Client Side Injection on Web Applications

Author: Milad Khoshdel

Blog: https://blog.regux.com

Email: miladkhoshdel@gmail.com

Contents

INTRODUCTION

As you know web applications or websites works with communicating with users. For example in a forum websites, all information will shared by different users from different places or in a news website users can comment on each news or articles and this comments are visible by other people. So dangerous data should not store by users in database and it is very important for developer and website admin to make sure that Meta characters will filter on user input forms data.

There are too many technic for performing client side attacks. I will explain bellow attacks in this articles.

- HTML Injection
- CSS Injection
- Client Side URL Redirect
- DOM-Based Client Side Scripting

HTML Injection Vulnerability

This vulnerability will happen when hacker is able to inject HTML tags into a form and finally store the html code into database.

Imagine we have a blog and users are able to share their comment under our blog posts. In a html injection vulnerable form, hacker is able to write html tags as comment and this code will be visible for all people. As you know with html tags hacker can do bellow malicious actions:

- Can change the style of the text
- Can insert a Link with a tag within his comment text and tempt people for clicking on this link (The link may contain malicious information).
- Can insert an Image within his comment

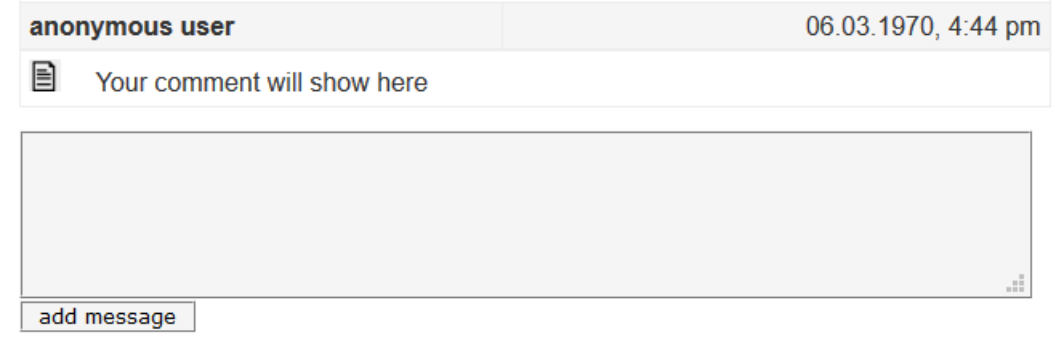
And if HTML Injection combine with other attacks:

- Can steal User Cookie. (XSS)
- Can redirect the vulnerable page to other destination automatically. (XSS)
- Can Deface the Vulnerable Webpage. (CSS Injection)

In this section I only explain HTML injection and I will cover other items in next sections.

Let's explain HTML injection vulnerability with a pictorial Example.

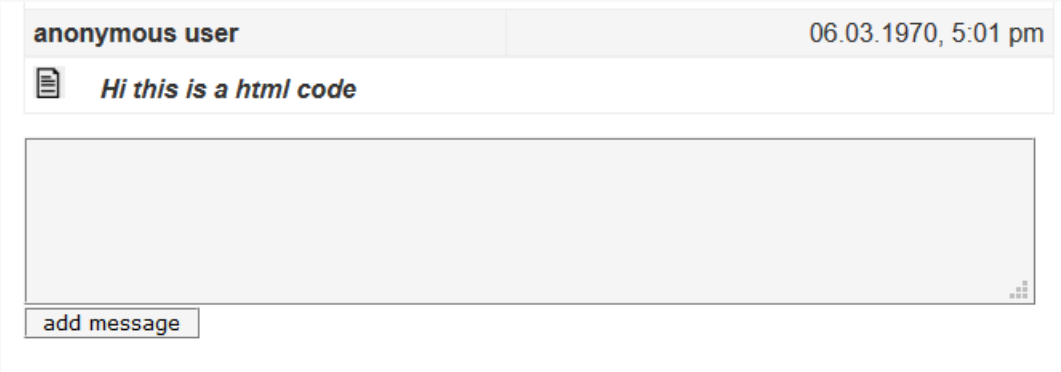
We have a website with a guestbook that allow users to share his comments for other people. You can see the comment section of this website in bellow image:



Now we want to store out html code into database instead of plain text comment. We try to submit bellow html code as a valid comment.

```
<b><i>Hi this is an html code</i></b>
```

And here is the result:



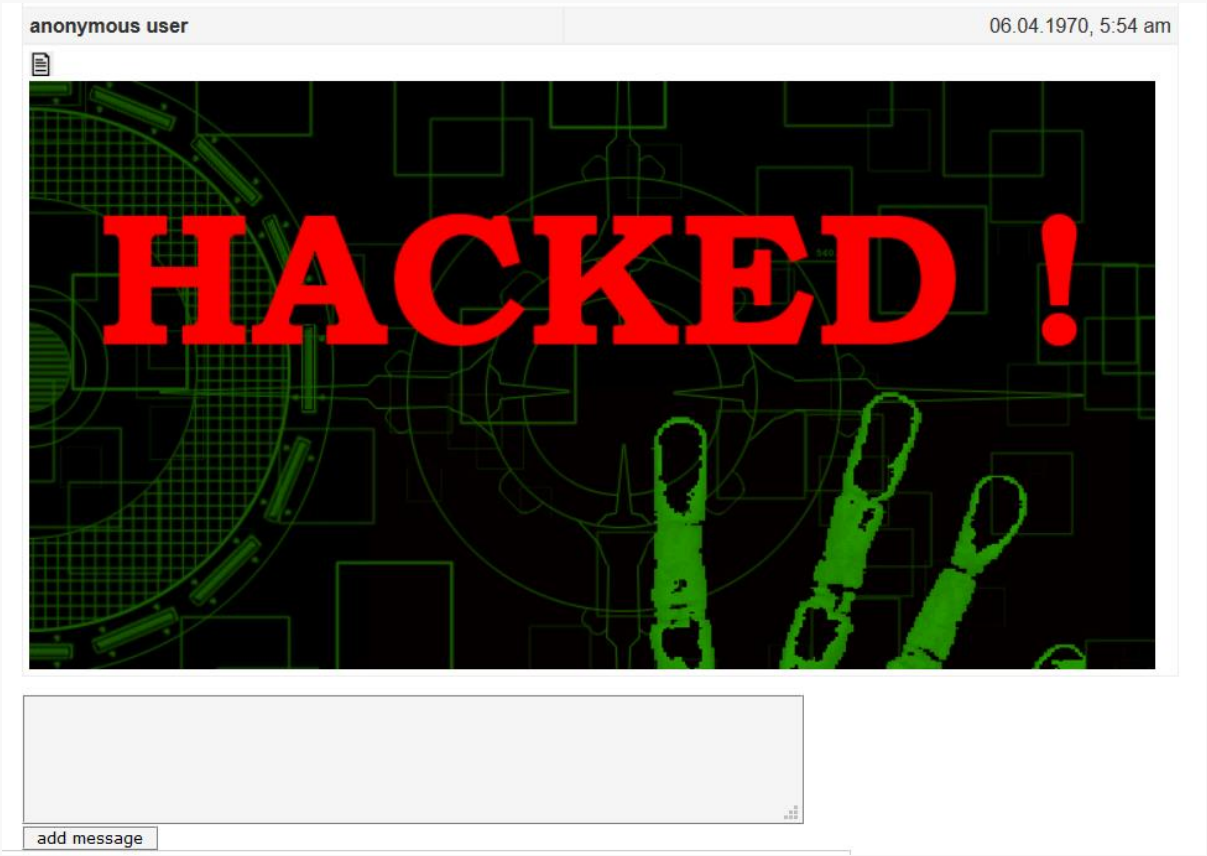
As you see, out comment is stored in HTML Format.

We can post plenty of things with HTML code, for example we can post an image or even we can load another site in this page using iframe tag and do a deface attack.

We are going to load an image in comment section.

```
<img src="YOUR IMAGE URL">
```

And here is the result:



If your image size is 2MP or more, you can do a successful deface attack using some CSS injection technique. I will explain it with details in next section.

How to Mitigate HTML Injection?

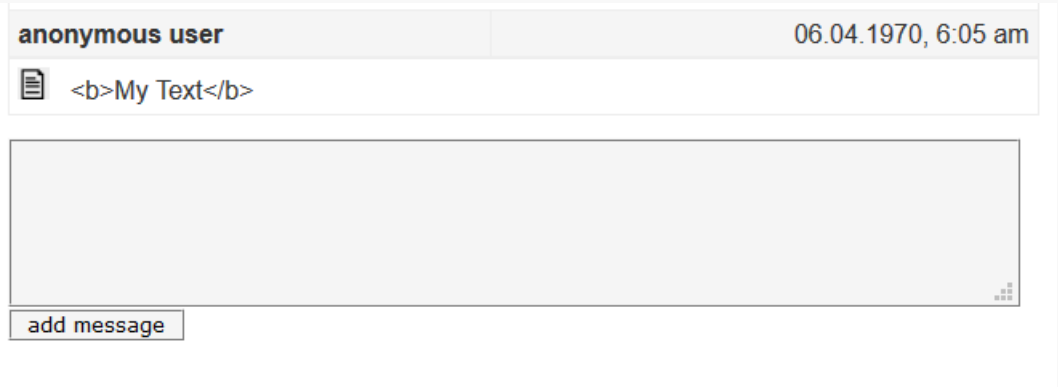
If you are a programmer and you are going to fix this vulnerability on your website, you just need to filter Meta characters and translate these characters to Entity Names.

Please note that if you don't use any frameworks (such as Django, Laravel, …), you need to do it manually. In the reference section, you have a link to the full encoding table. But for now, the most commonly used codes are:

```
Character  Entity Number  Entity Name  Description
"          &#34;          &quot;       quotation mark
'          &#39;          &apos;       apostrophe (does not work in IE)
&          &#38;          &amp;        ampersand
<          &#60;          &lt;         less-than
>          &#62;          &gt;         greater-than
```

So if you use Entity names for Meta characters, Browser does not parse HTML code in client side and just show the code in plain text format.

I use this technique in our vulnerable website and here is the result:



CSS Injection Vulnerability

In previous section, I explained HTML injection. Now I want to explain CSS Injection. As you know, CSS is used to define styles for web pages, including the design, layout and variations in display for different devices and screen sizes. It means if you found a web page that is vulnerable to HTML Injection, you can use CSS Injection for changing your injected HTML elements style. For example if you combine the malicious HTML code with CSS, you can change your image size and cover all the webpage screen with your image and it means you performed successful deface attack.

Here is the code:

```
<img style=”padding:1000px;” width=”500px;” height=”500px;” src=”OUR-IMAGE-URL”>
```

Also some times when an application imports a style sheet from a user-supplied URL, or embeds user input in CSS blocks without adequate escaping, we faced some kind of CSS Injection.

For Example, take a look to bellow vulnerable code:

```
<a id="input">Click me</a>
<script>
  if (location.hash.slice(1)) {
    document.getElementById("input").style.cssText = "color: " +
location.hash.slice(1);
  }
</script>
```

And here is the vulnerable URL:

```
https://www.url/#red
```

After loading URL, we will have bellow style in web page style tag:

```
<style>
 #input{
    color: red;
 }
</style>
```

Now we can inject our CSS at the end of the URL. For example if we load bellow URL:

```
https://www.url/#red; padding:1000px; font-size:70px; background-color:#000000
```

It means that we injected extra CSS code into style tag. After loading this URL the style tag will be as bellow:

```
<style>
 #input{
    color: red;
    padding:1000px;
    font-size:70px
    background-color:#000000;
 }
</style>
```

In some case you can even import CSS file from external source.

For example:

```
https://www.url/#red; @import "https://externalsource/navigation.css"
```

How to Mitigate CSS Injection

For preventing CSS Injection, you shouldn't allow your user to add code in CSS blocks using embeds input without escaping. So Ensure that user input is adequately escaped before embedding it in CSS blocks, and consider using a whitelist to prevent loading of arbitrary style sheets.

Finally, the vulnerability can be mitigated using the following best practice steps, which may look awfully familiar:

- Set the server header X-Frame-Options: deny on all pages
- Set the server header X-Content-Type-Options: nosniff on all pages
- Set a modern doctype (eg: <!doctype html>) on all pages

Client Side URL Redirect Vulnerability

Client Side URL Redirection, also known as Open Redirection and Base on OWASP explanation, It is an input validation flaw that exists when an application accepts an user controlled input which specifies a link that leads to an external URL that could be malicious. This kind of vulnerability could be used to accomplish a phishing attack or redirect a victim to an infection page.

This vulnerability occurs when an application accepts untrusted input that contains an URL value without sanitizing it. This URL value could cause the web application to redirect the user to another page as, for example, a malicious page controlled by the attacker.

By modifying untrusted URL input to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials. Since the redirection is originated by the real application, the phishing attempts may have a more trustworthy appearance.

For example, look at the bellow JavaScript code:

```
<script>
var redir = location.hash.substring(1);
if (redir)
   window.location='http://'+decodeURIComponent(redir);
</script>
```

This function will redirect user to another internal page after logout.

This is the URL:

```
http://url/?#www.url/login
```

In this vulnerable code, script does not perform any validation of the “redir” value that contains the user supplied input via the query string, this means that an attacker could redirect the victim to a phishing site simply by submitting the following URL:

```
http://url/?#www.attackerurl/phising-page
```

So after Logout, user will redirect to malicious URL rather than Login page.

How to Mitigate Client Side URL Redirect

Programmer can prevent this kind of attacks by checking the URL that is passed to the redirect function. Programmer should make sure all URLs in this function are relative paths – i.e. start with a single "/" character. (External URLs start with // will be translated by the browser as a protocol, absolute URL – so programmer should reject this kind of URLs too.)

If programmer needs to perform external URL redirection, he should consider white Lists for all of external URLs too.

Also for the second layer of defense, programmer should check that the `Referrer` in the HTTP request header matches the domain name whenever redirect process performs.

DOM-Based Client Side Scripting Vulnerability

Dom-Based XSS is an XSS attack that payload will inject by modifying the web page DOM Elements and it means that the client side code runs unexpectedly.

In previous attacks, I mean HTML Injection and URL Redirect, you can exploit Dom-Based XSS. For example,

In HTML Injection you can Exploit Dom-Based XSS with bellow payload:

```
<img src='some-link' onerror=alert(1)>
```

And here is the result:

And in Client Side URL Redirect you can Exploit Dom-Based XSS with bellow Payload:

```
http://www.url/?#javascript:alert(document.cookie)
```

Please note that vulnerable JavaScript code in this case is:

```
<script>
var redir = location.hash.substring(1);
if (redir)
   window.location='http://'+decodeURIComponent(redir);
</script>
```

Another example for Dom-Based Xss

Imagine we have a website with a form for changing the website language. Also we have a JavaScript function for handling this form.

This is the JavaScript Code:

```
<select><script>

document.write("<OPTION
value=1>"+document.location.href.substring(document.location.href.indexOf("
lang=")+8)+"</OPTION>");

document.write("<OPTION value=2>English</OPTION>");

</script></select>
```

And here is the changing language URL:

```
 http://www.url/page.php?lang=French
```

According to JavaScript Code, lang parameter load in URL using GET method, so if you change the value of lang parameter, the language of the page will change using JavaScript function. It means you can inject your JavaScript malicious payload as a value for lang parameter.

For Example, you can exploit Dom-Based XSS using bellow payload in URL:

```
http://www.url/page.php?lang=<script>alert(document.cookie)</script>
```

How to Mitigate DOM-Based Client Side Scriptings

If you need to Load HTML code into Dome Elements dynamically, you should use HTML encoding with bellow function:

- element.innerHTML
- element.outerHTML

Also for second layer of security, you can use JavaScript encoding for all untrusted input with bellow functions:

- document.write
- document.writeln

Also for the first example, preventing HTML Injection and Client Side URL Redirect can prevent DOM-Based XSS too.

Have Fun
Milad Khoshdel

References

1. https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet
2. https://portswigger.net/blog/detecting-and-exploiting-path-relative-stylesheet-import-prssi-vulnerabilities#badcss
3. https://www.owasp.org/index.php/Testing_for_Client_Side_URL_Redirect_(OTG-CLIENT-004)
4. https://portswigger.net/kb/issues/00501300_css-injection-reflected
5. https://www.owasp.org/index.php/Testing_for_CSS_Injection_(OTG-CLIENT-005)
6. https://www.hacksplaining.com/prevention/open-redirects