XSS1 writeup

XSS

xss는 공격자가 웹 리소스에 악성 스크립트를 삽입해 이용자의 웹 브라우저에서 해당 스크립트를 실행할 수 있도록 하는 공격이다. 브라우저가 해석하는 javascript에 임의 스크립트를 삽입하여 본래 개발자의 의도 외의 동작 등이 실행되는 취약점을 이용한 공격이다.

분석

login(xss1.py)

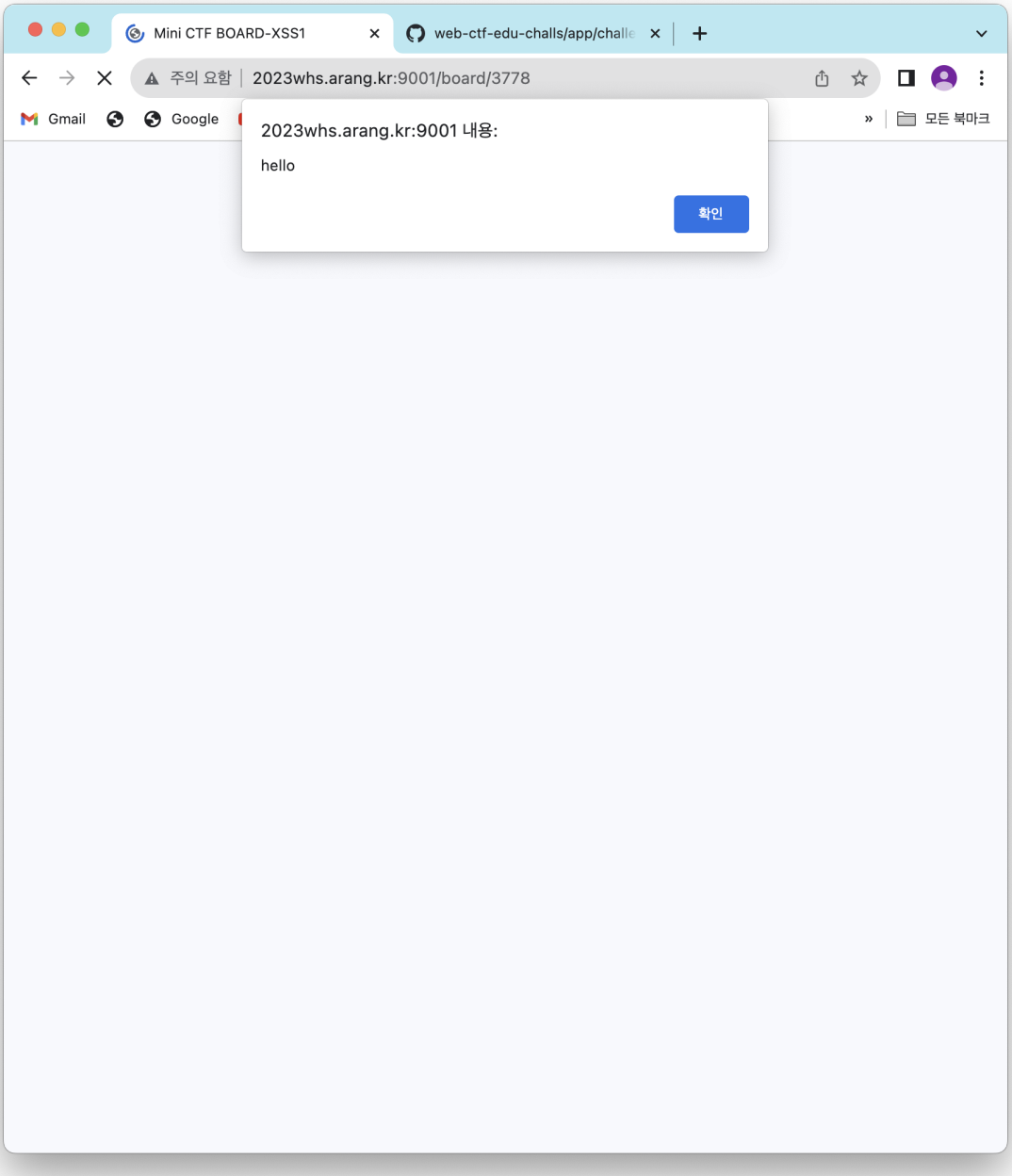
```
@app.route("/login", methods=["GET","POST"])
def login():
    if flask.request.method == "GET":
        if sessionCheck(loginCheck=True):
            return flask.redirect(flask.url_for("board"))

        return flask.render_template("login.html", msg="false")
    else:
        if sessionCheck():
            return flask.redirect(flask.url_for("board"))

        userid = flask.request.form["userid"]
        userpw = flask.request.form["userpw"]

        if userid in ids:
            if ids[userid] == userpw:
                flask.session["userid"] = userid
                flask.session["isLogin"] = True

                resp = flask.make_response(flask.redirect(flask.url_for("board")))
                resp.set_cookie('userid', flask.session["userid"])
                if userid == "admin":
                    resp.set_cookie('flag', os.getenv("xss1_flag"))
                return resp
            else:
                return flask.render_template("login.html", msg="login fail")
        else:
            return flask.render_template("login.html", msg="login fail")
```

로그인 함수를 보는데 flag가 눈에 띈다! 자세히 분석해봐야겠다.

먼저 get 메소드로 오는 요청은 셰션을 체크하고 로그인체크가 되어있다면 board 주소를 리다이렉트한다. post 메소드로 오는 요청도 마찬가지로 셰션 체크로 로그인 여부를 확인하고 POST로 답은 로그인 폼 데이터를 userid와 userpw로 받는다. userid는 중복체크를 하고 userid에 대한 셰션을 만들어준다. 응답에는 setcookie로 userid와 셰션값을 넣는데 이때 userid가 admin이면 쿠키의 셰션값이 flag가 되는 것을 알 수 있다. **즉 admin의 쿠키를 훔쳐야 한다!**

> 📌 목표 : admin 쿠키 탈취

write(xss1.py)


Mini CTF BOARD-XSS1
web-ctf-edu-challs/app/challe
주의 요함
2023whs.arang.kr:9001/board/3778
Gmail
Google
모든 북마크
2023whs.arang.kr:9001 내용:
hello
확인

```
            </thead>
            <tbody>
                <tr align="left">
                    <td>글쓴이
                    </td>
                    <td>
                    guest88
                    </td>
                </tr>
                <tr>
                    <td colspan="2" style="text-align: left;">
                        <!-- xss here! -->
                        <p><script>alert("hello")</script></p>

                    </td>
                </tr>
            </tbody>
        </table>

        <table class="table table-condensed">
            <thead>
                <tr>
```

게시글을 작성해보니 게시글에 xss 취약점이 발견되었다. 이 취약점을 통해 스크립트를 사용하여 정보를 빼내올 수 있을 것이라고 생각된다.

> 📌 취약점 : 게시글에 스크립트 작성시 실행

게시글에 <script>alert(document.cookie)</script>의 결과 나의 id인 guest88 나왔다.

report(xss1.py)

```
144
145     @app.route("/report", methods=["GET", "POST"])
146     def report():
147         if flask.request.method == "GET":
148             return '''
149             <form method="POST" action="/report">
150                 <input type="text" name="url" placeholder="input url..." style="width:25%; height: 7%;">
151                 <input type="submit" value="submit">
152             </form>
153             '''
154         elif flask.request.method == "POST":
155             url = flask.request.form['url']
156             requests.post(f"http://arang_client:9000/run", data=f"chal=xss1&url={url}", headers={"Content-Type":"app
157             return "<script>history.go(-1);</script>"
158
159
```

report 코드를 보니 GET 메소드 요청에 대해서는 현재페이지 리소스가 리턴되고 POST 메소드 요청이 들어오면 http:/arang_client:9000/run이라는 주소로 chal=xss1 정보와 url=사용자가 입력한 url 정보가 POST로 요청되고 사용자에게는 <script>history.go(-1);</script>가 리턴된다.

run(bot.py)

```python
@app.route("/run", methods=["POST"])
def run():
        url = flask.request.form['url']
        if 'chal' in flask.request.form:
                chal = flask.request.form['chal']
                print(f"[+] bot run arang_client:{chal} - {url}")
                doCrawl_server1(chal, url)
        else:
                print(url)
                doCrawl_server2(url)

        return "<script>history.go(-1);</script>"
```

위에서 보낸 post 요청은 chal(xss1으로 고정)이 폼데이터에 있으면 doCrawl_server1 함수가 실행된다.

```python
def doCrawl_server1(chal, url):
        crawler = Crawler()
        driver = crawler.driver
        try:
                crawler.req(f'http://arang_client:{server1_challs[chal]}/login')

                driver.find_element(By.ID,"userid").send_keys(ADMIN_ID)
                driver.find_element(By.ID,"userpw").send_keys(ADMIN_PASS)
                driver.find_element(By.ID,"submit-login").click()

                time.sleep(1)

                if crawler.req(url):
                        time.sleep(2)

                driver.quit()
        except:
                driver.quit()
                print(f"[x] error...")
                print(traceback.format_exc())
```

여기서는 http://arang_client:9001/login 페이지에 접속해 admin id와 pw를 적고 submit-login버튼을 눌러 로그인 한다. 그후 crawler.req(url)을 실행하는데

```
class Crawler:
        def __init__(self):
                options = webdriver.ChromeOptions()
                options.add_argument('--headless')
                options.add_argument('--no-sandbox')
                options.add_experimental_option("excludeSwitches", ["ena
                options.add_experimental_option("useAutomationExtension"

                self.driver = webdriver.Chrome(service=Service(ChromeDri
                self.driver.request_interceptor = interceptor
                self.driver.implicitly_wait(3)
                self.driver.execute_script("Object.defineProperty(naviga
                self.driver.execute_cdp_cmd('Page.addScriptToEvaluateOnN
                self.driver.set_page_load_timeout(3)

        def req(self, url):
                try:
                        print(f"[+] doing crawler req {url}")
                        self.driver.get(url)
                except:
                        print(f"[x] crawler driver req fail - {url}")
                        print(traceback.format_exc())
                        return False

                return self.driver
```

이는 url을 불러오는 것이다. 즉, arang_client:9001에는 이미 admin계정으로 로그인이 된 상태에서 url(사용자가 읽기를 요청한 게시글)을 불러오기 때문에 게시글에 악성스크립트를 통해 쿠키를 읽어오면 된다.

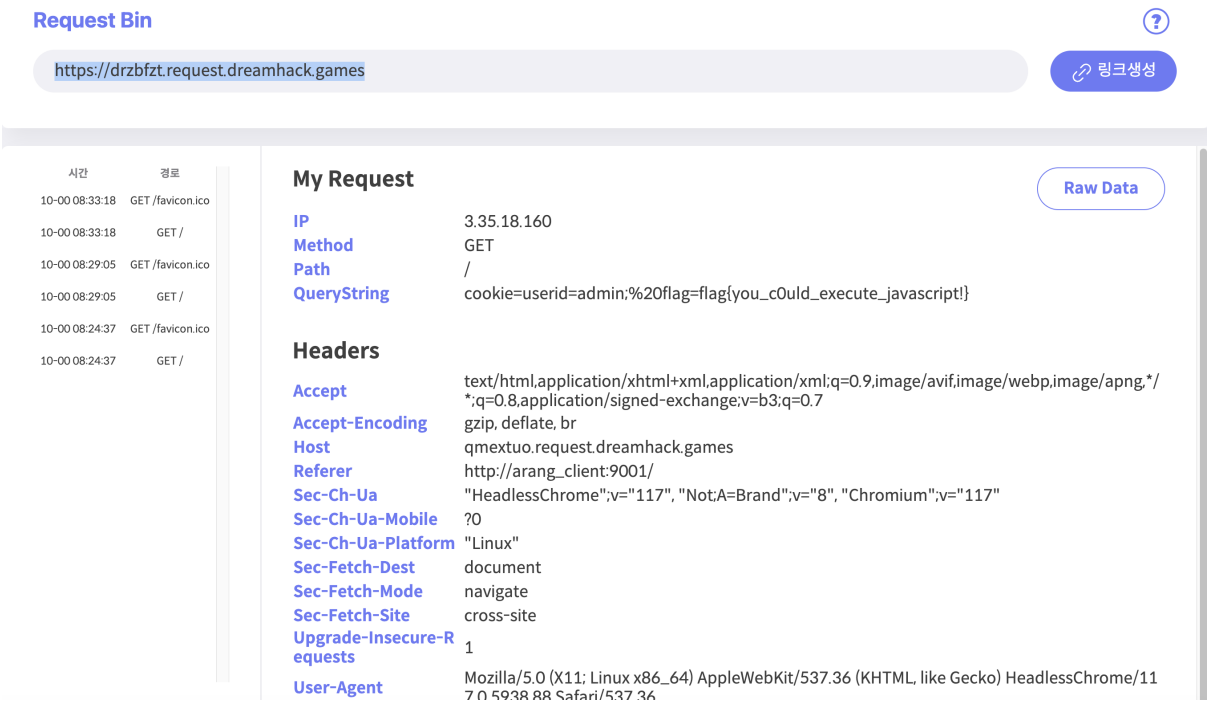
공격

1. 악성스크립트를 포함한 게시글을 작성한다.

주소를 request bin으로 생성한 임의 주소이며 이 스크립트를 admin이 실행하면 admin의 쿠키를 탈취할 수 있다.

2. 위의 게시글 주소로 관리자 읽기 권한 요청

해당 주소를 작성하고 submit을 누르면 먼저 arang_client:9000/run으로 post 리퀘가 가고 arang_client:9001/login에서 admin계정으로 로그인이 이루어진 뒤에 위의 주소로 접속한다. 이때 주의할 점은 위의 주소를 게시글의 게시글의 주소인 http://2023whs.arang.kr:9001/board/3793 으로 하면 위의 admin계정의 호스트와 달라 게시글 실행이 되지 않는다.



request bin에 admin의 쿠키값이 넘어온것을 확인할 수 있다.

대응방안

1. 입력값 필터링

데이터가 서버로 넘어가기 전에 필터링 하여 악성스크립트가 서버에 저장되지 않도록 하는 방법이 있다. check_xss() 함수를 생성하여 지정된 문자 또는 형식으로 입력되었는지 확인하고 정해진 규칙을 벗어난 입력값은 무효화 시키는 방법을 사용하면 된다.

2. 콘텐츠 보안 정책(Content Security Policy) 사용

csp는 공격자가 수행할 수 있는 작업을 제한하는 방법으로 이를 통해 xss를 완화할 수 있다. 스크립트 실행을 제한하거나, 특정 도메인에서만 리소스를 로드하도록 제한하여 xss를 대응할 수 있다.