2022년도

지금도 놓치고 있는 웹 취약점#1

본 문서는 ㈜파이오링크 침해대응센터에서 수행한 웹 침투 테스트의 실제 사례를 웹 공격 시나리오로 구성하여 악의적인 공격에 쉽게 활용되고 있는 위험성을 공유하고 각 고객사의 환경에 맞는 대응 방안을 마련하기 위해 작성되었습니다.

PIOLINK

파일 다운로드 취약점 → 소스코드 탈취 및 분석 → 악의적인 파일 업로드 → DB 접근 및 시스템 장악

File Download 기능이 있는 게시판에서 "../" 구문을 이용하면 상위 폴더 접근, 파일 다운로드 등 서버 및 서비스의 자원 탈취가 가능한 취약점이 일부 웹 서비스에서 여전히 발견되고 있다.

① robots.txt, view-source, Directory Scan 등을 통해 서버의 자원 경로 등의 정보(1차 정보)를 획득 할 수 있으며, 이렇게 확보한 1차 정보를 토대로 내부 자원(2차 정보)을 획득 할 수 있다.

② 획득한 2차 정보에 대한 코드 검사를 진행하여 include 정보 등 추가적인 정보(3차 정보)를 획득 할 수 있으며, 이를 통해 웹 개발 소스코드(4차 정보)를 추출 할 수 있다.

③ 추출한 웹 개발 소스코드에 대하여 소스코드 진단 등을 통해 SQL injection, File Upload, XSS, IDOR 등에 대한 취약점을 확인할 수 있다.

④ 확인 된 여러 취약점 중 시스템 장악에 가장 효과적인 WebShell(Oneline)을 업로드하면, 해당 WebShell을 통해서 시스템 제어(시스템 명령 실행) 단계에 이를 수 있다.

⑤ 이후 SQL Injection을 통해 Dump방식의 DB 추출이나 또는 소스코드에서 DB 계정 정보를 확보한다면 시스템 명령어 실행을 통한 DB로 직접 접근 등의 방법으로 DB 탈취도 가능하다.

①② 파일다운로드

[공격]

파일 다운로드 기능이 있는 게시판에서 Directory traversal 취약점을 악용 (/lib/download.php?file_name=PIOLINK&save_file=../../../../../../../etc/passwd&meta=free) 하면 서버의 소스코드를 탈취할 수 있다. 이후 Directory Scan, 웹 페이지 내 링크 주소 및 Directory traversal 취약점을 악용하여 이미 탈취한 소스코드의 include 정보 등을 통하여 내부 자원들의 경로 확보 및 추가적인 서버 자원을 탈취할 수 있다.

[탐지]

alert tcp any any -> any any (msg:"Path Traversal Prevention "; content:"GET |POST "; content:"../";)

alert tcp any any -> any any (msg:"Path Traversal Prevention "; content:"GET |POST "; content:"../../";)

[위협 요소 및 대응]

File Upload와 비교하여 상대적으로 취약하며, 소스코드 등의 탈취를 통해 이후 공격이 더욱 고도화되는 시발점이 될 수 있다. robots.txt, view-source 등을 통해서 기본적인 디렉토리 정보, 파일명 등을 확인할 수 있어 ../에 대한 단순 문자열 처리 뿐만 아니라 다운로드 기능과 관련된 전반적인 입력 값 검증 뿐 아니라 다운로드 디렉토리를 한정하는 등의 조치가 필요하다.

또한 해당 공격은 탐지 패턴만으로 탐지하기에는 그 케이스가 너무 다양하고 Directory Traversal 뿐만 아니라 FileDownload 관련 취약점을 통해서 복합적으로 발생할 수 있기에 해당 웹 서비스의 취약점 분석을 통해 필요한 패턴을 제작·적용해야 한다.

파일 다운로드 취약점 → 소스코드 탈취 및 분석 → 악의적인 파일 업로드 → DB 접근 및 시스템 장악

③ 소스코드 탈취 및 분석

[공격]

다운받은 소스코드 분석을 통해 File Download, File Upload, SQL Injection 등의 취약점을 확인할 수 있으며, 이러한 취약점을 악용하여 추가적인 서버의 자원 획득이 가능하다. 이러한 방법은 외부에서 침투 지점을 찾아내는 것보다 훨씬 정교하고 치명적인 공격이 가능하도록 할 수 있다.

[위협 요소 및 대응]

시큐어코딩 및 소스코드 진단 등을 통해서 취약점이 발생하지 않도록 미리 관리해야 하며, 불필요한 파일이나 경로에 대한 노출을 최소화해야 한다. 특히 오픈소스 사용이 늘어가고 있는 상황에서 해당 오픈소스의 버전 정보, 디렉토리 구조, 페이지 이름 등을 토대로 어떤 오픈소스가 사용되었는지를 공격자는 알 수 있기 때문에 오픈소스 사용 시에 이러한 점을 주의하여 커스터 마이징하거나 취약점에 관련된 정보를 주기적으로 확인하여 조치해야 한다.

④ 파일 업로드

[공격]

앞서 소스코드 탈취 및 분석을 통해 File Upload 취약점을 확인하였고 해당 취약점을 이용하여 WebShell을 업로드 할 수 있다. 해당 웹쉘을 통해 원격에서 명령을 실행 할 수 있었고, 추가적인 공격을 위해 좀 더 다양한 기능이 있는 WebShell을 업로드 하였고, 해당 WebShell을 통해 DB에 접근 및 조작, 추출을 진행하였다.

[탐지]

alert tcp any any -> any any (msg:"WebShell Detect 1"; content:"eval(gzinflate(base64_";)

alert tcp any any -> any any (msg:“WebShell Detect 2"; content:"GET |POST "; content:“eval(|passthru(|exec(|system(|shellexec("; nocase;)

alert tcp any any -> any any (msg:“WebShell Detect 3"; content:"GET |POST "; content:“subprocess.run|os.run|start-process"; nocase;)

alert tcp any any -> any any (msg:“Command Execute Detect 1”; content:“/bin/bash”; nocase; pcre:“/([0-9]{1,3}₩.){3}[0-9]{1,3}/”; pcre:“/[0-65535]/”)

[위협 요소 및 대응]

확장자를 제한하는 블랙리스트 방식으로 File Upload 검증 로직을 구현하고 있는데, 해당 방식은 실행 가능한 모든 형태의 확장자를 제한하지 않는 이상 취약한 부분이 생길 수 있어 파일 업로드 기능이 있는 게시판의 목적에 부합하게 화이트 리스트 방식으로 구현하는 것을 권장한다. 또한 의도하지 않은 파일이 업로드 되더라도 해당 파일의 경로나 이름을 추적할 수 없도록 변경하거나, 공격자가 URL을 통해 파일을 실행 할 수 없도록 웹 루트 폴더 외부에 저장되도록 구축하여야 한다.

파일 다운로드 취약점 → 소스코드 탈취 및 분석 → 악의적인 파일 업로드 → DB 접근 및 시스템 장악

⑤ DB 접근 및 시스템 장악

[공격]

업로드한 Webshell을 통해 DB에 직접 접근 및 데이터 조회가 가능하고, 추가적으로 관리 페이지 등 숨겨진 페이지/기능을 확인할 수 있다. 관리자 페이지 등에 접근 시 IP 등 접근권한을 체크하는 로직이 존재하지만, 이미 확보한 소소코드 내 SQL Injection 취약점을 이용하여 DB에 공격자의 IP를 추가함으로써 최종 관리자 페이지까지 장악이 가능하다.

[탐지]

alert tcp any any -> any any (msg:"SQL Injection bypass Prevention 1"; content:"GET |POST "; content:"₩/₩*"; content:"₩*₩/"; content: "select|union|insert"; flow:to_server)

alert tcp any any -> any any (msg:"SQL Injection bypass Prevention 2"; content:"GET |POST "; content: "select|union"; content:"information_schema|database()"; flow:to_server)

alert tcp any any -> any any (msg:"SQL Injection bypass Prevention 3"; content:"GET |POST "; content:"%27|₩'|%22|₩""; content"and|or|₩|₩||₩&₩&"; content:"=|like%"; flow:to_server)

alert tcp any any -> any any (msg:"SQL Injection bypass Prevention 4"; content:"GET |POST "; content: "select|union"; content:"@@version" flow:to_server)

[위협 요소 및 대응]

WebShell을 이용한 DB로의 직접 접속 시에도 관련 계정 정보가 필요하지만, 이러한 계정 정보가 웹 서버 내에 하드코딩 되어있을 경우 매우 취약한데 해당 정보에 대한 적절한 암호화를 구현하여 소스코드가 유출되더라도 DB접속 정보를 알 수 없게 구현해야 한다.

또한 이 시나리오에서는 DB 접근 및 장악 시에 WebShell 업로드를 통해 진행하였지만, 소스코드 분석을 통한 SQL Injection으로도 가능하기 때문에 SQL Injection과 관련된 룰들도 생성하여 대응해야 한다.

㈜ 파이오링크 I 02-2025-9665 I cert@piolink.com