
* Operation of the network packet flow when snort detects the triggered rule.

Problems faced

* Creating specific rules to trigger the attack
* Installing snort on the correct interface
* Creating port-based rules to trigger warnings
* Writing and forming sql injection rules

***REFERENCES***

* <https://www.eclipse.org/>
* <https://www.kali.org/downloads/>
* <https://www.sans.org/reading-room/whitepapers/securecode/sql-injection-modes-attack-defence-matters-23>
* <http://www.ijcce.org/papers/244-E091.pdf>
* <https://www.pgadmin.org/download/pgadmin-4-windows/>

<https://www.cisco.com/c/en/us/about/security-center/sql-injection.html>