

XSS 뽀개기 (2)

i-Keeper CERT 오나희

문제풀이 1

[3/6] Level 3: That sinking feeling...

Mission Description

As you've seen in the previous level, some common JS functions are *execution sinks* which means that they will cause the browser to execute any scripts that appear in their input. Sometimes this fact is hidden by higher-level APIs which use one of these functions under the hood.

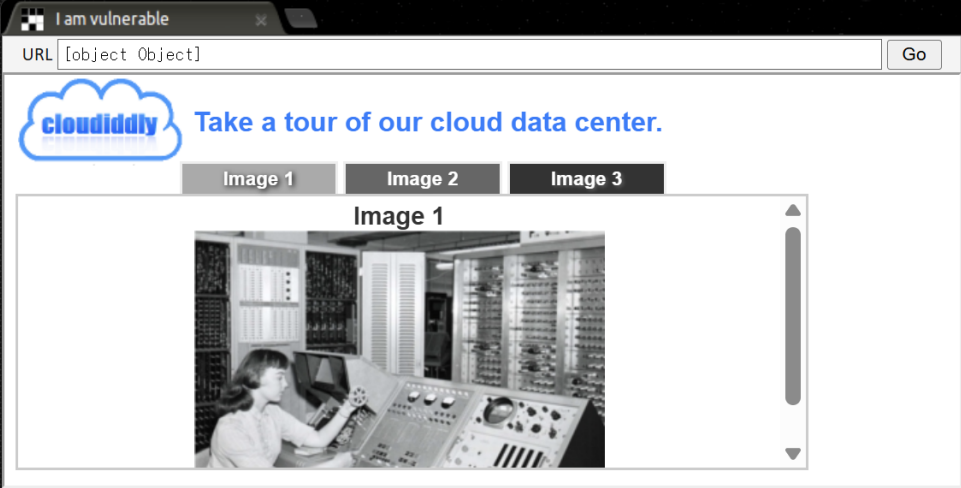
The application on this level is using one such hidden sink.

Mission Objective

As before, inject a script to pop up a JavaScript `alert()` in the app.

Since you can't enter your payload anywhere in the application, you will have to manually edit the address in the URL bar below.

Your Target



미션 설명

일부 자바스크립트 함수는 입력된 내용을 그대로 실행하는
*execution sink 역할을 한다
이 사실은 고수준 API 뒤에 숨겨져 있을 수 있으며
이번 단계의 애플리케이션도 그런 숨겨진 sink를 사용한다

미션 목표

애플리케이션에서 직접 입력할 수 있는 부분은 없으므로
URL 주소를 조작해 자바스크립트 `alert()`을 실행하는
스크립트를 삽입해야 한다

*execution sink : 데이터가 흘러 들어갔을 때 실제로 코드가 실행되는 지점

문제풀이 1

Level.py

```
class MainPage(webapp.RequestHandler):  
  
    def render_template(self, filename, context={}):  
        path = os.path.join(os.path.dirname(__file__), filename)  
        self.response.out.write(template.render(path, context))  
  
    def get(self):  
        self.render_template('index.html')  
  
application = webapp.WSGIApplication([ ('.*', MainPage), ], debug=False)
```

모든 요청을 MainPage가 처리하고 index.html만 내려준다
서버 반사(reflection)가 없어 Stored/Reflected XSS로 이어질 경로가 없음

쉽게 말하면 사용자로부터 받는 값 없이 홈페이지 자체를 보여주기만 함

문제풀이 1

Index.html

```
<!doctype html>
<html>
  <head>
    <!-- Internal game scripts/styles, mostly boring stuff -->
    <script src="/static/game-frame.js"></script>
    <link rel="stylesheet" href="/static/game-frame-styles.css" />

    <!-- Load jQuery -->
    <script
      src="//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js">
    </script>

    <script>
      function chooseTab(num) {
        // Dynamically load the appropriate image.
        var html = "Image " + parseInt(num) + "<br>";
        html += "<img src='/static/level3/cloud' + num + '.jpg' />";
        $('#tabContent').html(html);

        window.location.hash = num;

        // Select the current tab
        var tabs = document.querySelectorAll('.tab');
        for (var i = 0; i < tabs.length; i++) {
          if (tabs[i].id == "tab" + parseInt(num)) {
            tabs[i].className = "tab active";
          } else {
            tabs[i].className = "tab";
          }
        }
      }
    </script>
  </head>
  <body id="level3">
    <div id="header">
      
      <span>Take a tour of our cloud data center.</a>
    </div>

    <div class="tab" id="tab1" onclick="chooseTab('1')>Image 1</div>
    <div class="tab" id="tab2" onclick="chooseTab('2')>Image 2</div>
    <div class="tab" id="tab3" onclick="chooseTab('3')>Image 3</div>

    <div id="tabContent"> </div>
  </body>
</html>
```

```
// Tell parent we've changed the tab
top.postMessage(self.location.toString(), "*");
}

window.onload = function() {
  chooseTab(unescape(self.location.hash.substr(1)) || "1");
}

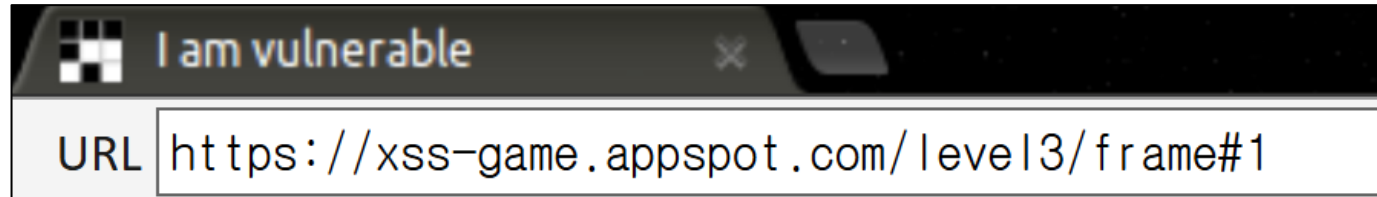
// Extra code so that we can communicate with the parent page
window.addEventListener("message", function(event){
  if (event.source == parent) {
    chooseTab(unescape(self.location.hash.substr(1)));
  }
}, false);
</script>

</head>
<body id="level3">
  <div id="header">
    
    <span>Take a tour of our cloud data center.</a>
  </div>

  <div class="tab" id="tab1" onclick="chooseTab('1')>Image 1</div>
  <div class="tab" id="tab2" onclick="chooseTab('2')>Image 2</div>
  <div class="tab" id="tab3" onclick="chooseTab('3')>Image 3</div>

  <div id="tabContent"> </div>
</body>
</html>
```

문제풀이 1



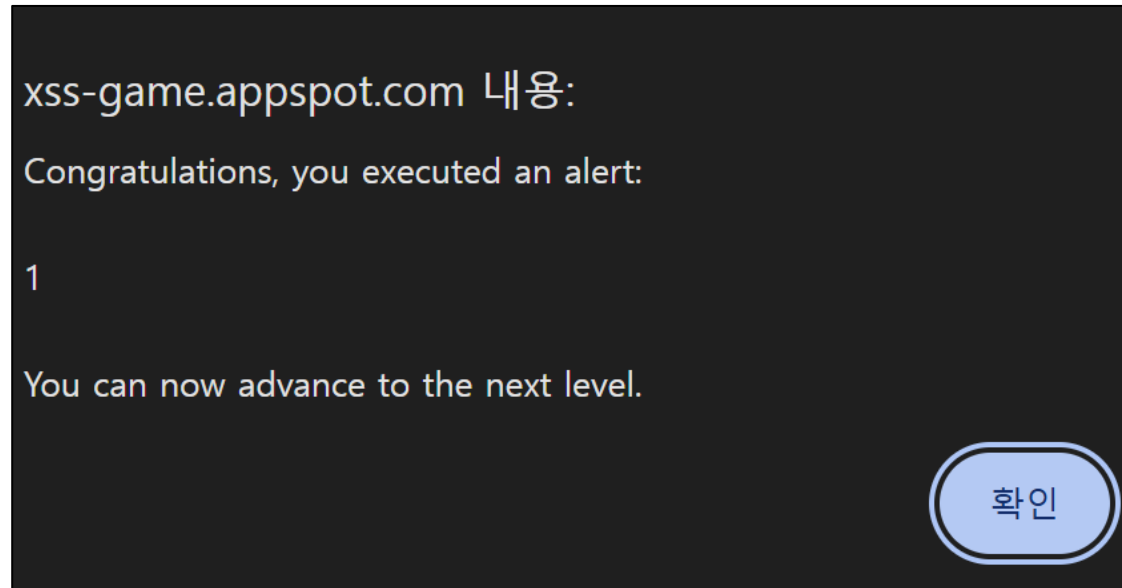
'onerror='alert(1)' x='
를 넣어보자

HTML 속성 탈출을 의도한 코드이다

원래 코드에서 만드는 HTML

을 넣었을 때
즉, src 속성을 끊어버리고 ('로) 뒤에 새로운 속성 추가

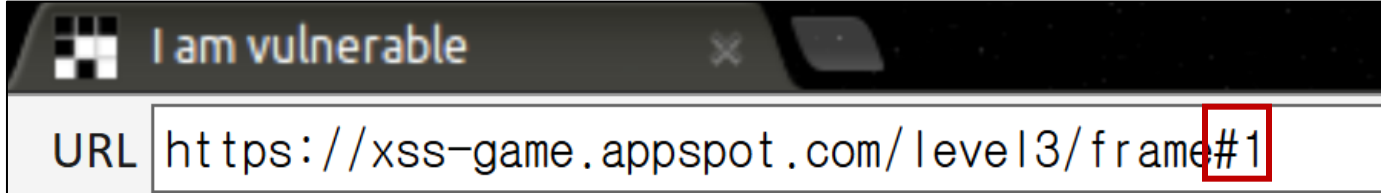
문제풀이 1



성공! 원리를 알아보자

DOM XSS?

type-0 XSS 로도 불리며 HTTP 응답 자체는 변화하지 않지만
클라이언트 측 자바스크립트가 DOM 환경을 조작하면서 의도하지 않은 방식으로 실행되는 공격
즉, 서버는 아무것도 바꾸지 않는데 **브라우저 안에서 이상하게 동작**한다는 것이 핵심



프래그먼트(해시, fragment identifier)

#1 은 *location.hash로 접근 가능한 값

서버로는 이 값이 전송되지 않고, 브라우저 안에서만 사용

*location.hash: URL에서 # 뒤의 문자열 서버엔 안 가고, 브라우저 자바스크립트에서만 읽고 쓸 수 있는 값

DOM XSS?

왜 URL에 #(fragment, 해시)라는 기능을 만들어줬을까?

1. 원래의 목적

문서 내부 이동(책갈피 기능)

즉, 긴 문서 안에서 바로 원하는 부분으로 점프할 수 있게 해주는 기능

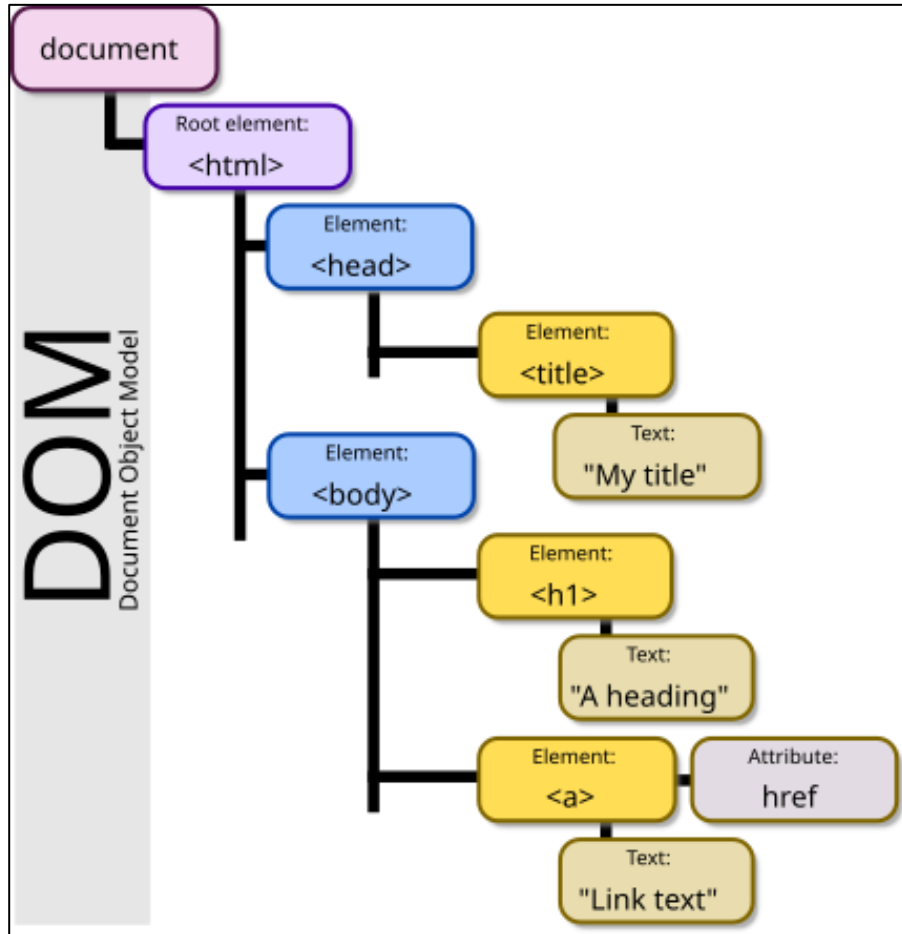
2. 현대의 목적

서버 트래픽 절약

리액트/앵귤러 같은 프레임워크는 서버에서 새로운 페이지를 안 불러오고

#login, #profile 같은 해시 값으로 페이지 전환을 흉내내게 된다

DOM XSS?



Document Object Model

브라우저가 HTML 문서를 읽어서 트리 구조의 객체로 만든 것
HTML 코드가 메모리 안에서 객체 구조로 표현된 것

자바스크립트는 DOM을 통해 페이지를 동적으로 바꿀 수 있다

동작 과정

1. ' onerror='alert(1)' x=' 값을 주소창에 넣는다
2. Js가 입력을 읽는다
3. DOM 조작

```
<img src='/static/level3/cloud1' onerror='alert(1)' x='.jpg' />
```

이미지 경로가 잘못되어 로드 실패 → onerror 이벤트가 발동 → alert(1) 실행

정리

Xss : **입력 값**을 실행 컨텍스트(HTML/JS/DOM)에 넣어 개발자의 의도와 다르게 발생시키는 것

유형	입력 경로	검증 부재 위치	실행되는 곳
Stored XSS	게시글·댓글 등 사용자 입력	서버(DB 저장·응답 시)	다른 사용자의 브라우저
Reflected XSS	URL 파라미터, 검색어	서버(응답 생성 시)	피해자 브라우저
DOM XSS	location.hash, URL 등 브라우저 값	클라이언트 자바스크립트	피해자 브라우저

location.hash : URL에서 # 뒤의 문자열 서버엔 안 가고, 브라우저 자바스크립트에서만 읽고 쓸 수 있는 값