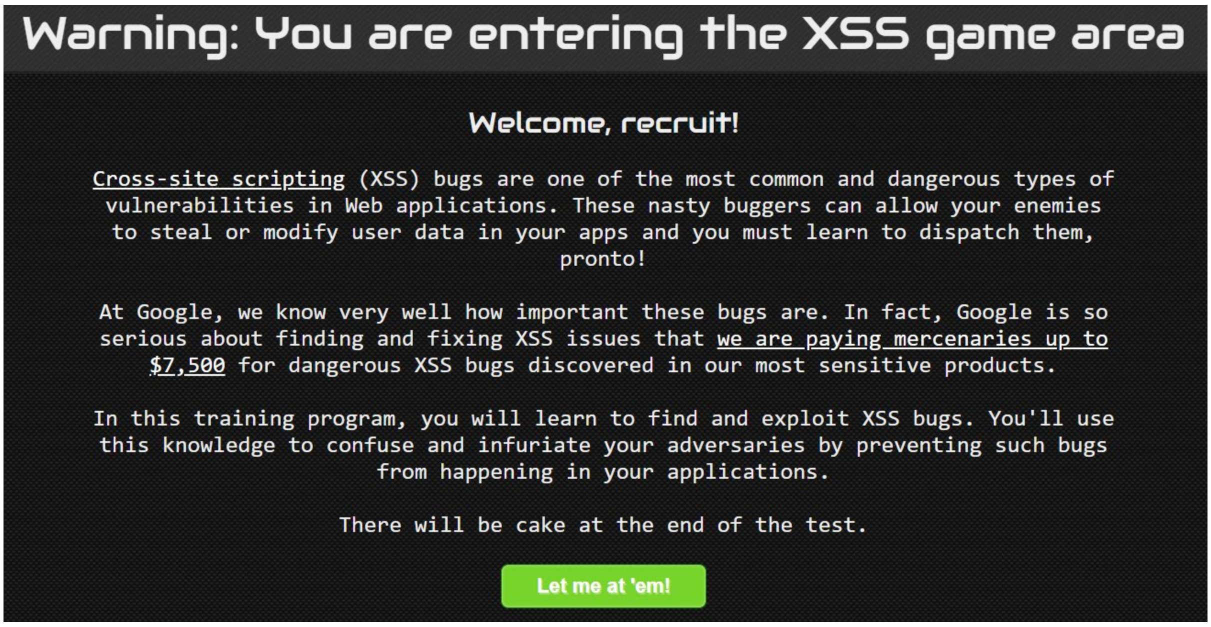
Google XSS-Game

This is a XSS ("Cross Site Scripting") introduction from google.

opening page:



Warning: You are entering the XSS game area

Welcome, recruit!

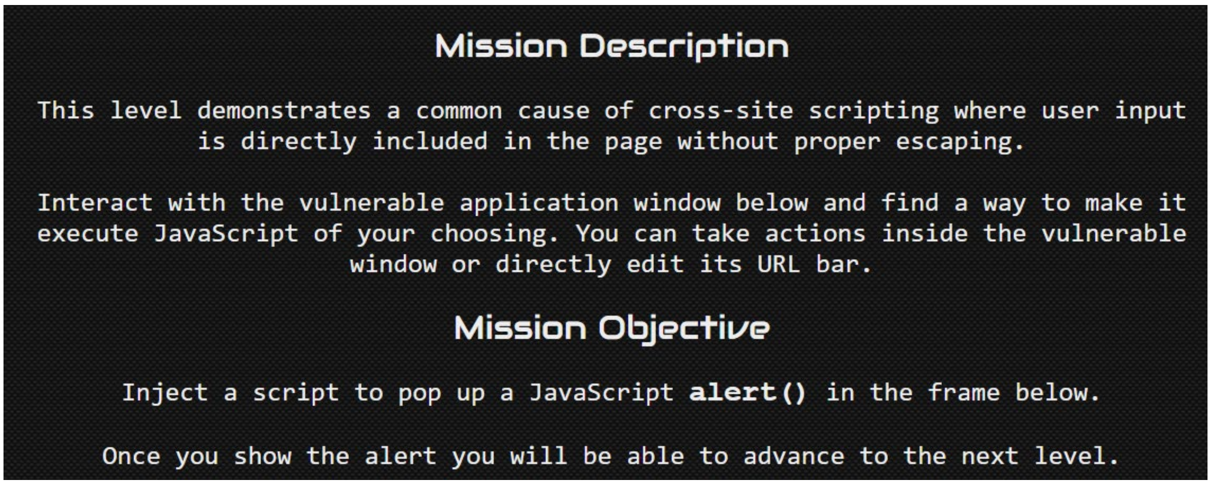
Cross-site scripting (XSS) bugs are one of the most common and dangerous types of vulnerabilities in Web applications. These nasty buggers can allow your enemies to steal or modify user data in your apps and you must learn to dispatch them, pronto!

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.

Level 1: Hello, world of XSS



Mission Description

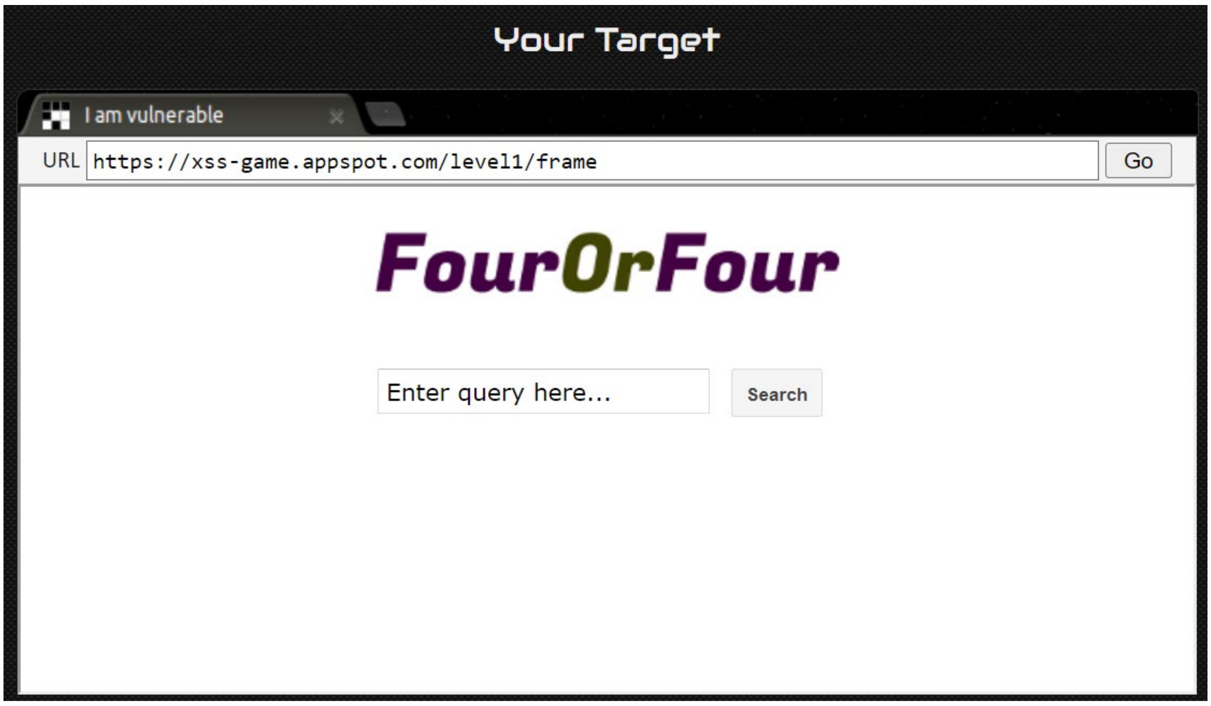
This level demonstrates a common cause of cross-site scripting where user input is directly included in the page without proper escaping.

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.

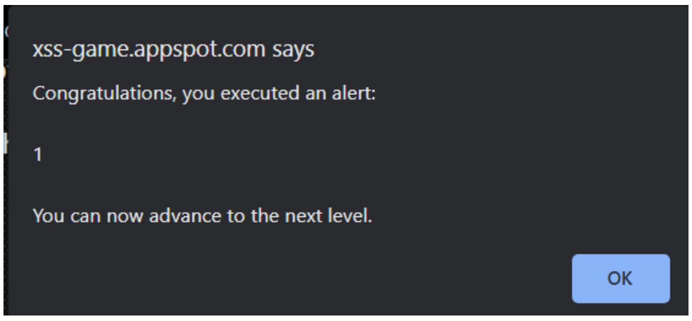
Once you show the alert you will be able to advance to the next level.



this is an easy one,

```
<script> alert(1) </script>
```

will pop up in your browser



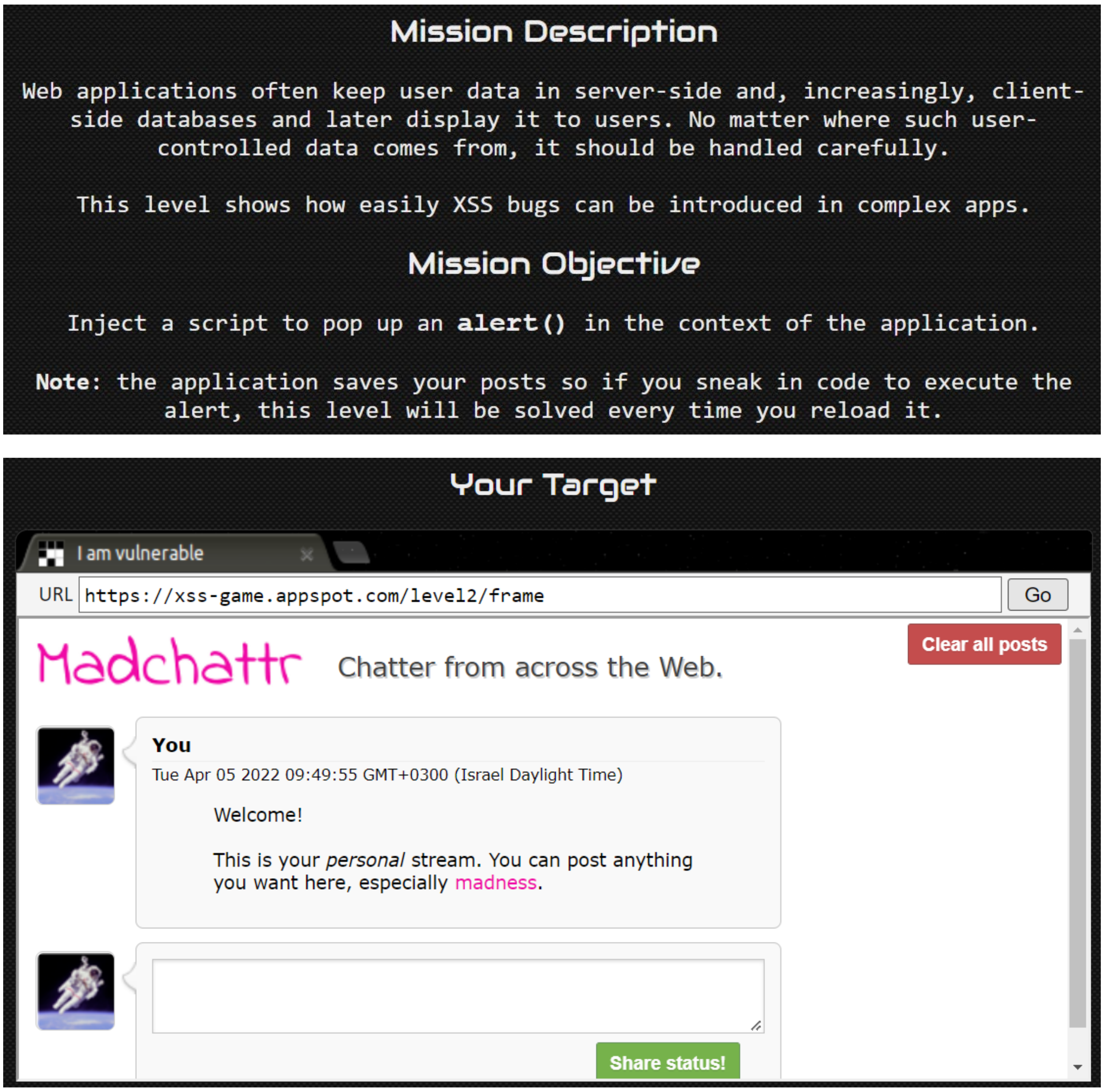
xss-game.appspot.com says

Congratulations, you executed an alert:

1

You can now advance to the next level.

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

I am vulnerable
URL https://xss-game.appspot.com/level2/frame
Go
Clear all posts
Madchattr
Chatter from across the Web.
Share status!

You
Tue Apr 05 2022 09:49:55 GMT+0300 (Israel Daylight Time)

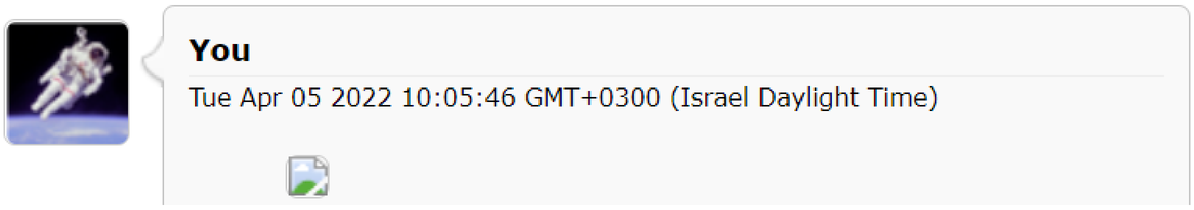
Welcome!

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

this level prevented us from using the tag <script> so we need to use other use reflected xss tags like <img> that is used to show images.

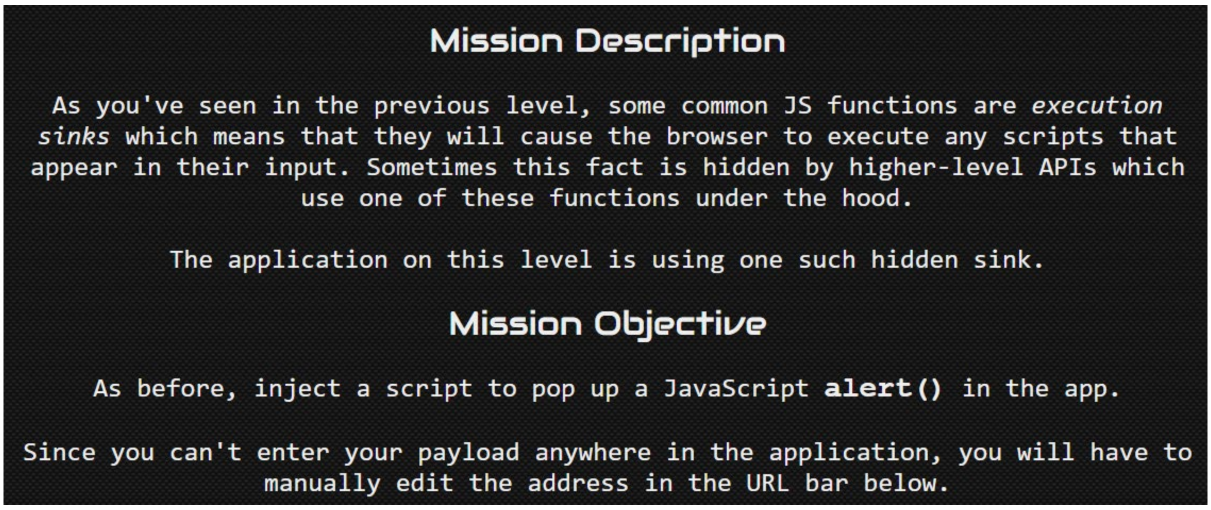
```
<img src=X onerror=javascript:alert(1)/img>
```

this will trigger the alert function and grant us the access to the next level.



You
Tue Apr 05 2022 10:05:46 GMT+0300 (Israel Daylight Time)

Level 3: That sinking feeling...


Mission Description

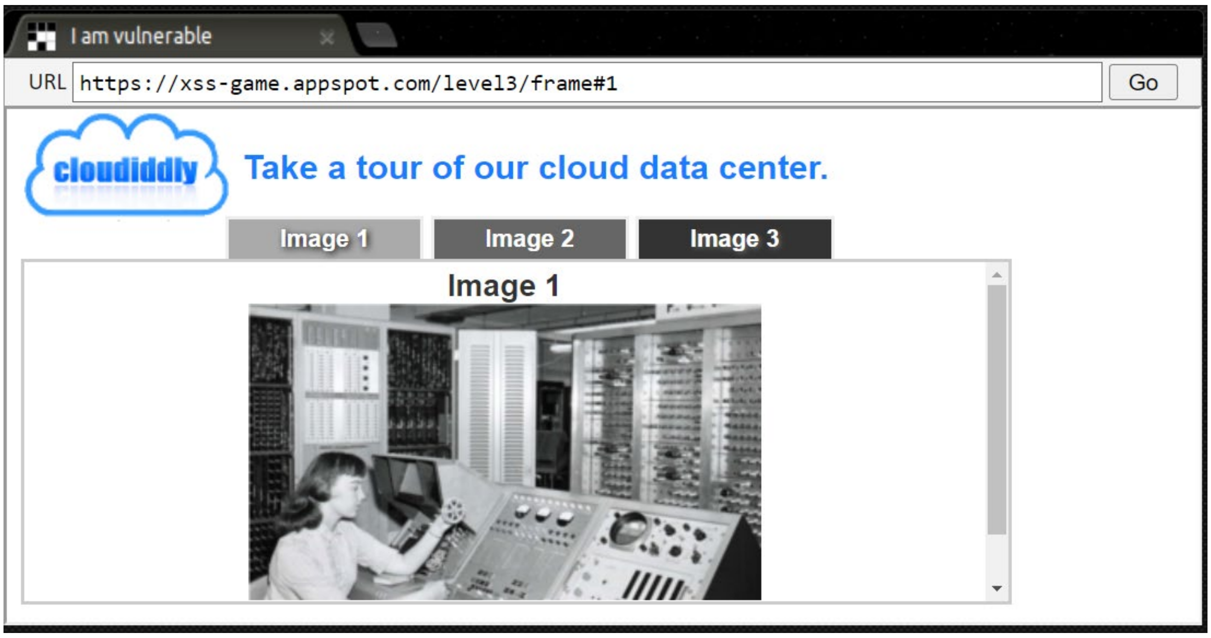
As you've seen in the previous level, some common JS functions are *execution sinks* which means that they will cause the browser to execute any scripts that appear in their input. Sometimes this fact is hidden by higher-level APIs which use one of these functions under the hood.

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.


I am vulnerable
URL https://xss-game.appspot.com/level3/frame#1
Go
cloudiddly
Take a tour of our cloud data center.
Image 1
Image 2
Image 3
Image 1

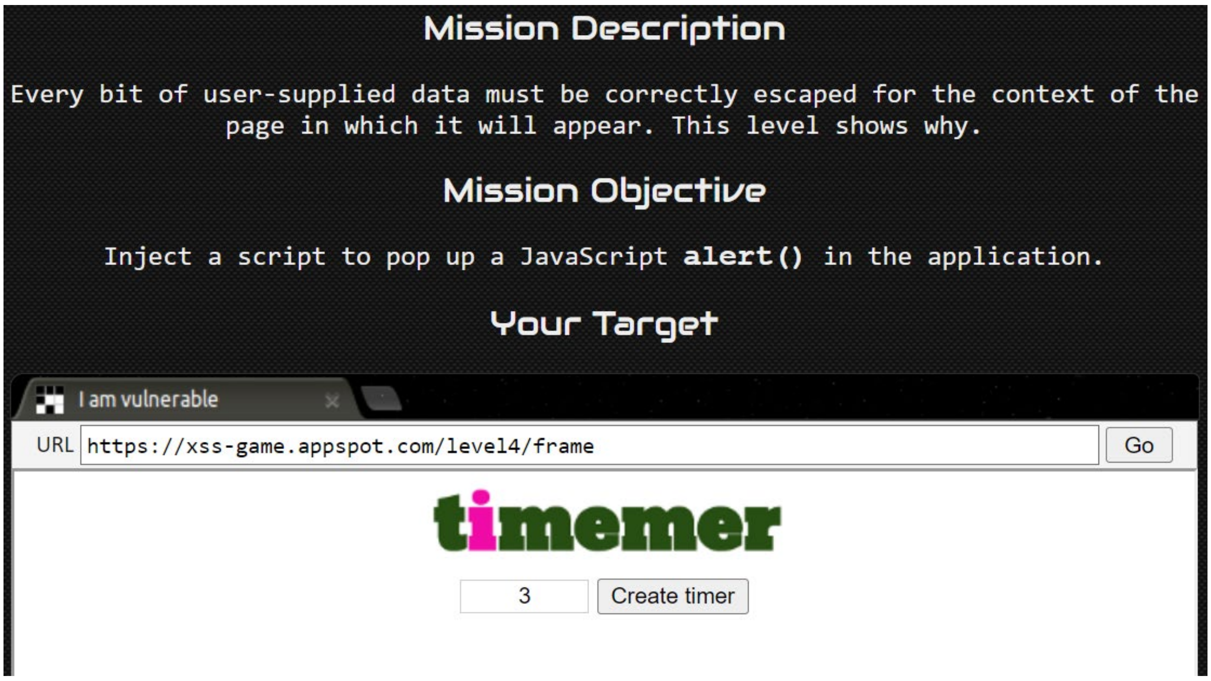
as the "mission objective" hinted, there is no place to inject the payload, but use the URL to our advantage.

changing the URL and adding the payload there:

```
https://xss-game.appspot.com/level3/frame#0' onerror='alert(1)'
```

the server is getting the input from the browser after the # and as we can see we injected an apostrophe ('), that triggered the error and injected the payload alert function.

Level 4: Context matters


Mission Description

Every bit of user-supplied data must be correctly escaped for the context of the page in which it will appear. This level shows why.

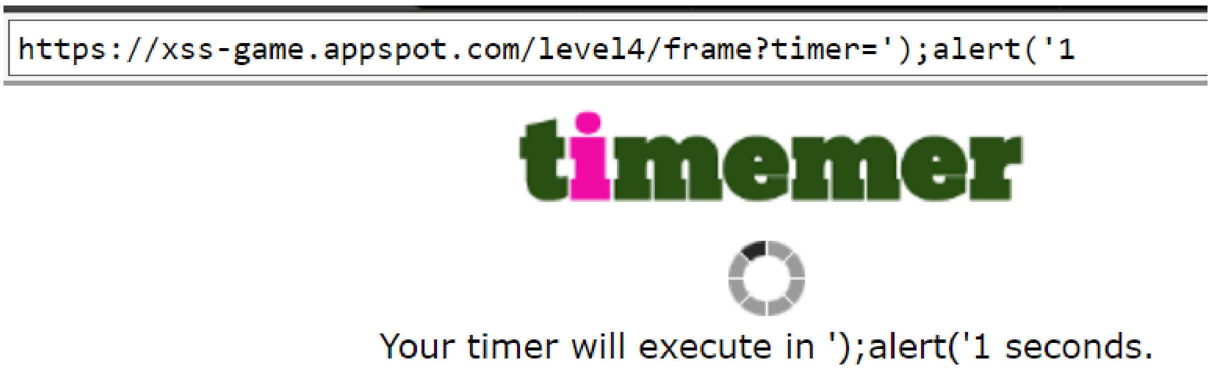
Mission Objective

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

Your Target

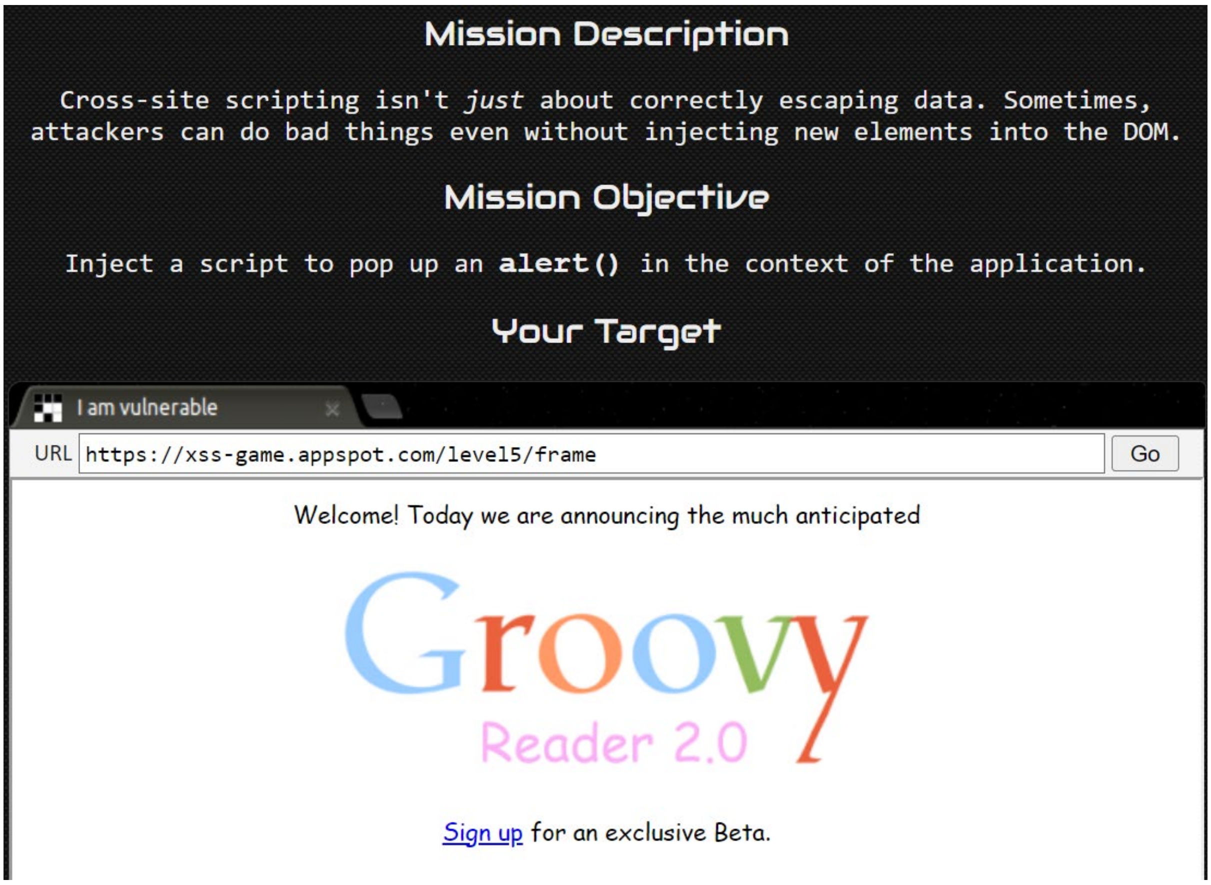
in this one we need to use the apostrophe again and inject the site

```
');alert('1
```


```
https://xss-game.appspot.com/level4/frame?timer=');alert('1
```

Your timer will execute in ');alert('1 seconds.

Level 5: Breaking protocol


Mission Description

Cross-site scripting isn't *just* about correctly escaping data. Sometimes, attackers can do bad things even without injecting new elements into the DOM.

Mission Objective

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

Your Target

I am vulnerable

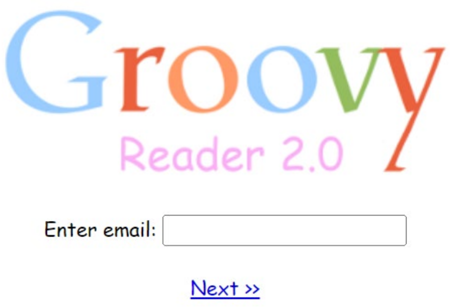
URL https://xss-game.appspot.com/level5/frame Go

Welcome! Today we are announcing the much anticipated

Groovy Reader 2.0

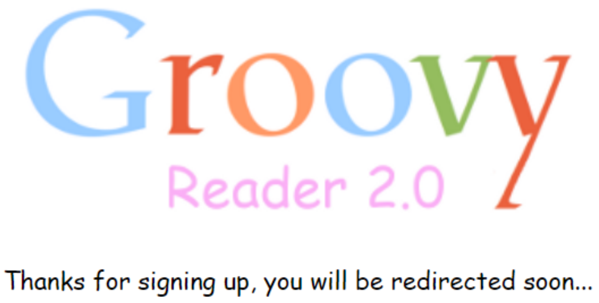
Sign up for an exclusive Beta.

clicking on the sign up leading to a fake email entering page (you can and should leave it blank, don't use your real email here)


Groovy Reader 2.0

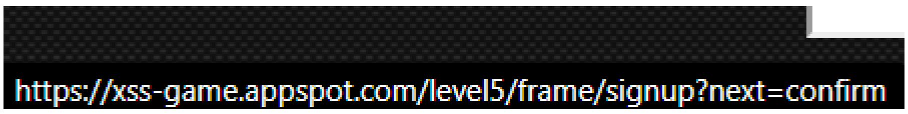
Enter email:

Next »


Groovy Reader 2.0

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

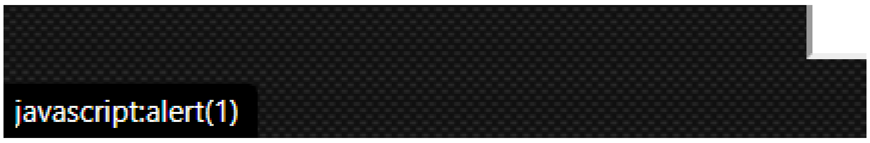
when we hover on the sign up button we can see the following:


https://xss-game.appspot.com/level5/frame/signup?next=confirm

if we change the URL and inject it with the XSS we can manipulate the site and use javascript here.

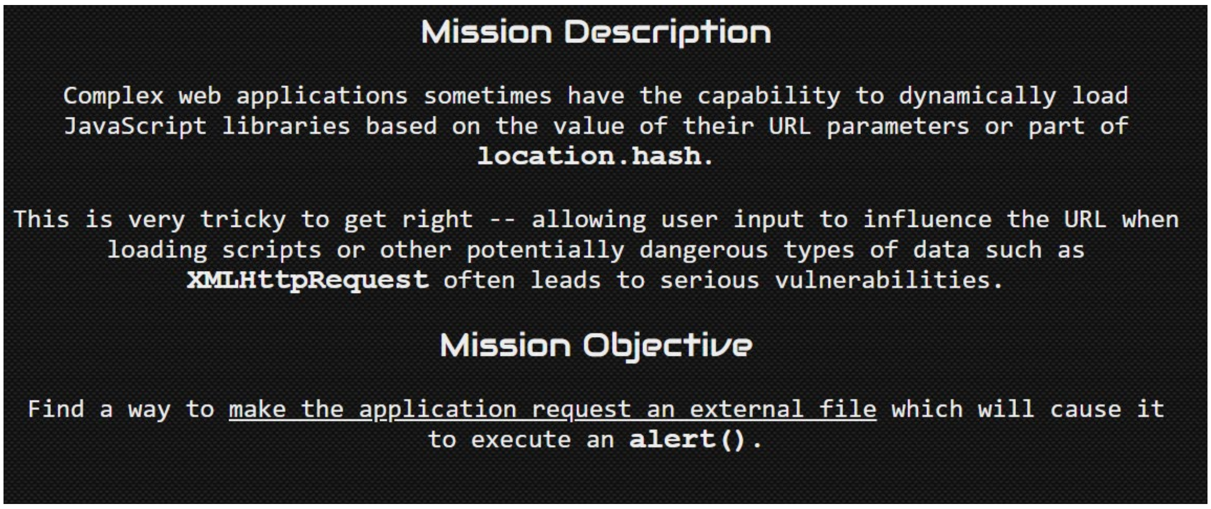
```
https://xss-game.appspot.com/level5/frame/signup?next=javascript:alert(1)
```

hovering over the "next" now will show us the injection



```
javascript:alert(1)
```

Level 6: Follow the 🐇



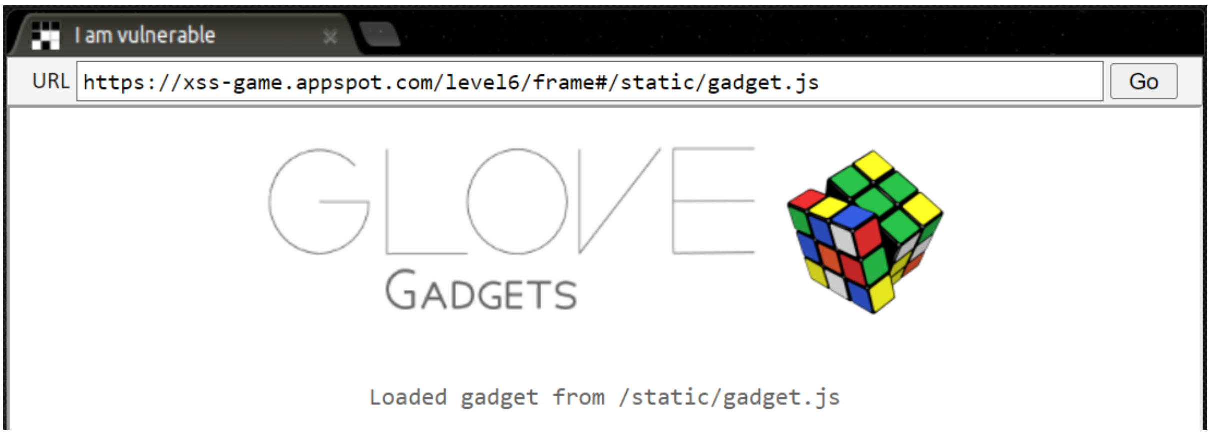
Mission Description

Complex web applications sometimes have the capability to dynamically load JavaScript libraries based on the value of their URL parameters or part of **location.hash**.

This is very tricky to get right -- allowing user input to influence the URL when loading scripts or other potentially dangerous types of data such as **XMLHttpRequest** often leads to serious vulnerabilities.

Mission Objective

Find a way to make the application request an external file which will cause it to execute an **alert()**.



Loaded gadget from /static/gadget.js

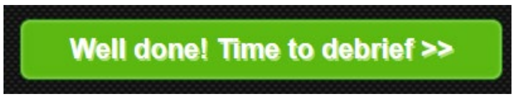
in this one we had to figure out how to host our own javascript code or find a code online.

we found out google is hosting a javascript api and we could use it.

the following will trigger the alert function:

```
https://xss-game.appspot.com/level6/frame#//www.google.com/jsapi?callback=alert
```

and now instead of the usual green botton we have:



You have successfully completed the game!