ATTACK DEFENSE

by PentesterAcademy

Name	BDD (Behavior Driven Development) Security
URL	https://www.attackdefense.com/challengedetails?cid=2063
Type	DevSecOps Basics: Dynamic Code Analysis

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Challenge Description

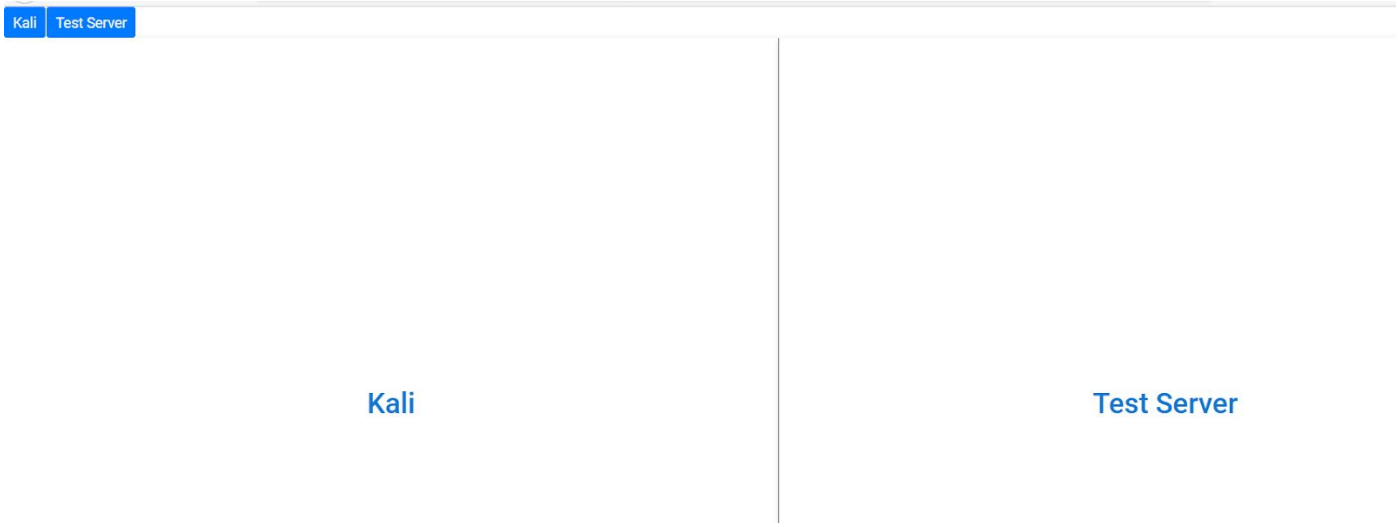
BDD-Security is a Behaviour Driven Development (BDD) security testing framework that uses tools like OWASP ZAP, SSLyze, Nessus to scan the web applications.

A Kali GUI machine (kali-gui) is provided to the user with BDD-Security on it. The source code for the web application is provided in the home directory of the root user. The WebGoat instance can be reached at the 'test-server' endpoint at port 8080.

Objective: Scan BDD security to find issues in the code!

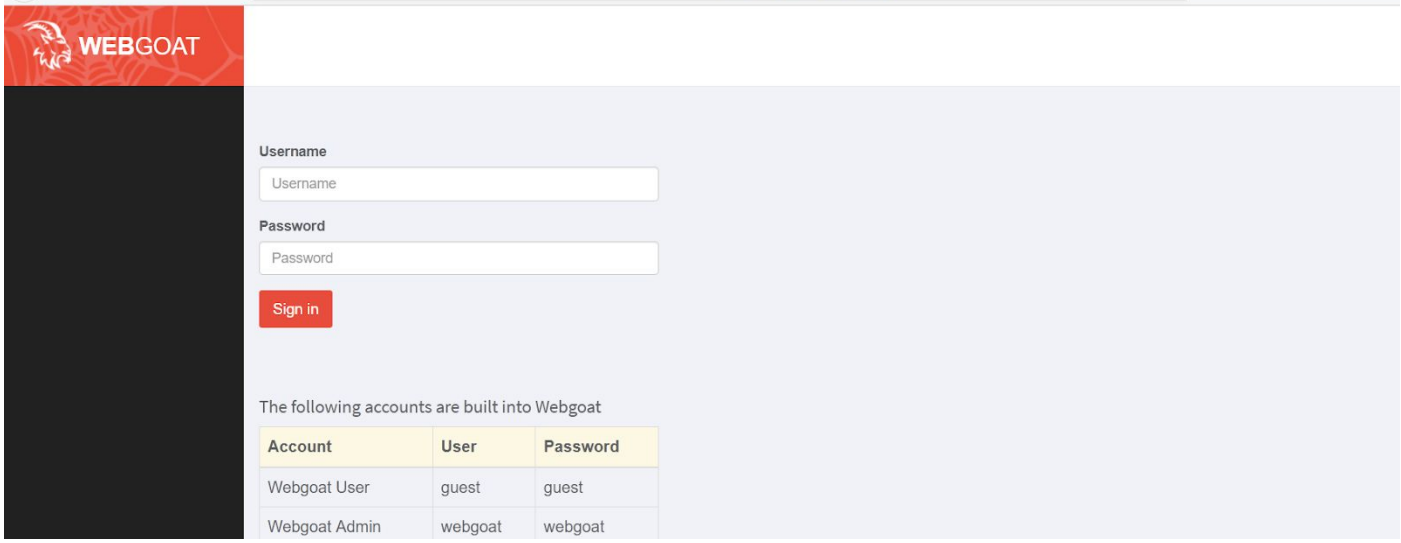
Lab Setup

On starting the lab, the following interface will be accessible to the user.



On choosing (clicking the text in the center) left panel, **a Kali GUI instance** will open in a new tab.

Similarly on selecting the right panel, a web UI of **WebGoat** will open in a new tab.



Solution

Example I: Local Web Application (RopeyTasks)

In this example, we will use the code provided on the Kali machine to host the web application on local port 8080.

Step 1: Check the provided source code of the web application.

Commands:
cd github-repos/RopeyTasks
ls

```
root@kali-gui:~# cd github-repos/RopeyTasks/
root@kali-gui:~/github-repos/RopeyTasks#
root@kali-gui:~/github-repos/RopeyTasks# ls
README.md                grails-app    prodDb.lock.db                ropeytasks.jar
application.properties   grailsw       prodDb.trace.db               test
build.standalone.sh      prodDb.h2.db  ropeytasks-grailsPlugins.iml  web-app
root@kali-gui:~/github-repos/RopeyTasks#
```

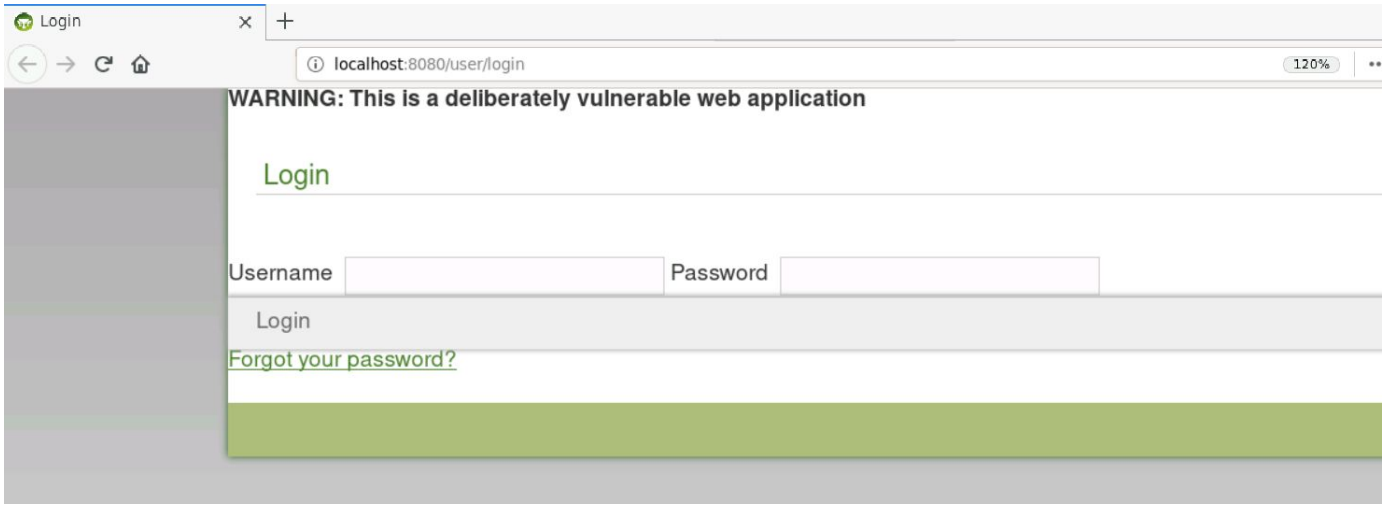
Step 2: Start the RopeyTasks server on localhost to test.

Command: java -jar ropeytasks.jar

```
root@kali-gui:~/github-repos/RopeyTasks# java -jar ropeytasks.jar
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
2020-09-19 13:38:22.825:INFO:oejs.Server:jetty-7.x.y-SNAPSHOT
2020-09-19 13:38:24.020:INFO:oejpw.PlusConfiguration:No Transaction manager found - if your webapp requires
one, please configure one.
2020-09-19 13:38:24.057:INFO:oejw.StandardDescriptorProcessor:NO JSP Support for , did not find org.apache.j
asper.servlet.JspServlet
2020-09-19 13:38:24.912:INFO::Initializing Spring root WebApplicationContext
2020-09-19 13:38:34.341:INFO:oejsh.ContextHandler:started o.e.j.w.WebAppContext{,file:/tmp/standalone-war/em
bedded6591073198614602452-exploded-1600502902343/},/tmp/standalone-war/embedded6591073198614602452-exploded-
1600502902343
2020-09-19 13:38:34.385:INFO::Initializing Spring FrameworkServlet 'grails'
2020-09-19 13:38:34.413:INFO:oejs.AbstractConnector:Started SelectChannelConnector@localhost:8080
Server running. Browse to http://localhost:8080
```

The ropeytasks server has been started on port 8080 of the localhost.

Step 3: Open the http://localhost:8080 URL in the web browser to check the web portal.



Step 4: Open another terminal and navigate to the BDD security directory

Commands:
cd ~/bdd-security
ls

```
root@attackdefense:~# cd bdd-security/
root@attackdefense:~/bdd-security#
root@attackdefense:~/bdd-security# ls
Dockerfile  build         config.xml  gradlew      lib          log4j.properties  zap
README.md   build.gradle  gradle      gradlew.bat  license.txt  src
root@attackdefense:~/bdd-security#
```

Step 5: The BDD security is set to default for testing the ropeytasks server.

Config.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<web-app>

   <!-- The settings in this file are for the demo ropey-tasks vulnerable web app available at:
https://github.com/stephendv/RopeyTasks,
         which is included in the bdd-security framework for demo purposes. -->

   <!-- The web driver to use, can be either Firefox, Chrome or HtmlUnit.  Optionally specify path to the driver
(required for linux)
       Some drivers require a path to the platform specific driver binary, for example chrome needs chromedriver.  If
these values are not specified, we'll use HtmlUnit
   <defaultDriver>firefox</defaultDriver>
   <defaultDriver path="src/test/resources/drivers/chromedriver-mac">Chrome</defaultDriver> -->

   <!-- Base URL of the application to test -->
   <baseUrl>http://localhost:8080/</baseUrl>

   <!-- A Java class to hold the Selenium steps to test the application in depth. Optionally required for in-depth
authn/z and session management testing. -->
   <class>net.continuumsecurity.examples.ropeytasks.RopeyTasksApplication</class>

   <!-- In order to install sslyze on a Linux system, these steps must be followed
         apt-get update
         apt-get install python-pip
         pip install sslyze
   -->
   <sslyze>
      <path>sslyze</path>
         <option>--regular</option>
      <targetHost>www.continuumsecurity.net</targetHost>
      <targetPort>443</targetPort>
   </sslyze>

   <!-- Optional names of the session ID cookies for session management testing. -->
```

```
    <sessionIds>
        <name>JSESSIONID</name>
    </sessionIds>

    <!-- the default user to use when logging in to the app -->
    <defaultUsername>bob</defaultUsername>
    <defaultPassword>password</defaultPassword>

    <scanner>
        <ignoreUrl>.*logout.*</ignoreUrl>
            <spiderUrl>baseUrl</spiderUrl>
            <maxDepth>5</maxDepth>
    </scanner>

    <!-- An upstream proxy through which all HTTP traffic must pass before hitting the target
    application under test.  The framework will configure both the WebDriver instance and ZAP to use this proxy.  Note
that non-HTTP traffic will not use this proxy. -->
    <upstreamProxy>
        <host></host>
        <port></port>
        <noProxyHosts></noProxyHosts><!-- ie: localhost,127.0.0.1,192.168.10.2 -->
    </upstreamProxy>

    <incorrectPassword>SDFsdfwjx1</incorrectPassword>
    <incorrectUsername>bobbles</incorrectUsername>

    <!-- Optional login credentials for the Nessus server, the server location is specified in the nessus_scan.story file
-->
    <nessus>
        <username>admin</username>
        <password>admin</password>
    </nessus>

    <!-- Optional location of a running OWASP ZAP instance.  Either an external- already running ZAP instance must
be specified here, or the zapPath must be specified to launch ZAP
    <proxy>
        <host>127.0.0.1</host>
        <port>8888</port>
        <api></api>
    </proxy>-->

    <zapPath>zap/zap.sh</zapPath>

</web-app>
```

Step 6: Start the scan on RopeyTasks server using gradlew (Gradlew is the wrapper of gradle).

Commands: ./gradlew -Dcucumber.options="--tags @authentication --tags ~@skip" test

The options defined:

--tags @authentication = Only performs authentication checks

--tags ~@skip = Perform checks except the 'skip' feature

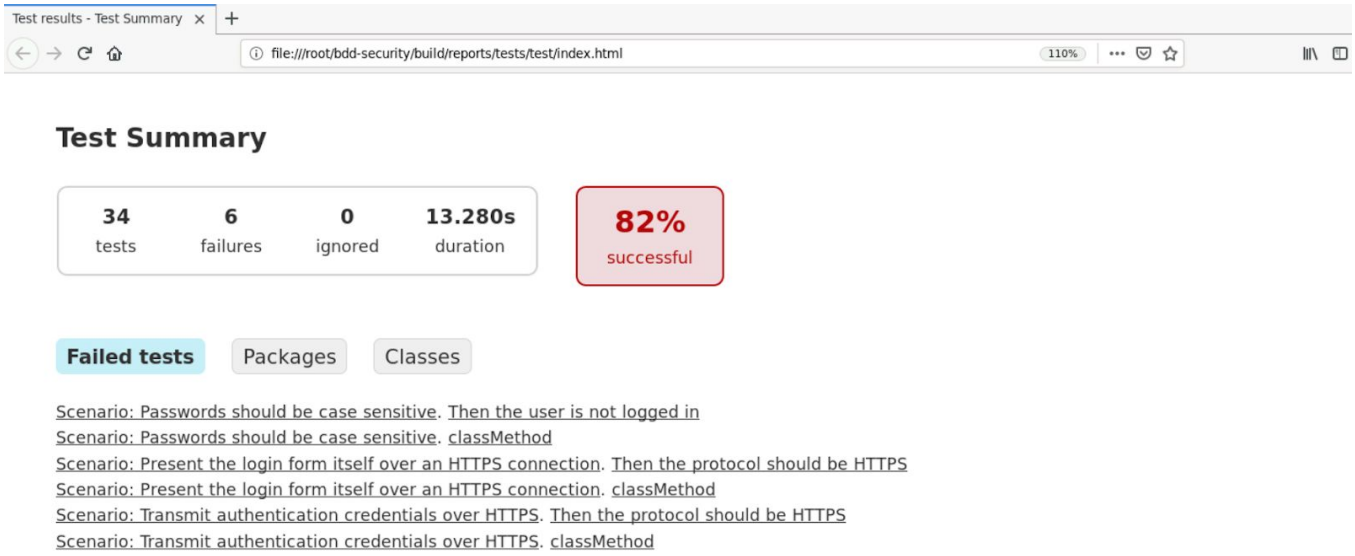
```
root@attackdefense:~/bdd-security# ./gradlew -Dcucumber.options="--tags @authentication --tags ~@skip" test
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Starting a Gradle Daemon, 1 incompatible and 1 stopped Daemons could not be reused, use --status for details

> Task :test
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Found Java version 1.8.0_212
Available memory: 32166 MB
Setting jvm heap size: -Xmx8041m
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
0 [main] INFO org.zaproxy.zap.DaemonBootstrap  - OWASP ZAP 2.6.0 started 09/09/20 13:39:25
36 [main] INFO org.parosproxy.paros.common.AbstractParam  - Setting config scanner.threadPerHost = 20 was 20
37 [main] INFO org.parosproxy.paros.common.AbstractParam  - Setting config spider.thread = 10 was 10
37 [main] INFO org.parosproxy.paros.common.AbstractParam  - Setting config api.key = zapapisecret was zapapi
secret
```

```
> Task :generateReportTask

Report available on: /root/bdd-security/build/reports/cucumber/pretty/feature-overview.html


FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':test'.
> There were failing tests. See the report at: file:///root/bdd-security/build/reports/tests/test/index.html

* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get more log outpu
t.

BUILD FAILED in 23s
5 actionable tasks: 2 executed, 3 up-to-date
root@attackdefense:~/bdd-security#
```

The build has failed due to some errors. Now, to check the errors open the report generated by bdd-security.

Step 7: Open the test report generated from the scan.

Command: firefox file:///root/bdd-security/build/reports/tests/test/index.html



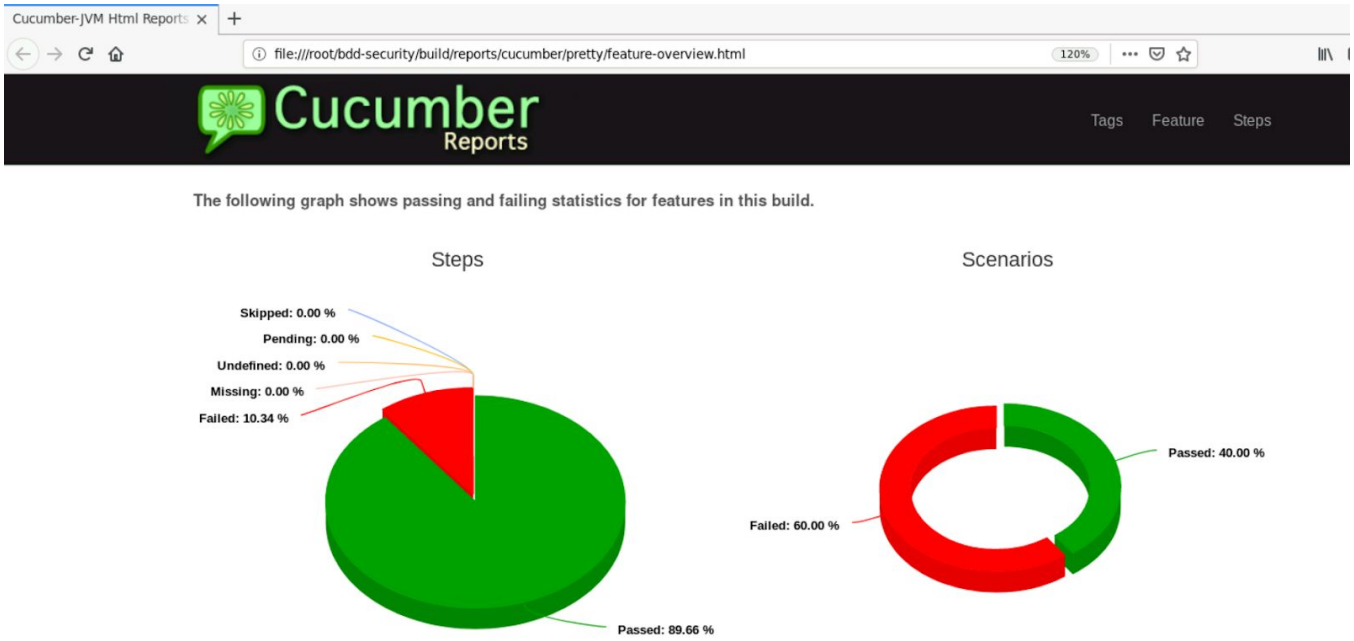
Issue Detected:

- Credentials are transmitted over HTTP

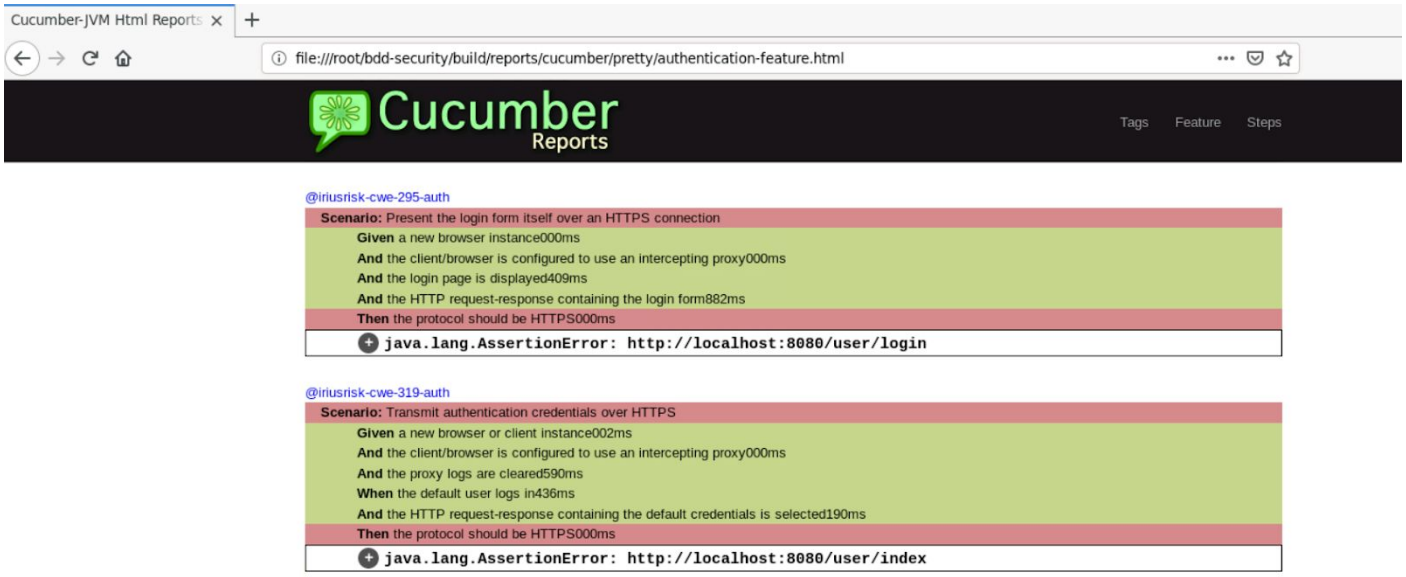
Step 8: Open the “function report” generated by bdd-security using firefox. The function report is located at build/reports/cucumber/pretty/feature-overview.html

Note: Cucumber is a tool which generates reports based on the tests which have been passed or failed. Cucumber uses reporter plugin in order to generate the reports.

Command: firefox /root/bdd-security/build/reports/cucumber/pretty/feature-overview.html



This is an overview report. Scroll down and click on the 'Authentication' to check the errors encountered during the authentication phase.

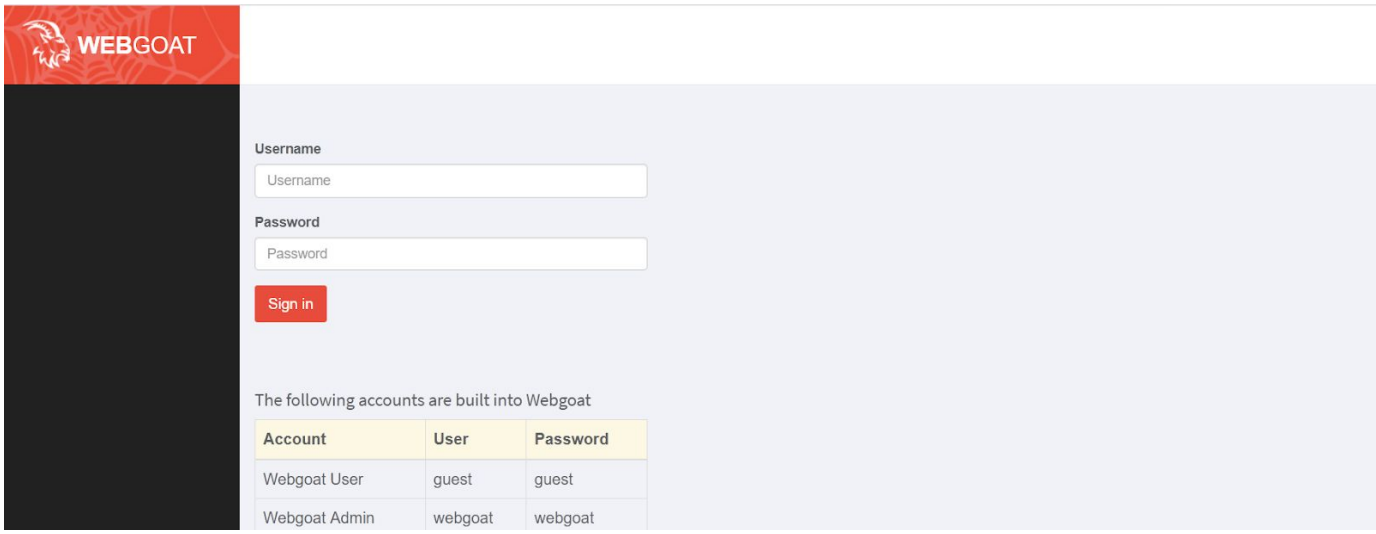


It can be observed that authentication is done on HTTP protocol which should have been performed over HTTPS.

Example 2: Remote Web Application (WebGoat)

In this example, we will test the web application hosted on port 8080 of the 'test-server'.

Step 1: Open the remote Application



Step 2: Modify the base URL, class and credentials in the configuration file of BDD security to test the web application.

Modified values:

- **BaseURL:** http://test-server:8080/
- **ClassName:** net.continuumsecurity.WebGoatApplication
- **DefaultUsername:** guest
- **DefaultPassword:** guest

The config.xml file can be found at bdd-security/config.xml

Modified Config.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<web-app>

    <!-- The settings in this file are for the demo ropey-tasks vulnerable web app available at:
https://github.com/stephendv/RopeyTasks,
```

```
        which is included in the bdd-security framework for demo purposes. -->

    <!-- The web driver to use, can be either Firefox, Chrome or HtmlUnit.  Optionally specify path to the driver
(required for linux)
        Some drivers require a path to the platform specific driver binary, for example chrome needs chromedriver.  If
these values are not specified, we'll use HtmlUnit
    <defaultDriver>firefox</defaultDriver>
    <defaultDriver path="src/test/resources/drivers/chromedriver-mac">Chrome</defaultDriver> -->

    <!-- Base URL of the application to test -->
    <baseUrl>http://test-server:8080/</baseUrl>

    <!-- A Java class to hold the Selenium steps to test the application in depth. Optionally required for in-depth
authn/z and session management testing. -->
    <class>net.continuumsecurity.WebGoatApplication</class>

    <!-- In order to install sslyze on a Linux system, these steps must be followed
        apt-get update
        apt-get install python-pip
        pip install sslyze
    -->
    <sslyze>
        <path>sslyze</path>
        <option>--regular</option>
        <targetHost>www.continuumsecurity.net</targetHost>
        <targetPort>443</targetPort>
    </sslyze>

    <!-- Optional names of the session ID cookies for session management testing. -->
    <sessionIds>
        <name>JSESSIONID</name>
    </sessionIds>

    <!-- the default user to use when logging in to the app -->
    <defaultUsername>guest</defaultUsername>
    <defaultPassword>guest</defaultPassword>

    <scanner>
        <ignoreUrl>.*logout.*</ignoreUrl>
        <spiderUrl>baseUrl</spiderUrl>
        <maxDepth>5</maxDepth>
    </scanner>

    <!-- An upstream proxy through which all HTTP traffic must pass before hitting the target
    application under test.  The framework will configure both the WebDriver instance and ZAP to use this proxy.  Note
that non-HTTP traffic will not use this proxy. -->
    <upstreamProxy>
```

```
        <host></host>
        <port></port>
        <noProxyHosts></noProxyHosts><!-- ie: localhost,127.0.0.1,192.168.10.2 -->
    </upstreamProxy>

    <incorrectPassword>SDFsdfwjx1</incorrectPassword>
    <incorrectUsername>bobbles</incorrectUsername>

    <!-- Optional login credentials for the Nessus server, the server location is specified in the nessus_scan.story file
-->
    <nessus>
        <username>admin</username>
        <password>admin</password>
    </nessus>

    <!-- Optional location of a running OWASP ZAP instance.  Either an external- already running ZAP instance must
be specified here, or the zapPath must be specified to launch ZAP
    <proxy>
        <host>127.0.0.1</host>
        <port>8888</port>
        <api></api>
    </proxy>-->

    <zapPath>zap/zap.sh</zapPath>

</web-app>
```

The configuration is defined for webgoat web instance, The application will perform several tests after logging into the application. Default credentials are specified in the configuration file.

Step 3: Create a test case for the remote website.

Command: vim src/test/java/net/continuumsecurity/WebGoatApplication.java

WebGoatApplication.java

```
package net.continuumsecurity;

import net.continuumsecurity.Config;
import net.continuumsecurity.Credentials;
import net.continuumsecurity.UserPassCredentials;
import net.continuumsecurity.behaviour.ILogin;
import net.continuumsecurity.behaviour.ILogout;
import net.continuumsecurity.behaviour.INavigable;
import net.continuumsecurity.web.WebApplication;
```

```
import org.openqa.selenium.By;

public class WebGoatApplication extends WebApplication implements ILogin,
        ILogout,INavigable {

        public WebGoatApplication() {
        super();
        }
        // Open the login page
        @Override
        public void openLoginPage() {
        driver.get(Config.getInstance().getBaseUrl() + "login.mvc");
        findAndWaitForElement(By.id("exampleInputEmail1"));
        }
        // Login into the application using the default credentials provided in config.xml
        @Override
        public void login(Credentials credentials) {
        UserPassCredentials creds = new UserPassCredentials(credentials);
        driver.findElement(By.id("exampleInputEmail1")).clear();
        driver.findElement(By.id("exampleInputEmail1")).sendKeys(creds.getUsername());
        driver.findElement(By.id("exampleInputPassword1")).clear();
        driver.findElement(By.id("exampleInputPassword1")).sendKeys(creds.getPassword());
        driver.findElement(By.tagName("button")).click();
        }
        // Check if user is logged in
        @Override
        public boolean isLoggedIn() {
        return true;
        }
        // Logout the application
        @Override
        public void logout() {}
}
```

This test case will:

- Try to login into the website
- Perform several checks (predefined by BDD-Security)
- Log out from the website

Step 4: Start the scan on the remote website using gradlew.

Command: ./gradlew -Dcucumber.options="--tags @authentication --tags ~@skip" test

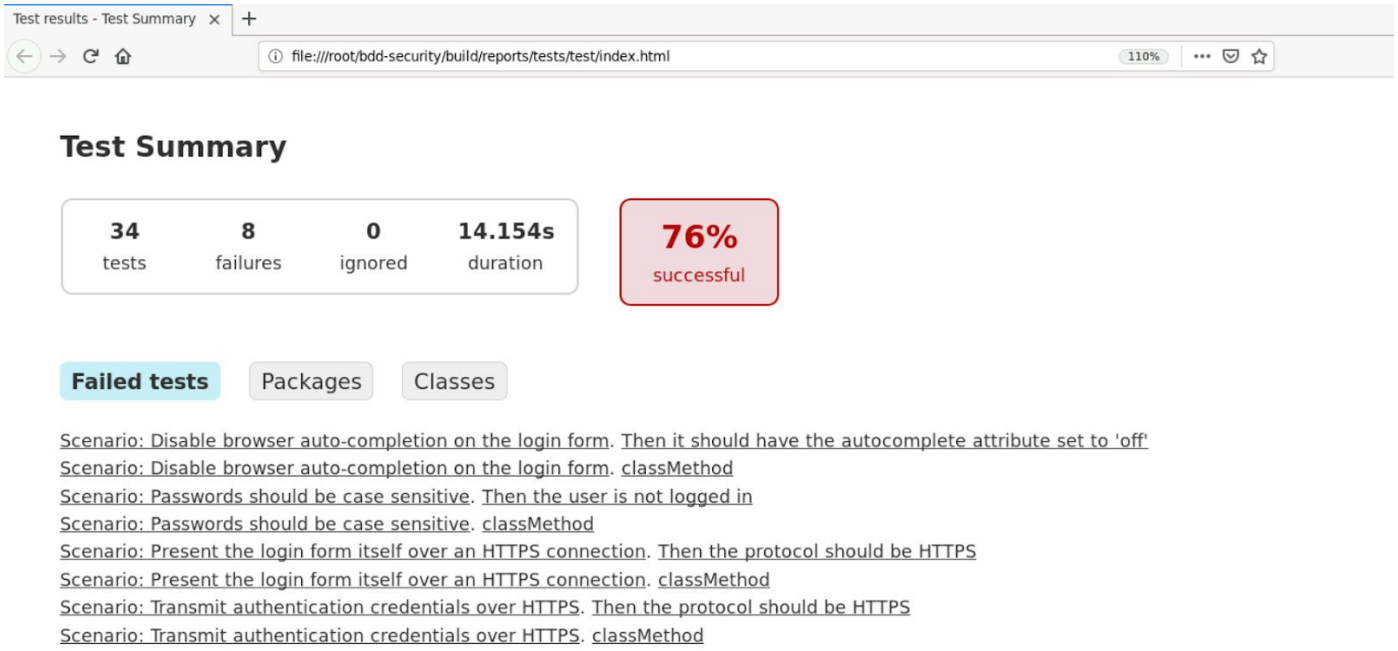
```
root@attackdefense:~/bdd-security# ./gradlew -Dcucumber.options="--tags @authentication --tags ~@skip" test
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true


> Task :compileTestJava
Note: /root/bdd-security/src/test/java/net/continuumsecurity/web/drivers/DriverFactory.java uses or override
s a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

> Task :test
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Found Java version 1.8.0_212
Available memory: 32166 MB
Setting jvm heap size: -Xmx8041m
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
0 [main] INFO org.zaproxy.zap.DaemonBootstrap  - OWASP ZAP 2.6.0 started 09/09/20 14:17:20
33 [main] INFO org.parosproxy.paros.common.AbstractParam  - Setting config scanner.threadPerHost = 20 was 20
33 [main] INFO org.parosproxy.paros.common.AbstractParam  - Setting config spider.thread = 10 was 10
34 [main] INFO org.parosproxy.paros.common.AbstractParam  - Setting config api.key = zapapisecret was zapapi
secret
```

```
> Task :generateReportTask

Report available on: /root/bdd-security/build/reports/cucumber/pretty/feature-overview.html


FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':test'.
> There were failing tests. See the report at: file:///root/bdd-security/build/reports/tests/test/index.html

* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get more log outpu
t.

BUILD FAILED in 23s
5 actionable tasks: 3 executed, 2 up-to-date
root@attackdefense:~/bdd-security#
```

The build has failed due to some errors. Now, to check the errors open the report generated by bdd-security.

Step 5: Open the test report generated from the scan.

Command: firefox file:///root/bdd-security/build/reports/tests/test/index.html

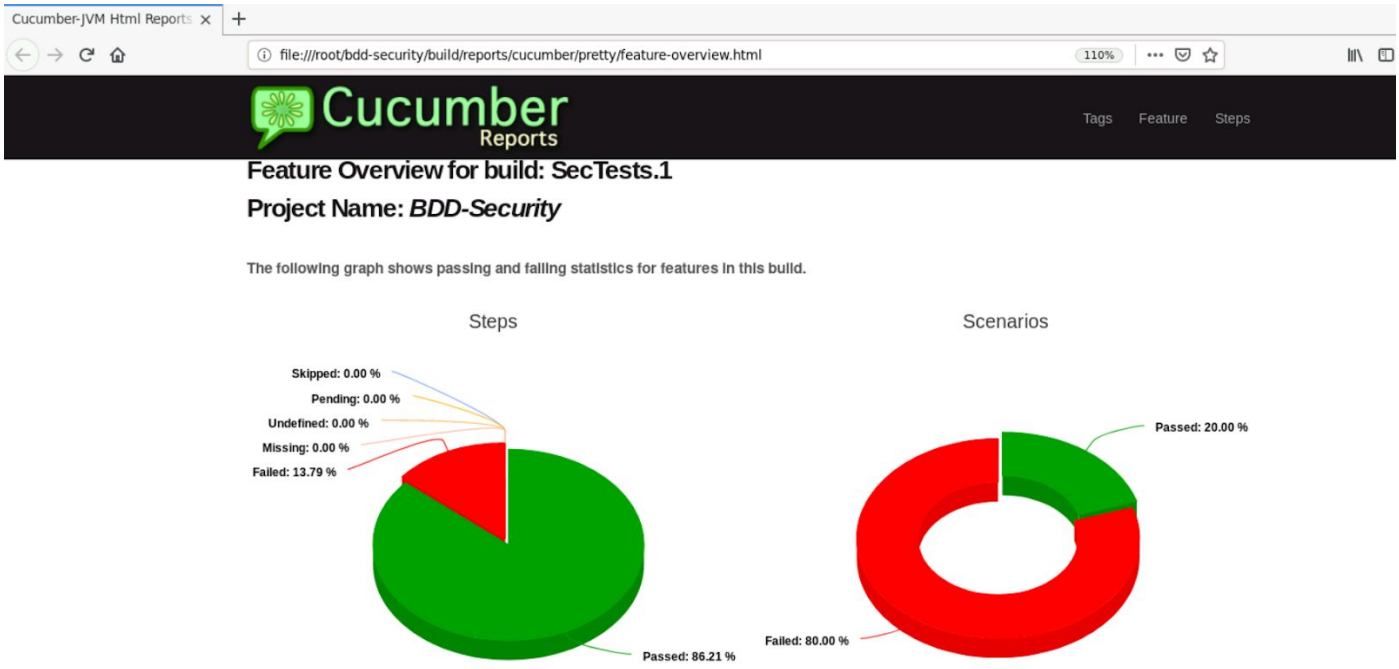


Issue Detected:

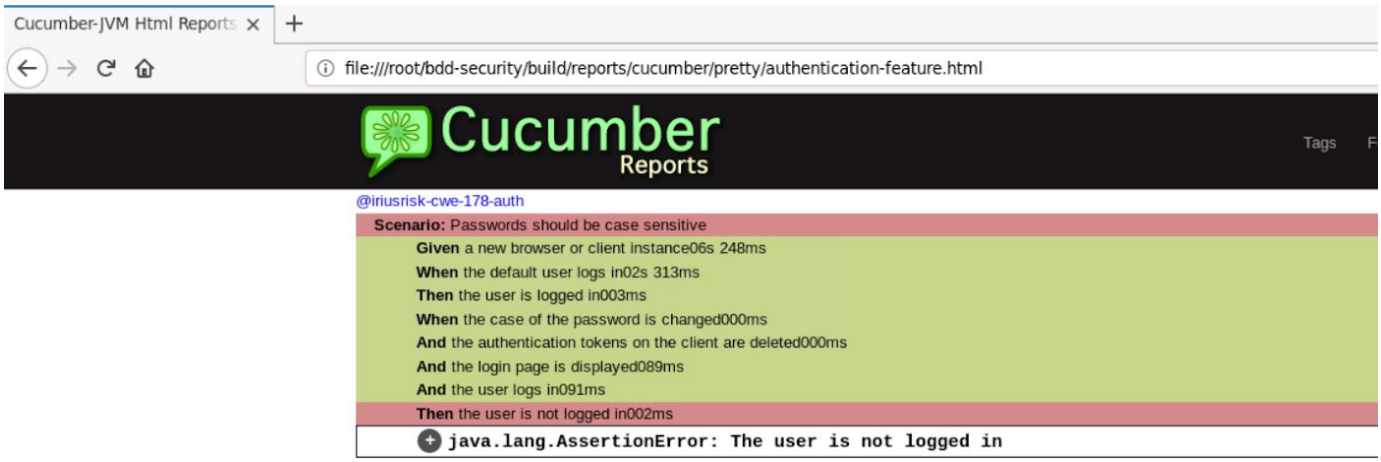
- Credentials are transmitted over HTTP

Step 6: Open the “function report” generated by bdd-security using firefox. The function report is located at build/reports/cucumber/pretty/feature-overview.html

Command: firefox /root/bdd-security/build/reports/cucumber/pretty/feature-overview.html



This is an overview report generated, scroll down and click on the 'Authentication' to check the errors generated during the authentication phase.



It can be observed that authentication is done on HTTP protocol which should have been performed over HTTPS.

Learnings

Perform Dynamic tests on the applications using BDD Security.