웹 취약점 검사 시스템

팀 명 : 코웹
지도 교수 : 양환석 교수님
팀 장 : 박시원
팀 원 : 김난희
홍지우
윤해정
홍유진

2021. 10.
중부대학교 정보보호학과

목 차

1. 서론

1.1. 연구 배경

현대 사회에서 웹의 가치는 시간이 갈수록 높아지며 여러 방향으로 발전해 나가고 있다. 이는 동시에 웹을 이용한 해킹이 날이 갈수록 더욱 다양해지고 복잡해진다는 뜻이기도 하다. 단순히 정보가 노출되는 것부터 특수한 코드를 입력하여 권한이 없음에도 중요 데이터에 접근하는 일이 일어나고 있다. 따라서 OWASP TOP10(10대 웹 애플리케이션의 취약점)이라는 자료를 참고하여 웹 보안에 대한 연구를 진행하였다.

1.2. 연구 필요성

21세기 현재 세계는 많은 것이 인터넷으로 이어져 있어 지구상에서 일어나는 상당수의 일들이 웹에서 이뤄진다고 해도 과언이 아니다. 현재 디지털 전환으로 인해 수많은 기업들이 업무를 단순히 웹으로 운영하는 것이 아닌 정보통신기술(ICT)을 플랫폼으로 구축하여 기존 운영방식과 서비스 등을 혁신하였다. 정부도 대부분의 행정서비스를 웹으로 전환해 가는 추세이다. 또한 증권, 금융 업무 같이 신뢰가 중요한 업무부터 항공, 호텔 예약이나 쇼핑 같은 부분까지 대부분의 사회 활동이 물리적 공간에서 웹을 통하여 이루어 지고 있다. 디지털 전환이 계속 증가하는 동시에 웹 해킹 사고도 끊임없이 발생하고 있다. '경찰청 2020 사이버범죄 동향 분석 보고서'에 따르면 16년 153.075건에서 5년이 지난 지금 234.096건으로 대략 53%의 증가했다. 특히 웹은 누구나 쉽게 접근할 수 있게 되있을 뿐만 아니라 다른 시스템에 비해 해킹이 비교적 쉽고 효과가 높아 사이버 범죄의 집중적인 공격 대상이 되고 있다.

1.3. 연구 목적 및 주제 선정

웹은 수많은 취약점을 가지고 있고 수많은 공격방식이 존재한다. 이 수많은 취약점 중에 가장 대표적인 것은 OWASP TOP10이라는 자료를 통해 알 수 있다. OWASP TOP10은 웹 애플리케이션 취약점 중에서 빈도가 높고, 보안상 영향을 크게 줄 수 있는 것들 10가지를 선정한 것으로 항목으로는 XSS, SQL Injection,취약한 인증 등이 존재한다. 이를 참고하여 웹사이트에 존재하는 취약점을 진단하는 프로그램 개발을 목표로 하였다.

2. 관련 연구

2.1. OWASP TOP10

오픈소스 웹 애플리케이션 보안 프로젝트로 보안담당자들이 웹 어플리케이션 보안을 위해 작성되는 표준 문서이다. 2017년 기준으로 Injection (인젝션) 취약한 인증 민감한 데이터 노출, XML 외부 개체, 취약한 접근 통제 등이 있었으며 이처럼 웹에 관한 악성 파일 및 스크립트, 정보노출, 보안 취약점 등을 연구하며, 3~4년 주기로 빈도가 가장 많이 발생하고 보안상 영향을 크게 줄 수 있는 것들을 10가지 선정하여 발표한다.

2.2. Python

파이썬은 귀도 반 로섬이 개발한 고급 프로그래밍 언어로 플랫폼이 독립적이며 인터프리터식, 객체 지향적, 동적 타이핑 대화형 언어이다. 간단하게 인터프리터 기반의 객체 지향 프로그래밍 언어라고 할 수 있다. 파이썬 프로그램은 파이썬 쉘(shell)에서 대화모드로 코드를 테스트하면서 작성할 수 있는데, 파이썬 인터프리터가 사용자가 모르는 사이에 바이트 코드를 생성하여 해당 코드를 실행할 때 더 빠른 속도로 실행 가능하게 해주기 때문이다. 인터프린터식이라 실행될 행 단위마다 번역하고 큰 기억장치가 필요 없고 번역과정이 다른 방식에 비해 간단하다. 접착 언어라고 불리는 만큼 다른 프로그래밍 언어와 쉽게 결합하여 사용할 수 있다는 특징이 있다. 게다가 파이썬 코드는 각종 유닉스, 리눅스, 라즈베리 파이, 마이크로소프트 윈도우, 안드로이드 등의 다양한 환경에서 모두 실행할 수 있고 다른 언어들에 비해 비교적 다루기 쉽기 때문에 대중적인 인기를 끌고 있다.

2.3. JavaScript

미국의 넷스케이프 커뮤니케이셔즈사에서 개발한 언어다. HTML, CSS 와 함께 웹을 구성하는 프로그래밍 언어다. HTML은 웹의 기본적인 시각적 구조를, CSS는 시각적 구조를 꾸며주는 역할을 하고, Javascript는 웹 페이지가 동작하는 역할을 한다. Javascript 꾸준한 발전을 통해 브라우저를 넘어서 서버와 데스크탑 어플리케이션에서 사용되기도 한다.

2.4. MySQL

MySQL은 2016년 기준 80% 이상의 시장 점유율을 차지하고 있는 관계형 데이터베이스 관리 시스템이다. 오픈소스로 개발되며, GNU GPL(GNU General Public License)과 상업용 라이센스의 이중 라이센스로 관리되고 있다. MySQL은 데이터를 저장 및 엑세스 하는 스토리지 엔진과 SQL파서를 따로 분리하여 용도에 따라 스토리지 엔진을 선택할 수 있는 멀티 스토리지 엔진 방식을 채용하고 있다. 이중 가장 많이 사용하는 두 가지 엔진은 MyISAM과 InnoDB로, 마이아이삼은 기능이 복잡하지 않으나 데이터 조회를 위한 셀렉트 속도가 이노디비에 비해 빠르다. 하지만 데이터베이스 내에서 한 번에 수행되는 일련의 연산들에 대한 안전성을 보장하는 트랜잭션이 지원되지 않기 때문에, 데이터 무결성을 보장해야 하는 경우 이노디비가 사용된다. 데이터베이스를 관리하기 위한 GUI 기반 툴을 따로 내장하지 않기 때문에 일반적으로 명령줄 인터페이스를 사용하거나, MySQL 워크벤치와 같은 MySQL 프론트앤드 소프트웨어 및 웹 애플리케이션을 이용한다.

2.5 JSON

JSON(JavaScript Object Notation)은 "속성-값 쌍" 또는 "키-값 쌍"으로 이루어진 데이터 오브젝트를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷이며 동시에 비동기 브라우저/서버 통신을 위해, 넓게는 XML 을 대체하는 주요 데이터 포맷이다. 인터넷에서 자료를 주고 받을 때 그 자료를 표현할 수 있으며 자료의 종류에 큰 제한은 없다. 특히 컴퓨터 프로그램의 변수 값을 표현하는데 적합하다. 또한 JSON은 자바스크립트 언어로부터 파생되었기에 자바스크립트의 구문 형식을 따르지만 언어 독립형

데이터 포맷이다. 즉 프로그래밍 언어와 플랫폼에 독립적이므로, c, c++,자바, 자바스크립트 등 수 많은 프로그래밍 언어에서 구문분석 및 JSON 데이터 생성을 위한 코드로 사용된다.

2.6 NodeJS

자바스크립트에서 파생된 것으로 프론트 엔드 분야에서만 사용됐던 자바스크립트를 서버 단 기술까지 제어할 수 있게 되었다. 정확히는 이벤트 처리 I/O 프레임워크로 서버 환경에서 자바스크립트로 애플리케이션을 작성할 수 있도록 도와줄 수 있다. 논 블로킹 I/O와 단일 스레드 이벤트 루프를 통한 높은 처리 성능을 가지고 있어 특히 서버 쪽 네트워크 어플리케이션 개발에 유용하다. 또한 웹 서버를 세심하게 통제할 수 있는데 이는 내장 HTTP 서버 라이브러리를 가지고 있어 웹 서버에서 별도의 소프트웨어 없이 동작할 수 있기 때문이다.

3. 본론

3.1 시스템 구성

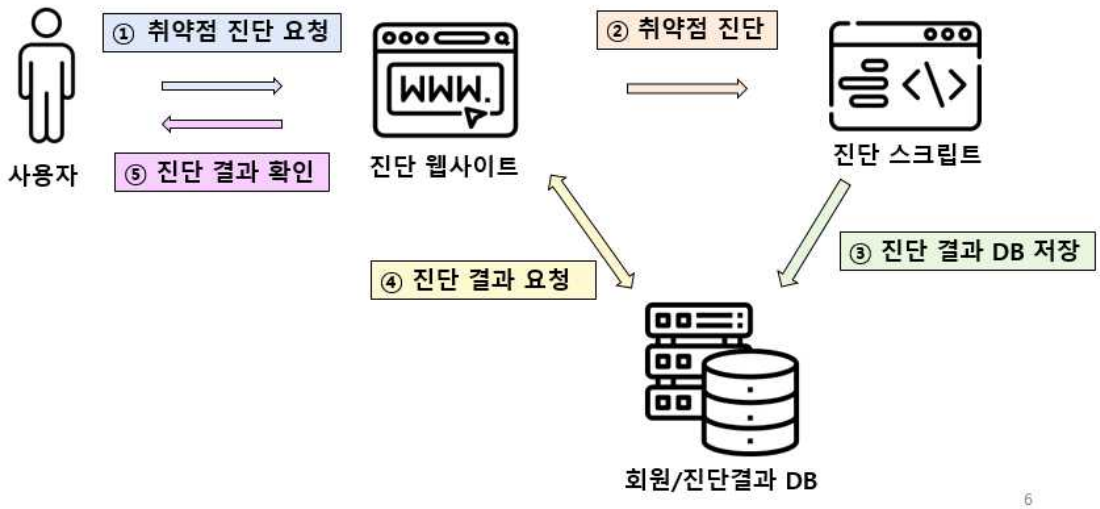


그림 1. 프로그램 구상도

3.2 웹 취약점 진단 스크립트

파이썬으로 웹 취약점을 진단하는 스크립트를 만들었다. 진단하는 항목은 총 9가지로 XSS, SQL Injection, CSRF, BruteForce, 쿠키 취약점, Redirect 취약점, 세션 고정 취약점, 민감한 데이터 취약점 그리고 평문 전송 취약점이다.

3.2.1 XSS

```
def XSS(url):
    target_url, method = getForm(url)

    insert_data = { "query" : "<script>alert('hi')</script>"}
    if method == "post":
        res=requests.post(target_url, data=insert_data).text
    else :
        res=requests.get(target_url, params=insert_data).text


    if  "<script>alert('hi')</script>" in res:
        result_list['XSS']='위험'
        result_list['XSS_info']=target_url
    else :
        result_list['XSS']='안전'
        result_list['XSS_info']=target_url
```

그림 2. XSS 스크립트

XSS는 Cross Site Scripting 의 줄임말로 사용자가 공격하려는 사이트에 악의적인 스크립트를 넣는 방법이다. 예를 들어 게시물에 제목이나 내용에 스크립트를 넣었을 경우 서버가 이를 필터링하지 못하고 게시물의 스크립트를 그대로 저장하면 다른 사용자가 그 게시물을 보았을 때 스크립트가 작동하는 것이다. 그림 2는 XSS에 대해 진단하는 스크립트이다. 'hi'라는 단어를 스크립트를 이용하여 넣었을 때 이를 필터링 하지 못하고 스크립트가 작동하여 'hi'라는 단어가 화면에 나타나면 현재 진단하는 웹사이트가 XSS에 대해 취약하다고 판단한다.

3.2.2 SQL Injection

```
def sqlInjection(url):
    target_url =url+"/doLogin"

    login_info ={
        "uid":"' or 1=1--+",
        "passw":"1234",
        "btnSubmit":"Login"
    }
    content=requests.post(target_url, data=login_info).text

    if "Welcome" in content:
        result_list['sqlin']='위험'
        result_list['sql_info']=target_url

    else:
        result_list['sqlin']='안전'
        result_list['sql_info']=target_url
```

그림 3. SQL Injection 스크립트

SQL Injection은 데이터베이스에 사용하는 언어인 SQL문을 활용하여 서버의 데이터베이스를 공격하는 방식을 말한다. 그림 3은 로그인에서 회원임을 검증하는 시스템을 무력화하는 방법이다. 일반적으로 로그인 폼에 아이디와 비밀번호를 입력하면 서버의 데이터베이스에서 입력한 정보와 일치하는 정보가 있는지 확인하고 맞다면 로그인에 성공한다. 하지만 그림 3과 같이 SQL문을 로그인 폼에 입력하면 회원가입이 되있지 않더라도 무조건 로그인에 성공하게 된다. 회원임을 조회하는 SQL문이 SELECT user FROM MEMBER WHERE uid='입력한 아이디' AND passw='입력한 비밀번호'; 라고 했을 때, 그림 3과 같이 입력하게 되면 uid="' or 1=1--"로 되어 1=1 부분이 true가 되어 코드가 전체적으로 false OR true AND flase 가 된다. 여기서 AND연산은 OR연산보다 우선순위가 높기 때문에 아이디와 비밀번호를 아무 값이나 입력하여도 ture가 되어 로그인이 성공한다. 위와 같은 공격방법이 성공하여 로그인이 되면 취약하다는 판단을 하도록 하였다.

3.2.3 CSRF

```
def CSRF(url):
    target_url =url+"/doLogin"
    session = requests.session()
    login_info ={
        "uid":"' or 1=1--+",
        "passw":"1234",
        "btnSubmit":"Login"
    }
    res=session.post(target_url, data=login_info)
    target_url = url+"/sendFeedback"

    send_info ={
        "cfile":"comments.txt",
        "name": "<body onload='csrf.submit();'><form name='csrf'action='http://altoromutual.com:8080/logout.jsp'</form>",
        "email_addr":"1@gmail.com",
        "subject":"2",
        "comments": "3",
        "submit":"Submit"
    }
```

그림 4. CSRF 스크립트

CSRF는 Cross Site Request Forgery의 약자로 XSS와 매우 유사하다. 큰 차이점은 XSS는 공격 대상이 클라이언트이나 CSRF는 서버라고 볼 수 있다. XSS의 경우 사용자가 서버측을 신뢰하여 무방비하게 웹사이트를 이용할 경우 일어나는 경우 대부분이다. 반면에 CSRF는 사용자를 이용하여 서버에 공격을 하는 방법이다. 그림 4는 서버측에 메일을 보내서 메일을 열람하면 자동으로 로그아웃 하도록 하였다. 이는 간단한 방법으로 더 나아가면 메일을 열람하면 자동으로 상품을 결제하거나 아니면 관리자가 메일을 봤을 때 관리자의 권한을 주도록 일반 계정을 바꾸는 등의 방법이 가능하다.

3.2.4 BruteForce

```
def bruteForce(url):
    target_url =url+"/doLogin"
    session = requests.session()

    password_list = open(os.getcwd()+"\password.txt", "r")
    password = password_list.readlines()

    for psword in password:
        passwordd = psword.strip()

        login_info ={
            "uid":"admin",
            "passw": passwordd,
            "btnSubmit":"Login"
        }

        content=session.post(target_url, data=login_info).text
```

그림 5. BruteForce 스크립트

브루트 포스는 조합 가능한 모든 문자열을 대입하는 방법으로 암호를 해독하는 방법이다. 웹에서는 주로 암호화가 되있는 시스템을 뚫기 위해 모든 경우의 수를 대입하는 방식으로 공격한다. 그림 5에서는 브루트 포스를 가장 많이 사용하는 비밀번호들을 텍스트 파일로 만들어 준비된 비밀번호들을 하나씩 대입해보는 방식으로 구현하였다.

3.2.5 쿠키 취약점

```
def cookie(url):
    target_url =url+"/doLogin"
    session = requests.session()

    login_info ={
    "uid":"' or 1=1--+",
    "passw":"1234",
    "btnSubmit":"Login"
    }
    session.post(target_url, data=login_info)
    cookiejar = session.cookies

    for cookie in cookiejar:
        if cookie.secure == False :
            result_cookie_secure= False
        else :
            result_cookie_secure= True

        if cookie.expires == None :
            result_cookie_expires= False
        else :
            result_cookie_expires= True
```

그림 6. Cookie 스크립트

쿠키는 주로 사용자의 인증 정보나 웹 사이트 설정 같은 부분을 저장하여 사용자가 웹 사이트를 이용하기 편리하도록 만들어준다. 쿠키에 사용자의 인증 정보가 담겨있는 만큼 쿠키의 정보가 공격자로 부터 탈취되지 않도록 방어하는 여러 옵션을 설정할 수 있다. 그림 6에서는 그 중 쿠키 값이 SSL 프로토콜을 통해 패킷 전송이 암호화 된 경우에만 전달되도록 하는 secure 값을 설정되어 있는지, 한번 만들어진 쿠키 값이 브라우저에 며칠이나 남아 있을 수 있는지를 설정할 수 있는 expires 값이 설정되어 있는지를 체크하여 취약함을 판단하도록 하였다.

3.2.6 Redirect 취약점

```
def redirect(url):
    target_url =url+"/doLogin"
    r = requests.get(target_url)
    try:
        session2 = r.headers['Strict-Transport-Security']
```

그림 7. redirect 스크립트

서버에서 HTTP 요청을 받고 HTTPS로 리다이렉트 하는 경우, 예를 들어 http://abc.com나 abc.com만 입력할 경우 서버측에서 302 Redirect를 이용하여 HTTPS로 전환시킬 수 있다. 하지만 이는 암호화되지 않고 평문으로 정보를 주고 받기 때문에 악의적인 다른 페이지로 리다이렉트 되도록 할 수 있는 취약점이 있다. 서버에서 HSTS를 설정하면 클라이언트가 HTTPS 요청을 하도록 강제하고 HTTP 요청은 시도되지 않도록 하여 이를 방지할 수 있다. 그림 7은 HSTS가 설정되어있는지 header에서 값을 확인하여 취약함을 판단하도록 하였다.

3.2.7 세션 고정 취약점

```
def session_fixation(url):
    target_url =url+"/doLogin"
    with requests.Session() as s:

        res = s.get(target_url, data=login_info)
        cookie1 = res.headers['Set-Cookie']

    with requests.Session() as s:
        res = s.get(target_url, data=login_info)

        cookie2 = res.headers['Set-Cookie']
```

그림 8. 세션 고정 스크립트

공격자가 세션 값을 미리 발급받고 공격대상에게 악성 스크립트를 활용하여 자신의 세

션 값으로 쿠키를 설정하도록 한다. 그리고 공격 대상이 로그인을 했을 때 세션 값이 고정되도록 설정되어있으면 공격자와 공격 대상이 같은 세션으로 접근하게 되어 둘의 정보가 공유되는 취약점이 생긴다. 그림 8에서는 진단하려는 사이트에 로그인을 여러 번 해서 각 세션 값이 같은지를 확인하여 취약함을 판단하도록 하였다.

3.2.8 민감한 데이터 노출

```
result_list['base64Vul'] = ''
for i in range(len(cookieValue)):
    data_bin=base64.b64decode(cookieValue[i])
    if isinstance(data_bin, str ):
        result_list['base64Vul'] = result_list['base64Vul'] + data_bin
    else :
        result_list['base64Vul'] = result_list['base64Vul'] + str(data_bin)
```

그림 9. 민감한 데이터 노출 스크립트

쿠키에는 사용자의 인증 정보 같은 민감한 정보들이 저장 된다. 이런 정보는 안전하게 암호화해서 전송되야 하지만 base64 같은 복호화 되기 쉬운 알고리즘으로 암호화를 하면 민감한 정보가 쉽게 노출된다. 그림 9는 base64로 암호화된 값을 복호화를 시도하여 복호화가 되는지 확인될 경우 취약하도록 판단하게 하였다.

3.2.9 평문 전송 취약점

```
def sniffing():
    sniff(lfilter=showPacket, timeout=6)

temp=0
def showPacket(packet):
    scapy.all.load_layer("http")
    target_url =url+"/doLogin"
    login_info ={
        "uid":"admin",
        "passw":"admin",
        "btnSubmit":"Login"
    }
    requests.post(target_url, data=login_info)
    pk = raw(packet)
    strpk = str(pk)
```

그림 10. 평문 전송 취약점

HTTP프로토콜을 사용할 경우 모든 데이터는 암호화 되지 않고 전송되기 때문에 이를 방지하기 위해 HTTPS프로토콜을 사용해야 한다. 평문 전송 여부는 URL의 프로토콜을 확인하여 간단하게 알 수 있다. 본 프로젝트에서는 스니핑을 통하여 중간에서 패킷을 감청해 HTTP프토토콜을 사용할 경우 어떻게 패킷에 직접 나타나게 되는지를 확인하여 취약

함을 판단하도록 하였다.

3.3 웹 사이트

그림 11. 홈페이지

홈페이지에 접속하면 바로 웹 취약점을 검사할 수 있는 검색창이 나온다. 검사를 진행하려면 로그인이 필요하도록 하여 먼저 회원가입을 진행해야 한다.

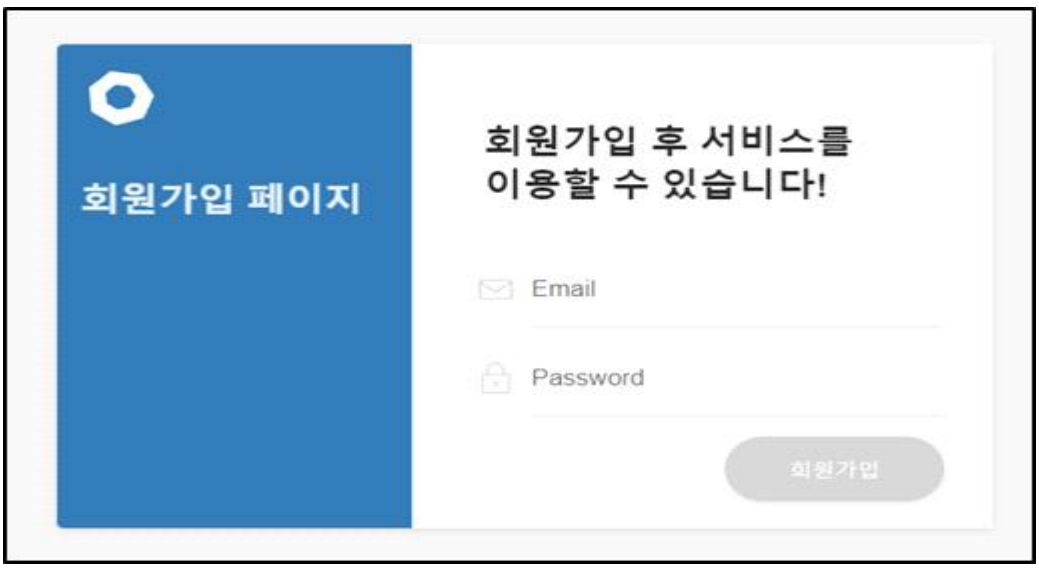


그림 12. 회원가입 페이지

진단 결과

#	항목	결과	정보
1	XSS	위험	http://altoromutual.com:8080/search.jsp
2	SQL INJECTION	위험	http://altoromutual.com:8080/doLogin
3	CSRF	위험	http://altoromutual.com:8080/sendFeedback
4	BruteForce	위험	http://altoromutual.com:8080/doLogin
5	Plaintext	위험	Https프로토콜필요
6	Cookie	위험	secure,expire 설정필요
7	Redircet	안전	
8	Session	위험	
9	Sensitive Data Exposure	b'800000~Corporate~-2.8079967124639E11|800001~Checking~2.3689348818021414E20|800002~Savings~-6.878684229873258E13|800003~Checking~5.1803111 E13|800005~Checking~4.666666669E10|800006~Savings~1.555682547E9|800007~Checking~786816.0|4539082039396288~Credit Card~-6.222232211008E10|4485 0Atxdbxc0x02xf4x00)xe3px82xe7xcf:xdfxa0xc2'	

그림 13. 검사 결과 페이지

검사를 진행하면 그림13과 같은 결과가 나온다.

그림 14. 진단 결과 목록 페이지

검사했던 기록들을 진단결과 페이지로 가면 볼 수 있다.

그림 15. 진단 항목 설명 페이지

진단한 항목에 대해 간단한 설명을 제공한다.

4. 결론

4.1 결론

웹은 인터넷을 이용한 서비스 중에서 접근성이 높다. 그리고 많은 정보가 웹을 통해 저장되기 때문에 새로운 해킹방법이 끊임없이 만들어져 항상 조심해야 한다. 하지만 웹은 갈수록 엄청나게 커지고 있기 때문에 그 많은 웹 사이트를 설계하는 모든 사람이 보안을 완벽하게 하지 않는다. 본 프로젝트를 통해서 보안에 대한 대책이 제대로 갖추어진 웹 사이트들의 취약점을 간편하게 확인할 수 있었다. 보안이 허술한 웹 사이트는 간단한 방법으로도 민감한 정보를 알아낼 수 있다. 따라서 웹을 사용할 때는 항상 보안에 경각심을 가지고 사용해야 한다.

4.2 기대효과

본 프로젝트를 통해 우리가 자주 사용하는 웹 사이트에 어떤 취약점들이 있는지 간편히 알 수 있다. 그리고 경각심이 없이 중요한 정보들을 웹에서 활용하던 사람들에게 이런 취약점들을 인지시킴으로써 좀 더 개인 정보 보호에 관심을 가지도록 할 수 있을 것이다.

5. 별첨

5.1 소스코드

webvul.py (웹 취약점 진단 스크립트)

```
import requests
from bs4 import BeautifulSoup as bs
import sys
import os
from requests.sessions import session
import json
import base64
from scapy.all import *
from scapy.layers.http import HTTPRequest
result_list = {}

def getForm(url):
    s = requests.session()
    soup=bs(s.get(url,verify = False).text, "html.parser")
    obj_form=soup.find_all("form")

    while(not obj_form):
        for x in soup.find_all("a"):
            href = x.attrs.get("href")
            soup=bs(requests.get(url+href).text, "html.parser")
            obj_form=soup.find_all("form")

    for x in obj_form:
        action = x.attrs.get("action")
        method = x.attrs.get("method")

    obj_input=soup.find_all("input")
    obj_textarea=soup.find_all("textarea")

    inputs = {}
    i = 0
    for x in obj_input:

        input_type = x.attrs.get("type")
        input_name = x.attrs.get("name")
        input_value = x.attrs.get("value")
        inputs["type{}".format(i)] = input_type
```

```
            inputs["name{}".format(i)] = input_name
            inputs["value{}".format(i)] = input_value
            i = i+1
        z = 0
        for x in obj_textarea:
            textarea_name = x.attrs.get("name")
            inputs["textarea_name{}".format(z)] = textarea_name
            z = z+1


        #딕셔너리 생성 : action, method, input
        form_info ={}
        form_info["action"] = action
        form_info["method"] = method

        target_url = url+form_info["action"]

        return target_url, form_info["method"]

def XSS(url):
        target_url, method = getForm(url)

        insert_data = { "query" : "<script>alert('hi')</script>"}
        if method == "post":
            res=requests.post(target_url, data=insert_data).text
        else :
            res=requests.get(target_url, params=insert_data).text


        if  "<script>alert('hi')</script>" in res:
            result_list['XSS']='위험'
            result_list['XSS_info']=target_url
        else :
            result_list['XSS']='안전'
            result_list['XSS_info']=target_url


def CSRF(url):
        target_url =url+"/doLogin"
        session = requests.session()
        login_info ={
            "uid":"' or 1=1--+",
```

```
            "passw":"1234",
            "btnSubmit":"Login"
        }
        res=session.post(target_url, data=login_info)
        target_url = url+"/sendFeedback"

        send_info ={
            "cfile":"comments.txt",
            "name": "<body onload='csrf.submit();'><form
name='csrf'action='http://altoromutual.com:8080/logout.jsp'</form>",
            "email_addr":"1@gmail.com",
            "subject":"2",
            "comments": "3",
            "submit":"Submit"
        }
        res=session.post(target_url, data=send_info)
        session = requests.session()
        res=requests.get(url)

        if "Sign In" in res.text:
            result_list['CSRF']='위험'
            result_list['CSRF_info']=target_url
        else:
            result_list['CSRF']='안전'
            result_list['CSRF_info']=target_url
def sqlInjection(url):
        target_url =url+"/doLogin"

        login_info ={
            "uid":"' or 1=1--+",
            "passw":"1234",
            "btnSubmit":"Login"
        }
        content=requests.post(target_url, data=login_info).text

        if "Welcome" in content:
            result_list['sqlin']='위험'
            result_list['sql_info']=target_url

        else:
            result_list['sqlin']='안전'
            result_list['sql_info']=target_url
```

```
def bruteForce(url):
    target_url =url+"/doLogin"
    session = requests.session()

    password_list = open(os.getcwd()+"₩password.txt", "r")
    password = password_list.readlines()

    for psword in password:
        passwordd = psword.strip()

        login_info ={
            "uid":"admin",
            "passw": passwordd,
            "btnSubmit":"Login"
        }

        content=session.post(target_url, data=login_info).text

        if "Welcome" in content:
            result_list['bruteForce']='위험'
            result_list['bruteForce_info']=target_url
            return
    result_list['bruteForce']='안전'
    result_list['bruteForce_info']=target_url

def cookie(url):
    target_url =url+"/doLogin"
    session = requests.session()

    login_info ={
    "uid":"' or 1=1--+",
    "passw":"1234",
    "btnSubmit":"Login"
    }
    session.post(target_url, data=login_info)
    cookiejar = session.cookies

    for cookie in cookiejar:
        if cookie.secure == False :
            result_cookie_secure= False
        else :
            result_cookie_secure= True
```

```
            if cookie.expires == None :
                result_cookie_expires= False
            else :
                result_cookie_expires= True

        if result_cookie_secure and result_cookie_expires :
            result_list["cookie"]='안전'
            result_list['cookie_info']='secure,expire 설정됨'

        else:
            result_list["cookie"]='위험'
            result_list['cookie_info']='secure,expire 설정필요'

def httpCheck(url):
        if "https" in url:
            try:
                res = requests.get(url,verify=False)
                if res.status_code == 200:
                    result_list['http']="안전"

            except:
                result_list["http"]="위험"
                result_list["http_info"]="Https프로토콜필요"

        else:
            result_list["http"]="위험"
            result_list["http_info"]="Https프로토콜필요"
def base64Vul(url):
        target_url =url+"/doLogin"
        session = requests.session()

        login_info ={
        "uid":"' or 1=1--+",
        "passw":"1234",
        "btnSubmit":"Login"
        }
        session.post(target_url, data=login_info)
        cookiejar = session.cookies

        cookieValue = []
        for cookie in cookiejar:
```

```
            cookieValue.append(cookie.value)

        result_list['base64Vul'] = ''
        for i in range(len(cookieValue)):
            data_bin=base64.b64decode(cookieValue[i])
            if isinstance(data_bin, str ):
                result_list['base64Vul'] = result_list['base64Vul'] + data_bin
            else :
                result_list['base64Vul'] = result_list['base64Vul'] + str(data_bin)

def sessionVul(url):
        target_url =url+"/doLogin"
        login_info ={
            "uid":"admin",
            "passw":"admin",
            "btnSubmit":"Login"
        }

        r = requests.get(target_url)
        try:
            session2 = r.headers['Strict-Transport-Security']
        except KeyError as session2:
            result_list['sessionVul'] = '위험'
        else:
            result_list['sessionVul'] = '안전'

def sniffing():
         sniff(lfilter=showPacket, timeout=6)

temp=0
def showPacket(packet):
        scapy.all.load_layer("http")
        target_url =url+"/doLogin"
        login_info ={
            "uid":"admin",
            "passw":"admin",
            "btnSubmit":"Login"
        }
        requests.post(target_url, data=login_info)
        pk = raw(packet)
        strpk = str(pk)
        global temp
```

```
    if 'uid=admin' in strpk:
     result_list['sniff'] = '위험'
     temp=1
    elif temp==1:
        result_list['sniff'] = '위험'
    else:
        result_list['sniff'] = '안전'

url = str(sys.argv[1])
sqlInjection(url)
CSRF(url)
cookie(url)
httpCheck(url)
bruteForce(url)
XSS(url)
base64Vul(url)
sessionVul(url)
sniffing()
json_result=json.dumps(result_list)
print(json_result)
```

main.js (웹 서버)

```
var express = require('express');
var app = express();
var path = require('path');
var url = require('url');
var template = require('./lib/template.js')
var bodyParser = require('body-parser');
var mysql = require('mysql');
var MySQLStore = require('express-mysql-session');
var flash = require('connect-flash');
var spawn = require('child_process').spawn;
var session = require('express-session');
const { Script } = require('vm');

var db = mysql.createConnection({
  host     : 'localhost',
  user     : 'root',
```

```
  password : 'sql1105',
  database : 'result'
})

var options= {
  host     : 'localhost',
  user     : 'root',
  password : 'sql1105',
  database : 'result'
}
var sessionStore = new MySQLStore(options);


db.connect();
app.set('veiw engine', 'ejs');
app.set('views','./views');
app.use(express.static(path.join(__dirname, '/public')));
app.use(bodyParser.urlencoded({ extended: false}));
app.use(flash());
app.use(session({
  secret: 'sessionKey',
  resave: false,
  saveUninitialized: true,
  store: sessionStore
}));

app.get('/', function(request, response){
  if(request.session.isLogined){
    response.render(__dirname+'/views/member.ejs');
  }
  else{
    response.render(__dirname+'/views/index.ejs');
  }

});

app.get('/profile', function(request, response){
  if(request.session.isLogined){
    response.render(__dirname+'/views/profile_logined.ejs');
  }
  else{
    response.render(__dirname+'/views/profile.ejs');
```

```
  }
});


app.get('/member_data', function(request, response){
  if(request.session.isLogined){
    db.query(`SELECT * FROM resultlist WHERE name='${request.session.userId}'`, function(err,
_url){
        if(err){throw err}
        var list = template.list(_url);
        var html = template.HTML(list);
        response.send(html);
      });
  }
  else{
    response.send('<script>alert("로그인이 필요합니다")</script>');
  }

});

app.get('/member_record/:userId', function(request, response){
  if(request.session.isLogined){
    db.query(`SELECT * FROM resultlist WHERE id='${request.params.userId}'`, function(err,
record){
        if(err){throw err;}
        var html = template.result(record[0].sqlin, record[0].sql_info, record[0].CSRF,
record[0].CSRF_info, record[0].cookie, record[0].cookie_info, record[0].http, record[0].http_info,
record[0].bruteForce, record[0].bruteForce_info, record[0].XSS, record[0].XSS_info,
record[0].base64Vul, record[0].sessionVul, record[0].sniff);
        response.send(html);
      });
    }
  else{
    response.send('<script>alert("로그인이 필요합니다")</script>');
  }

});

app.get('/register', function(request, response) {
  response.render(__dirname+'/views/register.ejs')
})
```

```
app.post('/register_process', function(request, response) {
  var name = request.body.name
  var pwd = request.body.pwd
  var sql_insert={name:name,pwd,pwd}
  db.query('SELECT name FROM member where name=?', [name],function(err,rows){
    if(rows.length){
      response.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
      response.write('<script>alert("중복되는 아이디입니다")</script>');
      response.write('<script>window.location.href="http://localhost:3000/register"</script>');
    } else {
      db.query('INSERT INTO member set?',sql_insert,function(err,rows){
        if(err) throw err;
      response.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
      response.write('<script>alert("회원가입을 완료했습니다")</script>');
      response.write('<script>window.location.href="http://localhost:3000/"</script>');
      })
    }
  });
});

app.get('/login', function(request, response) {
  response.render(__dirname+'/views/login.ejs')
})

app.post('/login_process', function(request, response) {
  var name = request.body.name
  var pwd = request.body.pwd
  var sql_insert={name:name,pwd,pwd}
  db.query('SELECT * FROM member where name=?', [name],function(err,rows){
    if(err){
      throw err
    }
    if(rows.length){
      if(rows[0].name === name){
        db.query('SELECT * FROM member where pwd=?', [pwd],function(err,rows){
          if(err){
            throw err
          }
          if(rows.length){
            request.session.userId=rows[0].name
            request.session.password=rows[0].pwd
            request.session.isLogined=true;
```

```
                request.session.save(function(){
                   response.redirect('/')
                })
              }else {
                response.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
                response.write('<script>alert("비밀번호가 틀립니다")</script>');

response.write('<script>window.location.href="http://localhost:3000/login"</script>');
              }
            })
          }
      } else {
          response.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
          response.write('<script>alert("아이디가 틀립니다")</script>');
          response.write('<script>window.location.href="http://localhost:3000/login"</script>');
    }
    });
});

app.get('/logout', function(request, response){
    delete request.session.userId;
    delete request.session.password;
    delete request.session.isLogined;
    request.session.save(function(){
      response.redirect('/')
    })
})

app.get('/result_process', function(request, response){
      var _url = request.url
      var queryData = url.parse(_url, true).query;
      var result = spawn('python', ['webvul.py', queryData.id]);
      result.stdout.on('data', function(data){                    //data는 python에서 json타입
으로 넘어옴
            var list = data.toString();
            var json_data=JSON.parse(list);

            db.query(
             `INSERT INTO resultlist  (name, URL, sqlin, sql_info, CSRF, CSRF_info, cookie,
cookie_info, http, http_info, bruteForce, bruteForce_info, XSS, XSS_info, base64Vul,
sessionVul, sniff) VALUES ('${request.session.userId}', '${queryData.id}',
'${json_data['sqlin']}','${json_data['sql_info']}', '${json_data['CSRF']}', '${json_data['CSRF_info']}',
```

```
'${json_data['cookie']}', '${json_data['cookie_info']}', '${json_data['http']}',
'${json_data['http_info']}', '${json_data['bruteForce']}', '${json_data['bruteForce_info']}',
'${json_data['XSS']}', '${json_data['XSS_info']}', "${json_data['base64Vul']}",
'${json_data['sessionVul']}', '${json_data['sniff']}')`

        , function(error, result){
            if(error){
                throw error;
            }
        })

        response.render('result.ejs', json_data);
    });
    result.stderr.on('data', function(data) {
        response.send(data.toString());
    });
});

app.get('/XSS', function(request, response){
  if(request.session.isLogined){
    response.render('XSS.ejs');
  }
  else{
    response.send('<script>alert("로그인이 필요합니다")</script>');
  }

});
app.get('/SQLINJECTION', function(request, response){
  if(request.session.isLogined){
    response.render('SQLINJECTION.ejs');
  }
  else{
    response.send('<script>alert("로그인이 필요합니다")</script>');
  }

});
app.get('/CSRF', function(request, response){
  if(request.session.isLogined){
    response.render('CSRF.ejs');
  }
  else{
    response.send('<script>alert("로그인이 필요합니다")</script>');
```

```
    }

});
app.get('/BRUTE', function(request, response){
    if(request.session.isLogined){
        response.render('BRUTE.ejs');
    }
    else{
        response.send('<script>alert("로그인이 필요합니다")</script>');
    }

});
app.get('/PLAIN', function(request, response){
    if(request.session.isLogined){
        response.render('PLAIN.ejs');
    }
    else{
        response.send('<script>alert("로그인이 필요합니다")</script>');
    }

});
app.get('/COOKIE', function(request, response){
    if(request.session.isLogined){
        response.render('COOKIE.ejs');
    }
    else{
        response.send('<script>alert("로그인이 필요합니다")</script>');
    }

});
app.get('/SNIFF', function(request, response){
    if(request.session.isLogined){
        response.render('SNIFF.ejs');
    }
    else{
        response.send('<script>alert("로그인이 필요합니다")</script>');
    }
    });
app.get('/SESSION', function(request, response){
    if(request.session.isLogined){
        response.render('SESSION.ejs');
    }
```

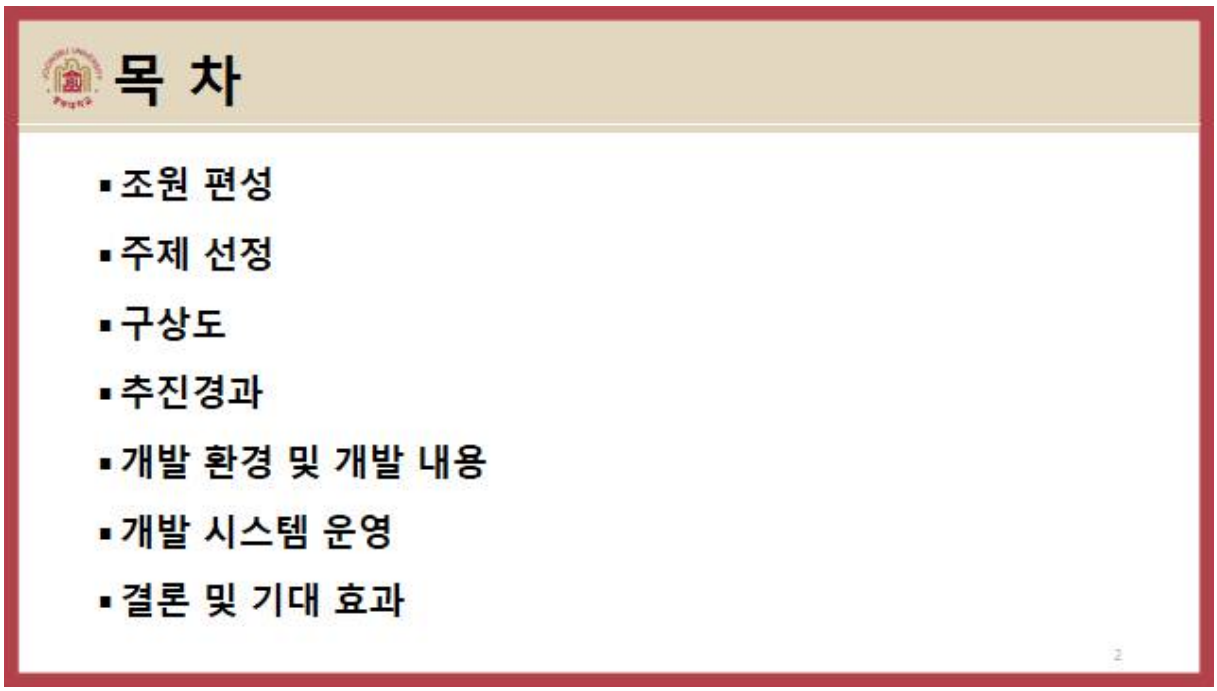
```
    else{
        response.send('<script>alert("로그인이 필요합니다")</script>');
    }
    });
app.get('/LOCAL', function(request, response){
    if(request.session.isLogined){
        response.render('LOCAL.ejs');
    }
    else{
        response.send('<script>alert("로그인이 필요합니다")</script>');
}

});
app.listen(3000, function() {

})
```

5.2 발표자료


웹 취약점 검사 시스템
2021. 10.
지도교수: 양환석 교수님
코 웹:
박시원
김난희
윤해정
홍유진
홍지우
JOONGBU UNIVERSITY
JBU
중부대학교
1


목 차
조원 편성
주제 선정
구상도
추진경과
개발 환경 및 개발 내용
개발 시스템 운영
결론 및 기대 효과
2

조원 편성

이 름	역 할
박시원(조장)	백엔드 개발 (진단 시스템) [총괄]
김난희	백엔드 개발 (진단 시스템), PPT 작성
홍지우	백엔드 개발 (진단 시스템), 보고서 작성
윤해정	프론트엔드 개발 (웹 사이트 개발)
홍유진	프론트엔드 개발 (웹 사이트 개발)

3

주제 선정(1/2)

웹을 통해 전달되는 민감한 정보가 기하급수적으로 증가

[단독] 국내 공공기관·기업 23곳 해킹 정보, 다크웹서 유통

국정원 확인, 보안조치 당부

박남은 기자

국가정보원이 마약·음란물 유통 등에 쓰이는 비밀 웹사이트인 '다크웹'에서, 국내 공공기관·기업 23곳의 서버(중앙 컴퓨터)에 어떻게 침투할 수 있는지 등 취약점을 상세히 정보가 유통되고 있는 것을 확인한 것으로 11일 전해졌다. 이를 활용하면 해커가 원격으로 국내 기관·기업의 서버에 몰래 침투해 각종 비밀 정보를 빼내거나 데이터를 인질로 삼아 거액을 요구할 수 있다는 것이다.

[유통가 해킹 전쟁]③ 재택근무로 랜섬웨어 공격 가능성 커져… "사물인터넷도 취약"

글로벌 랜섬웨어 피해액 6년만에 62배로 급증
기업 겨냥 해킹 공격 몸값, 3년만에 4배로 늘어
전문가 "모의훈련으로 경각심 높이는 게 최선"

윤희훈 기자

4

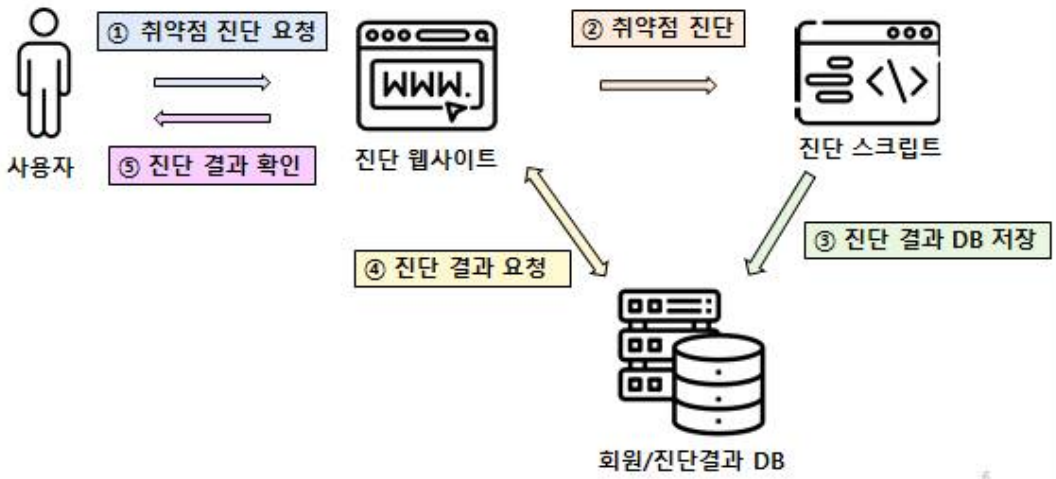
주제 선정(2/2)

주요 해킹 기법으로 꾸준히 정보 침해

간편한 취약점 검사 프로그램 개발

5

구상도



6

추진 경과

추진기간(2021) / 수행업무	3월	4월	5월	6월	7월	8월	9월	10월
자료조사 및 연구								
진단 스크립트 개발								
웹 페이지 개발								
DB구축 및 연결								
시스템 검증 및 보완								

개발 환경 및 개발 내용(1/5)

WEB Server

DB

Front-End

진단 스크립트

개발 환경 및 개발 내용(2/5)

진단 스크립트

XSS

```
def XSS(url):
    target_url, method = getform(url)

    insert_data = { "query" : "<script>alert('hi')</script>"}
    if method == "post":
        res=requests.post(target_url, data=insert_data).text
    else :
        res=requests.get(target_url, params=insert_data).text


    if  "<script>alert('hi')</script>" in res:
        result_list['XSS']='위험'
        result_list['XSS_info']=target_url
    else :
        result_list['XSS']='안전'
        result_list['XSS_info']=target_url
```

SQL Injection

```
def sqlInjection(url):
    target_url =url+"/doLogin"

    login_info ={
        "uid":"' or 1=1--+",
        "passw":"1234",
        "btnSubmit":"Login"
    }
    content=requests.post(target_url, data=login_info).text

    if "Welcome" in content:
        result_list['sqlin']='위험'
        result_list['sql_info']=target_url

    else:
        result_list['sqlin']='안전'
        result_list['sql_info']=target_url
```

9

개발 환경 및 개발 내용(3/5)

진단 스크립트

CSRF

```
def CSRF(url):
    target_url =url+'/doLogin'
    session = requests.session()
    login_info ={
        "uid":"' or 1=1--+",
        "passw":"1234",
        "btnSubmit":"Login"
    }
    res=session.post(target_url, data=login_info)
    target_url = url+'/sendFeedback'

    send_info ={
        "cfile":"comments.txt",
        "name": "<body onload='csrf.submit();'><form name='csrf' action='http://kitoroautual.com:8888/logout.jsp'></form>",
        "email_addr":"1@gmail.com",
        "subject":"1",
        "comments": "1",
        "submit":"submit"
    }
```

Brute Force

```
def bruteforce(url):
    target_url =url+"/doLogin"
    session = requests.session()

    password_list = open(os.getcwd()+"\password.txt", "r")
    password = password_list.readlines()

    for pswrd in password:
        passwordd = pswrd.strip()

        login_info ={
            "uid":"admin",
            "passw": passwordd,
            "btnSubmit":"Login"
        }

        content=session.post(target_url, data=login_info).text
```

10

개발 환경 및 개발 내용(4/5)

진단 스크립트

쿠키 취약점

```
def cookie(url):
    target_url =url+"/doLogin"
    session = requests.session()

    login_info ={
    "uid":"' or 1=1--+",
    "passw":"1234",
    "btnSubmit":"Login"
    }
    session.post(target_url, data=login_info)
    cookiejar = session.cookies

    for cookie in cookiejar:
        if cookie.secure == False :
            result_cookie_secure= False
        else :
            result_cookie_secure= True

        if cookie.expires == None :
            result_cookie_expires= False
        else :
            result_cookie_expires= True
```

세션 고정 취약점

```
def session_fixation(url):
    target_url =url+"/doLogin"
    with requests.Session() as s:

        res = s.get(target_url, data=login_info)
        cookie1 = res.headers['Set-Cookie']

    with requests.Session() as s:
        res = s.get(target_url, data=login_info)

        cookie2 = res.headers['Set-Cookie']
```

리다이렉트 취약점

```
def redirect(url):
    target_url =url+"/doLogin"
    r = requests.get(target_url)
    try:
        session2 = r.headers['Strict-Transport-Security']
```

11

개발 환경 및 개발 내용(5/5)

진단 스크립트

민감한 데이터 노출

```
result_list['base64Val'] = ''
for i in range(len(cookieValue)):
    data_bin=base64.b64decode(cookieValue[i])
    if isinstance(data_bin, str ):
        result_list['base64Val'] = result_list['base64Val'] + data_bin
    else :
        result_list['base64Val'] = result_list['base64Val'] + str(data_bin)
```

평문 전송 취약점

```
def sniffing():
    sniff(lfilter=showPacket, timeout=6)

temp=0
def showPacket(packet):
    scapy.all.load_layer("http")
    target_url =url+"/doLogin"
    login_info ={
        "uid":"admin",
        "passw":"admin",
        "btnSubmit":"Login"
    }
    requests.post(target_url, data=login_info)
    pk = raw(packet)
    strpk = str(pk)
```

12

개발 시스템 운영(1/4)

개발 시스템 운영(2/4)

개발 시스템 운영(3/4)

진단 목록

진단 결과

본인의 진단 목록에서 검사 결과 기록을 볼 수 있다.

15

개발 시스템 운영(4/4)

설명 페이지

각 항목을 클릭하면 해당 취약점에 대한 설명을 볼 수 있다.

16

결론 및 기대 효과

◆ 결론

- 주요 웹 취약점에 대하여 진단 결과 제공
- 진단 결과를 바탕으로 취약점 보완을 통해 안전한 웹 서비스 제공

◆ 기대효과

- 웹에 많은 보안 취약점들을 알고 정보보호에 경각심을 심어줌

17

JOONGBU UNIVERSITY
JBU
중부대학교

Q & A

감사합니다

18