Problem Statement-2

Deploy a local k8s cluster (using minikube, k3s, or anything else) and deploy the DVWA application. Showcase/demo 3 attack surfaces as mentioned in its documentation.

1. Sample k8s setup

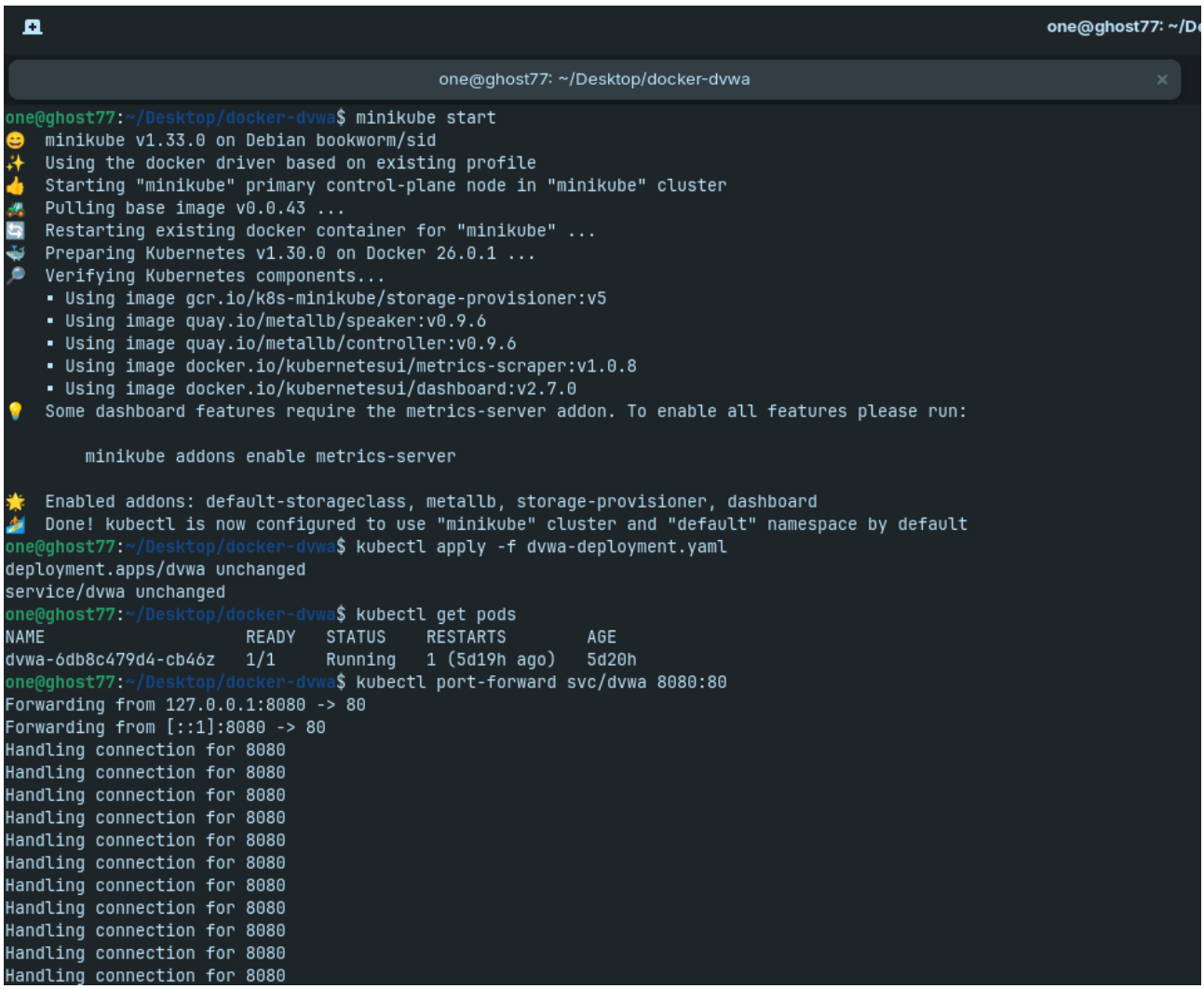
Deployment File

```
-------------------------------------------
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dvwa
spec:
  replicas: 1
  selector:
    matchLabels:
      app: dvwa
  template:
    metadata:
      labels:
        app: dvwa
    spec:
      containers:
      - name: dvwa
        image: vulnerables/web-dvwa
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: dvwa
spec:
  selector:
    app: dvwa
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80

-----------------------------------
```

The is a Kubernetes deployment and service configuration for deploying the DVWA (Damn Vulnerable Web Application) application. DVWA is intentionally designed to have multiple vulnerabilities for practicing and understanding web application security.

The deployment file deploys a single replica of the DVWA application using the vulnerables/web-dvwa Docker image. The application listens on port 80. The service definition creates a Kubernetes service named "dvwa" to expose the DVWA deployment internally within the cluster on port 80.



To showcase three attack surfaces mentioned in the DVWA documentation:

1. SQL Injection: Exploit the vulnerable login page by entering malicious SQL code in the username or password field to manipulate the database or retrieve sensitive information.

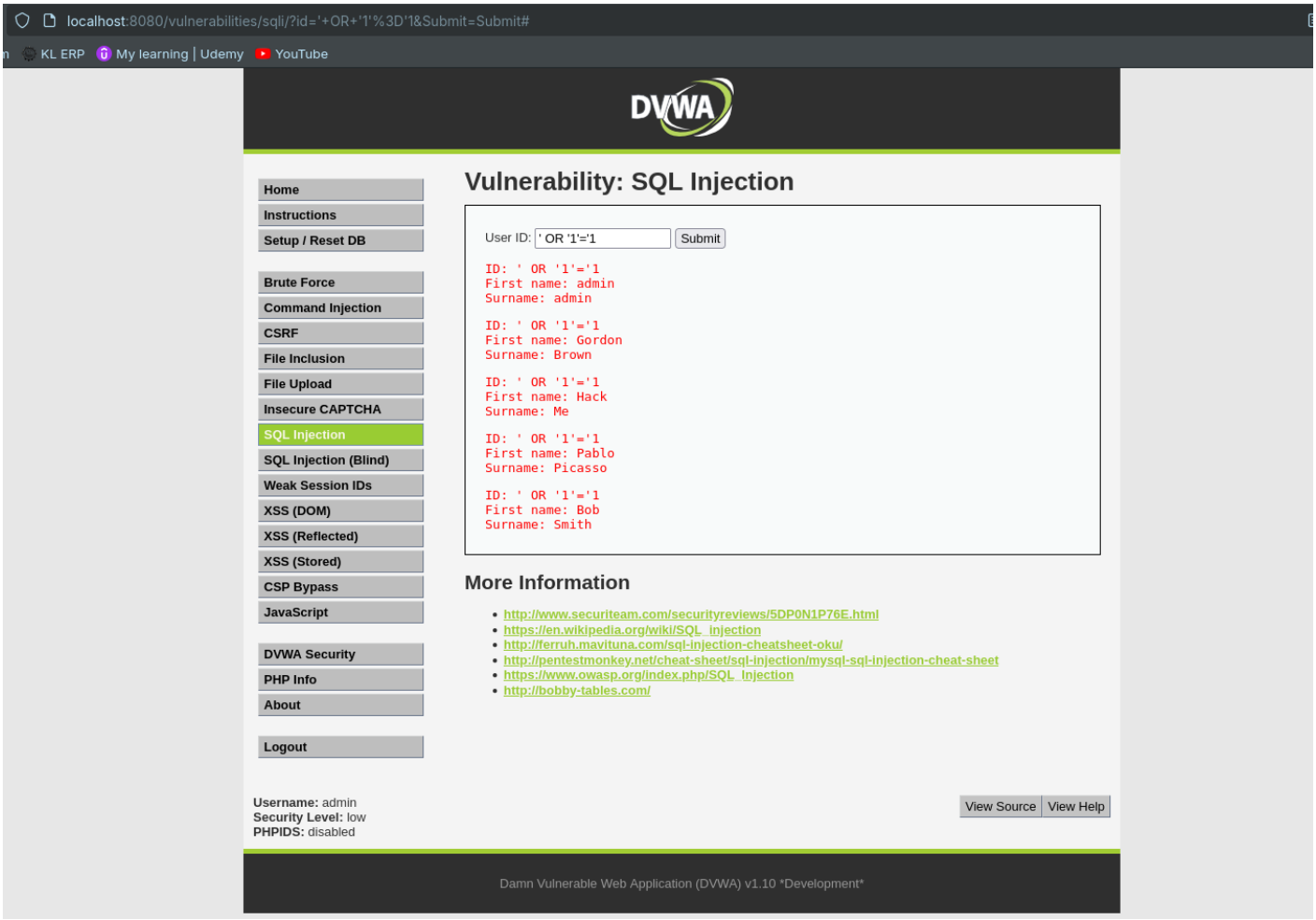
2. Cross-Site Scripting (XSS): Inject JavaScript code in the user input field vulnerable to XSS attacks to execute malicious scripts on other users' browsers.

3. Command Injection: Exploit vulnerabilities in the DVWA application to execute arbitrary commands on the underlying system by injecting malicious commands in user inputs.

The above surface attacks 1.SQL Injection 2.Cross-Site Scripting 3.Command Injection are demonstrated with images and explaination.

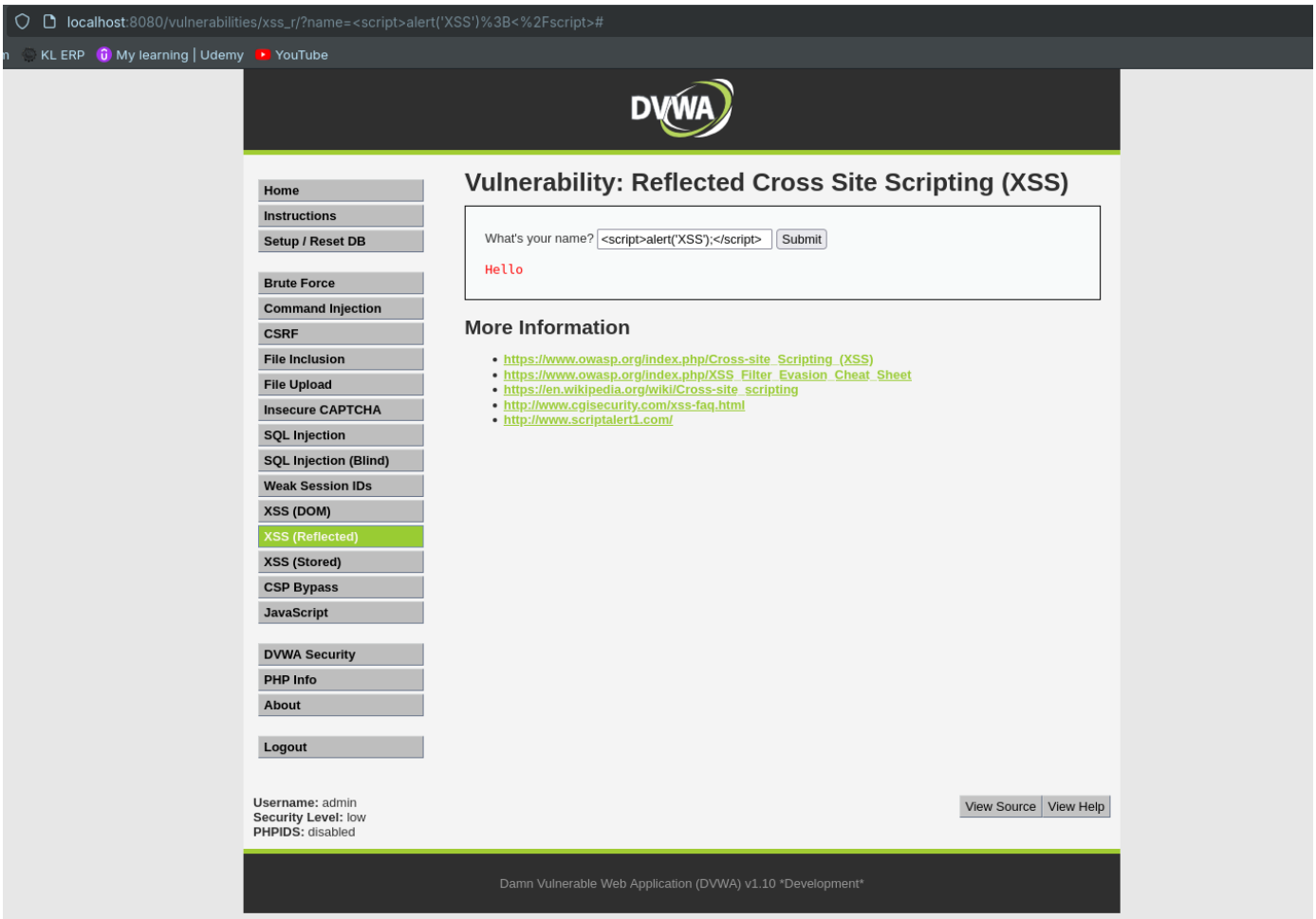
SQL Injection:

- The input `' OR '1'='1` is a common example of a SQL injection payload.
- By entering this payload in the username or password field, the attacker is attempting to manipulate the SQL query used for authentication.
- The condition `'1'='1'` always evaluates to true, effectively bypassing the login mechanism.
- As a result, the attacker gains unauthorized access to the application, bypassing the need for valid credentials.



Cross-Site Scripting (XSS)

- The payload `<script>alert('XSS');</script>` is an example of an XSS payload.
- By injecting this payload into an input field, the attacker aims to execute arbitrary JavaScript code on other users' browsers.
- When a victim user views the compromised content containing the injected payload, the JavaScript code `alert('XSS');` executes, displaying an alert box with the message "XSS".
- This showcases the ability to run unauthorized scripts on the victim's browser, which could be used for various malicious purposes.



Command Injection

- The payload `127.0.0.1; ls -la` is an example of a command injection payload.
- By injecting this payload into an input field vulnerable to command injection, the attacker attempts to execute arbitrary commands on the underlying system.
- In this example, the injected command is `ls -la`, which lists the files and directories in the current directory on the target system.
- If the command injection vulnerability is present and not properly mitigated, the attacker can execute unauthorized commands and potentially gain control over the system.