SCENARIO

The application a serialization-based session mechanism and is vulnerable to arbitrary object injection as a result. We will try to create and inject a malicious serialized object to exploit this vulnerability and use it to delete the morale.txt file.

**PROCEDURE**

1. Go the application and login using the given credentials to act as an user.
2. Navigate to the **My Account** page and inspect the session cookie because it contains a serialized PHP object.
3. Open the BurpSuite’s Target’s Sitemap, then open the **engagement tool** named **discover content** and go to the **Site Map**.
4. There we see a file named:

**/libs/CustomTemplate.php**

1. In the source code, notice the **CustomTemplate** class contains the **\_\_destruct() magic method**. This will invoke the **unlink()** method on the **lock\_file\_path** attribute, which will delete the file on this path.
2. We notice that the session cookie is encoded in Base64 and then in URL encoding.
3. We decode it in BurpSuite and we can see that it appears in the format below:

**O:4:"User":2:{s:8:"username";s:6:"wiener";s:12:"access\_token";s:32:"h3nnlst119zpa0gun0mr8o7zefbyjlny";}**

1. So, we inject the payload in place of the above value and send the request, even though we get an error but the lab is solved.
2. Now we deleted the file successfully without interacting with the system directly.

**PAYLOAD**

O:14:"CustomTemplate":1:{s:14:"lock\_file\_path";s:23:"/home/carlos/morale.txt";}

**REMEDIATION**

1. **Safe Deserialization:** Opt for libraries and tools that enable safe deserialization, ensuring that objects can't be tampered with or crafted maliciously during the deserialization process.
2. **Whitelisting:** If deserialization is essential, maintain a whitelist of classes that are safe to deserialize. By whitelisting classes, you prevent arbitrary class instantiation which can lead to unwanted side effects, such as invoking malicious methods.
3. **Limit Functionality:** Eliminate or limit the use of "magic methods" or destructors that get automatically triggered after deserialization. If these functions are necessary, they should be written in a way that they cannot cause any unintended side effects.
4. **Input Validation:** Before deserializing any object, ensure proper validation is performed on the serialized data. Any anomalies or unexpected patterns should be flagged and rejected.
5. **Avoid Exposing Sensitive Info in Serialized Objects:** Serialized objects should not contain sensitive application information, as this can become a potential vector for attack if the serialized object is manipulated.
6. **Encrypt and Sign Serialized Data:** If serialized data must be exposed to untrusted zones (like client-side), make sure to encrypt and sign it. Before deserializing it, verify the signature to ensure it wasn't tampered with.