**Documentation**

* What is xss?
* Cross site scripting is a security bug which can affect our website. This bug can allow the attacker to inject malicious JavaScript code or html tags. Once executed this can cause some of issues like: changing the appearance and behavior of the website, data leakage or perform some actions on the behalf of the user and etc.

There are several types of xss:

1. Reflected xss.
2. Persistent xss.
3. DOM xss.
4. Mutation xss

**Reflected XSS:**

* Reflected xss attack is also known as non-persistent xss occurs when a malicious script is reflected off of a web application to the victims’ browser.

The script will be activated through a link when the script payloads will be injected into the code and then it will be send It to user through third party website or in comment section or in social media. And this code will be injected in such a way into anchor tag, which makes user to click it and which initiates the xss to an exploited website, reflecting the attack back to the user.

Example of tornado web application:

Here is the link for tornado web application which contains 3 types of xss.

https://github.com/sinujacob/TornadoVulnerableWebApplication

For reflected:

Login as : <http://localhost:8004/signup>

- Here you will give user credentials and signup.

- After that you will login with proper user credentials, which will take you to the page:

<http://localhost:8004/home>

which will give you some user contained list:

* List all the users
* DomXSS example
* Logout
* Search
* Then you go to the search button.

For example in search, you will search for a name “Jacob” you will get the message:

“Welcome Jacob”

* So now when you inject the script “<script>alert(XSS)</script>” an alert box will pop up

Saying xss. This is a bug

This is reflected xss attack.

So this way an attacker can visit any page. This page might include an invisible iframe that points to site which is vulnerable to xss, along with the payload to exploit the vulnerability. Xss vulnerabilities can result in much more than alerts on a page, it is just a convenient way for an attacker to detect the xss bug.

**Prevention:**

* Every data must be encoded and special characters must be escaped properly means user input should be validated.
* There are many data that should be sanitized properly, like:
  + The URL
  + GET parameters from a form.
  + GET parameters from a form.
  + Document.location
  + Document.URL and many more.
* Special characters must be encoded, like:
  + & - &amp;
  + < - &lt
  + > - &gt
* Example from tornado web application:
* Link : https://github.com/sinujacob/SecureTornadoWebApplication
  + After the user logs in, it goes to the list of user profile:

<http://localhost:8004/home>

which will give you some user contained list:

* + List all the users
  + DomXSS example
  + Logout
  + Search

Now an attacker in search bar injects JavaScript code “<script>alert(‘hi’)</script>”, after executing it will show “Welcome <script>alert(‘hi’)</script>” because the special characters are encoded. So when you look into the source code, it will be displayed as “&lt;script&gt;alert(‘hi’)&lt;/script&gt” because it is encoded.

* Another way of preventing is “crossing boundaries” policy where authenticated users have to re-enter their passwords before accessing certain services. For example, even if a user has a cookie which will automatically log them into a website, they should be force to enter the username and password again when trying to access the sensitive data.

**Persistent xss:**

* In case of persistent xss, the code injected by the attacker will be stored in the secondary storage devices. The damage caused by the persistent attack is more than the non-persistent attack.
* For example, consider a web application that allows user to enter username that is displayed in each user’s profile page. The application stores each username in local database. A malicious user notice that web application fails to sanitize the username field and inputs malicious JavaScript code as part of their username. When other users view the attacker’s profile page, the malicious code automatically executes in the context of their session.

.

Example of tornado web application:

* Here you will give user credentials and do signup.

<http://localhost:8004/signup>

* Then you will login with proper username. Then you can see the user list in the below mentioned options:

<http://localhost:8004/home>

Which will give you some user contained list:

* List all the users
* DomXSS example
* Logout
* Search
* Now attacker can see some flaw here. So he can do signup by giving JavaScript code in username. “<script>alert(‘XSS’)</script>”.
* So whenever a valid user logs in, an alert box will popup saying xss, whenever he tries to list the user. As the javacsript code will be stored as a part of username in local database.
* So here an attacker can perform malicious actions like: hijack an account, control the browser remotely and many more.
* Prevention same as reflected.

**DOM XSS:**

* DOM based XSS is an XSS attack wherein the attack payload as a result of modifying the DOM environment in the victim’s browser used by the original client side script, so that the client side code runs in an unexpected manner.
* There are many objects where an attacker can manipulate in order to generate xss condition. Like: document.refrerrer, document.url, and document.Location and many more.
* The most dangerous part of Dom xss is, the attack is often a client side attack, the attackers payload is never sent to the server.
* Example from tornado web application:
  + User login:

<http://localhost:8004/login>

* There will be list of users for the particular user:
  + - List all the users
    - DomXSS example
    - Logout
    - Search
* Go to DomXss, in that you have to search for a language.
  + For example you are giving “Russian” then it will take you to the url

Saying <http://localhost:8004/dom1?lang=russian>

* + Now an attacker can see in the url that something he can sniff after “Question mark”, because after that it is user input and that is not sanitized properly.
  + So an attacker can inject javacsript code in that url after lang : [http://localhost:8004/dom1?lang=<script>alert(‘hi’)</script>](http://localhost:8004/dom1?lang=%3cscript%3ealert('hi')%3c/script%3e) then after executing this you will get an alert popup box saying hi. So there is bug in client side.
* **Prevention:**
* Avoid client side sensitive actions such as rewriting or redirecting, using client-side data.
* Sanitization of client-side code such as URL, location, referrer especially in cases when Dom can be modified.
* Need to be validated properly in the client side code.
* Example for tornado web application:
  + User login:

<http://localhost:8004/login>

* There will be list of users for the particular user:
  + - List all the users
    - DomXSS example
    - Logout
    - Search
* Go to DomXss, in that you have to search for a language.
* Now when you inject a JavaScript code in the url “[http://localhost:8004/dom1?lang=<script>alert(‘hi’)</script>](http://localhost:8004/dom1?lang=%3cscript%3ealert('hi')%3c/script%3e)” alert box will not be poped up as it is encoded.

encodeURI(document.location.href.substring(document.location.href.indexOf("lang=")+5)