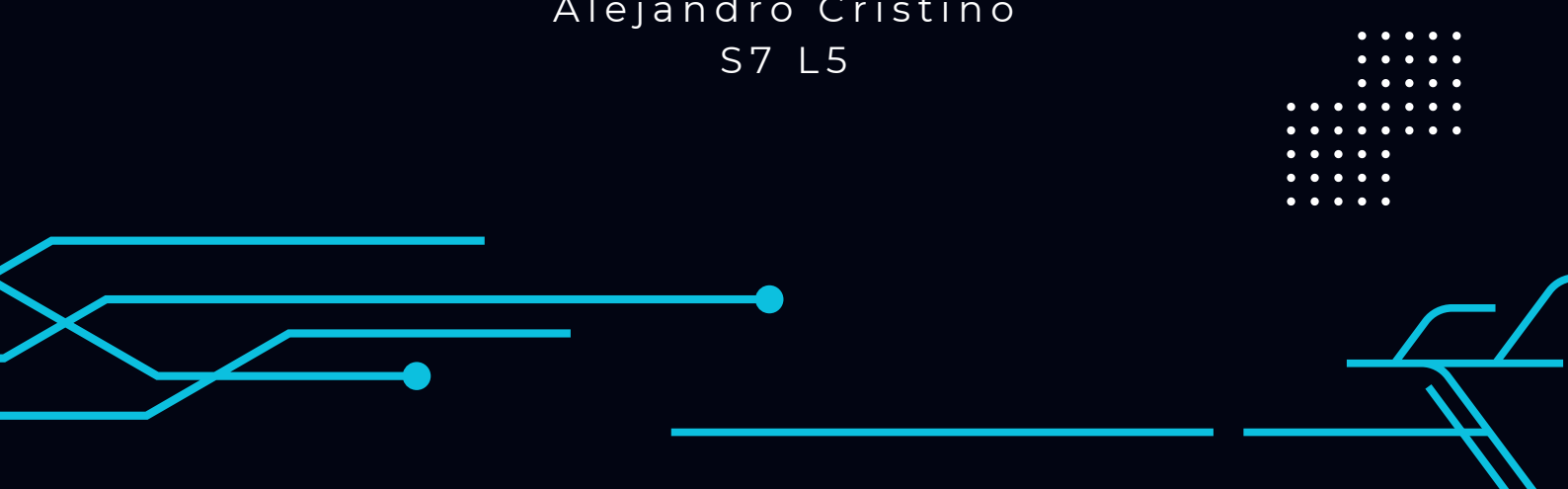




REPORT

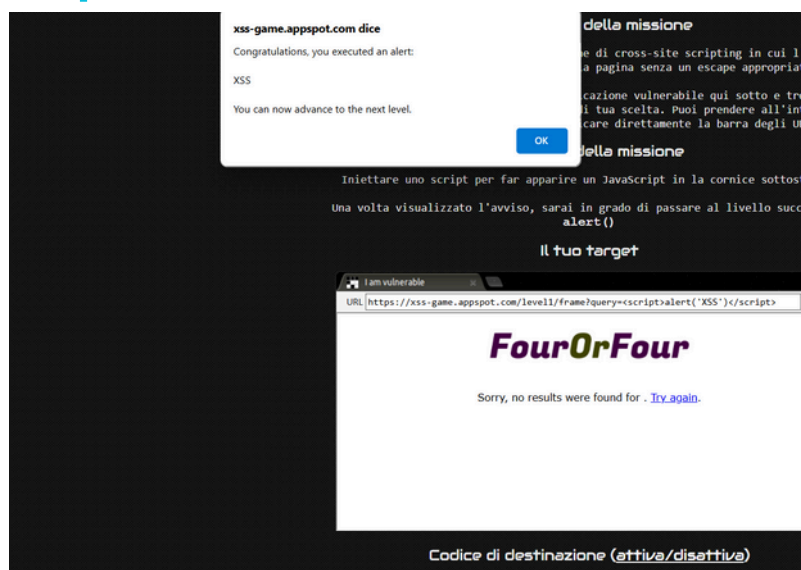
BONUS

Alejandro Cristino
S7 L5



Livello 1:

Superato:

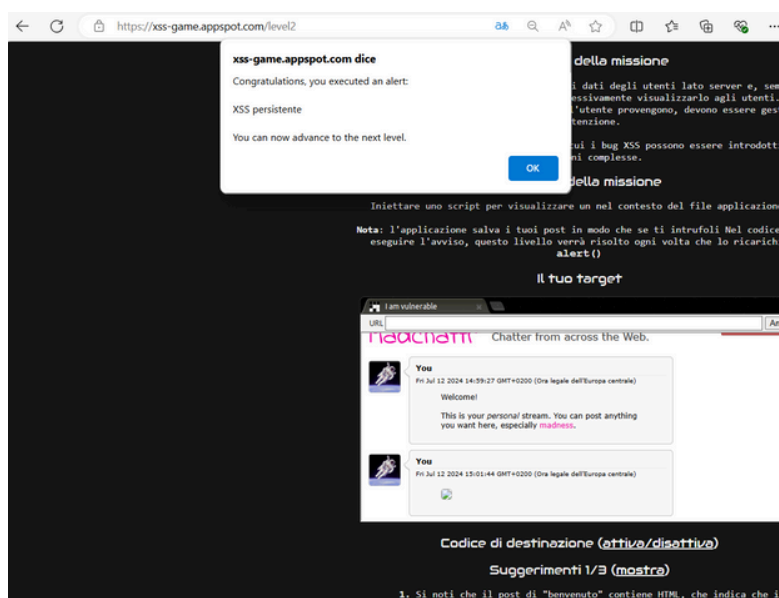


Soluzione:

Iniettando il seguente script nel campo URL per far apparire un avviso (alert):
`<script>alert('XSS')</script>`

Livello 2:

Superato:



Soluzione:

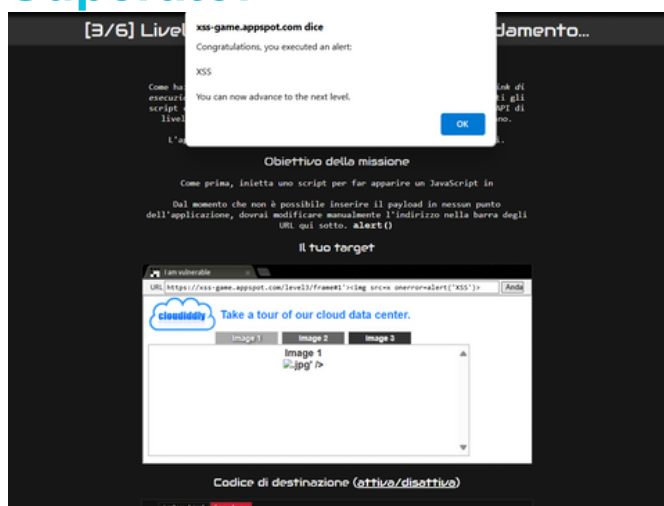
inserito il seguente codice nel campo di testo per sfruttare una vulnerabilità XSS persistente:

```

```

Livello 3:

Superato:



Soluzione:

<https://xss-game.appspot.com/level3/frame#1'>>

1. Chiusura del Contesto HTML:

- La stringa 1' chiude correttamente l'attributo src dell'immagine che chooseTab tenta di caricare.
- Questo è seguito da un tag di chiusura >, che chiude l'elemento immagine corrente.

2. Iniezione di un Nuovo Elemento HTML:

- Viene creato un nuovo elemento ``.
- Questo nuovo elemento immagine ha un src non valido (x), causando l'attivazione dell'evento onerror.

3. Esecuzione del Codice:

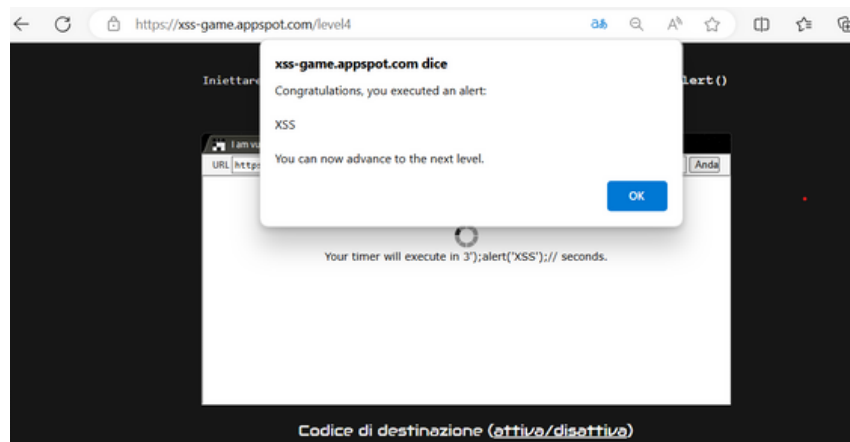
- Quando l'errore si verifica, l'evento onerror esegue `alert('XSS')`, mostrando l'avviso.

Motivo del Successo

- Manipolazione del Frammento: `window.location.hash` viene passato a `chooseTab`, che non valida correttamente il contenuto prima di inserirlo nel DOM.
- Inserimento Direttamente in `innerHTML`: Il contenuto viene inserito direttamente in `innerHTML`, permettendo l'esecuzione del payload.

Livello 4:

Superato:



Soluzione:

`https://xss-game.appspot.com/level4/frame?`

`timer=3%27%29%3Balert%28%27XSS%27%29%3B%2F%2F`

1. Chiusura del Contesto:

- `3%27%29` corrisponde a `3')`, che chiude correttamente l'attributo `onload`.

2. Iniezione del Codice:

- `alert('XSS')` è il codice JavaScript iniettato.

3. Commento del Resto del Codice:

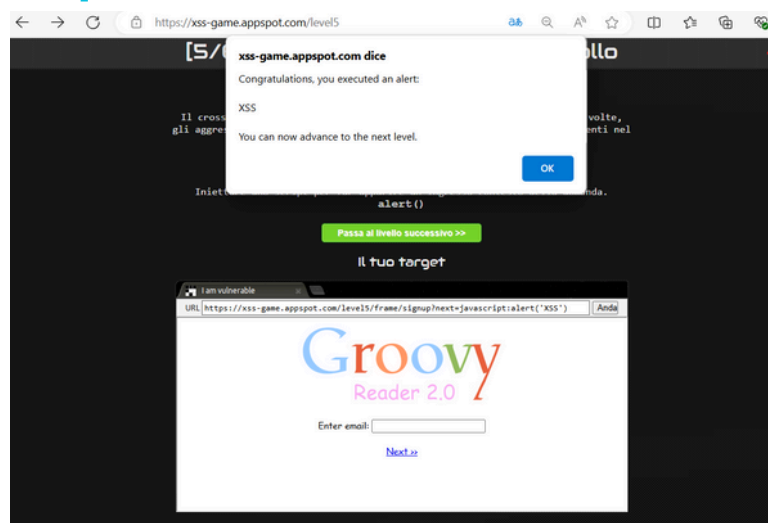
- `;%2F%2F` corrisponde a `;//`, che commenta il resto della linea per evitare errori di sintassi.

Passaggi di Attacco

1. **Identificazione della Vulnerabilità:** Il parametro `timer` viene inserito direttamente nel codice JavaScript senza sanificazione.
2. **Creazione del Payload:** Utilizzare una virgoletta singola per chiudere il contesto corrente e iniettare il proprio script.
3. **Codifica del Payload:** Codificare il payload per inserirlo correttamente nell'URL.
4. **Verifica dell'Esecuzione:** Ricaricare la pagina con il payload per verificare l'esecuzione dell'avviso.

Livello 5:

Superato:



Soluzione:

<https://xss-game.appspot.com/level5/frame/signup?next=javascript%3Aalert%28'XSS'%29>

1. Parametro next:

- Il parametro next viene passato attraverso l'URL alla pagina signup.html.
- In signup.html, il parametro next viene inserito direttamente nell'attributo href di un link.

2. Iniezione di JavaScript:

- Invece di un URL normale, il payload inietta un codice JavaScript utilizzando `javascript:alert('XSS')`.
- Quando l'utente clicca sul link "Next", il codice JavaScript viene eseguito, mostrando un pop-up con il messaggio "XSS".

Passaggi di Attacco

1. Identificazione della Vulnerabilità:

- Il parametro next viene inserito senza sanificazione nell'attributo href del link.

2. Creazione del Payload:

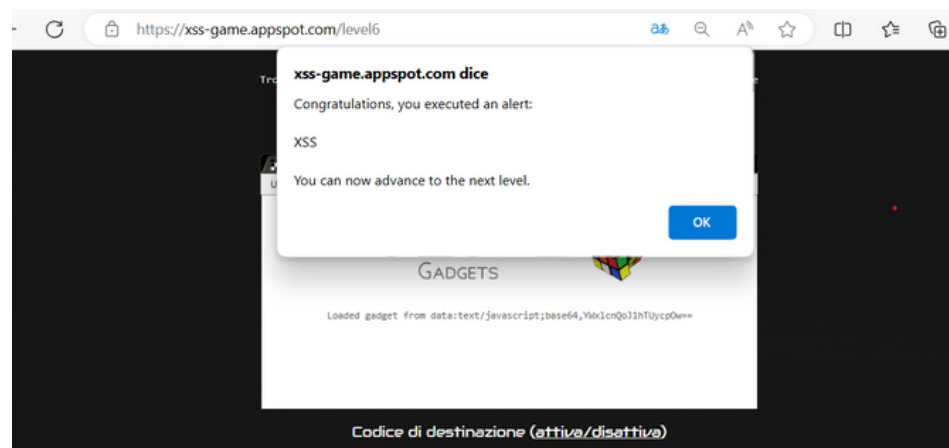
- Utilizzo del protocollo `javascript:` per eseguire il codice JavaScript direttamente.

3. Codifica del Payload:

- Codifica del payload per inserirlo correttamente nell'URL.

Livello 6:

Superato:



Soluzione:

`https://xss-game.appspot.com/level6/frame#data:text/javascript;base64,YWxlcuQoJ1hTUyYcpOw==`

1. **Data URI:** Utilizza il formato `data:text/javascript;base64`, per codificare inline uno script JavaScript.
2. **Codifica in Base64:**
3. Il codice JavaScript `alert('XSS');` viene codificato in Base64 come `YWxlcuQoJ1hTUyYcpOw==`.
4. **Caricamento Dinamico:**
5. La funzione `includeGadget(url)` carica lo script specificato nel frammento dell'URL, bypassando il controllo di sicurezza che blocca solo "http" e "https".

