Hardik Solanki Follow
Dec 16, 2020 · 4 min read · Listen

HTML Injection-Stored: which ultimately resulted into a CVE-2020–26049

Hey Guys! I know HTML injection is not something you wanted to read but however its my first hunt for product security which ultimately resulted into a CVE-2020–26049.

HTML is a markup language, where all the website's elements are written in the tags. Web pages are sent to the browser in the form of HTML documents.

What is HTML Injection?

This injection attack is injecting HTML code through the vulnerable parts of the website. The Malicious user sends HTML code through any vulnerable field with a purpose to change the website's design or any information that is displayed to the user. As a result, the user may see the data that was sent by the malicious user. Therefore, in general, HTML Injection is just the injection of markup language code to the document of the page.

Types of HTML Injection

- Stored HTML Injection
- Reflected HTML Injection

#1) Stored HTML Injection: stored injection attack occurs when malicious HTML code is saved in the web server and is being executed every time when the user calls an appropriate functionality.

#2) Reflected HTML Injection: In the reflected injection attack case, malicious HTML code is not being permanently stored on the web server. Reflected Injection occurs when the website immediately responds to the malicious input.

> *while doing my pentest research, I ended upon a project management software called Nifty-PM.*

> *Product Details:*

Product: Nifty-PM (Version: CPE 2.3)

Vendor: Nifty

Vendor URL: https://niftypm.com/security/

Component URL: https://testing-stuff-010101.nifty.pm/

Bug: HTML Injection — Stored

Exploitable: Yes

Reported on: 23rd September 2020

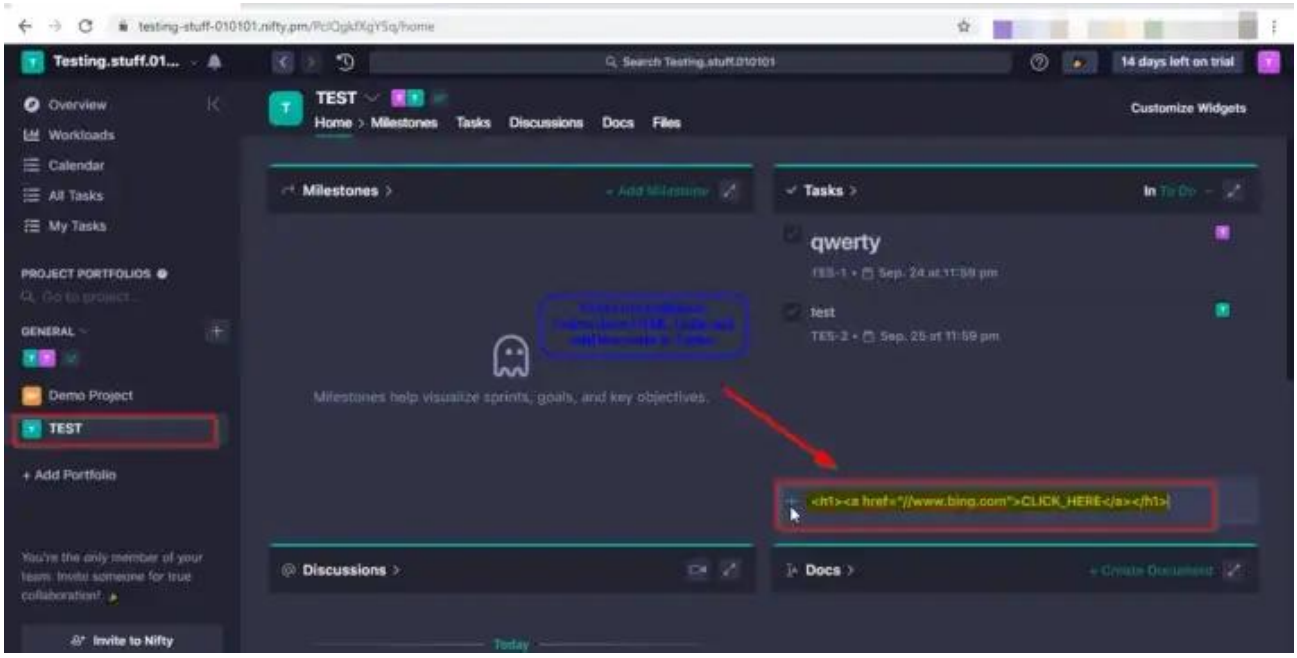
Vendor Fixed Issue on: 23rd September 2020

> *Impact of the vulnerability:*

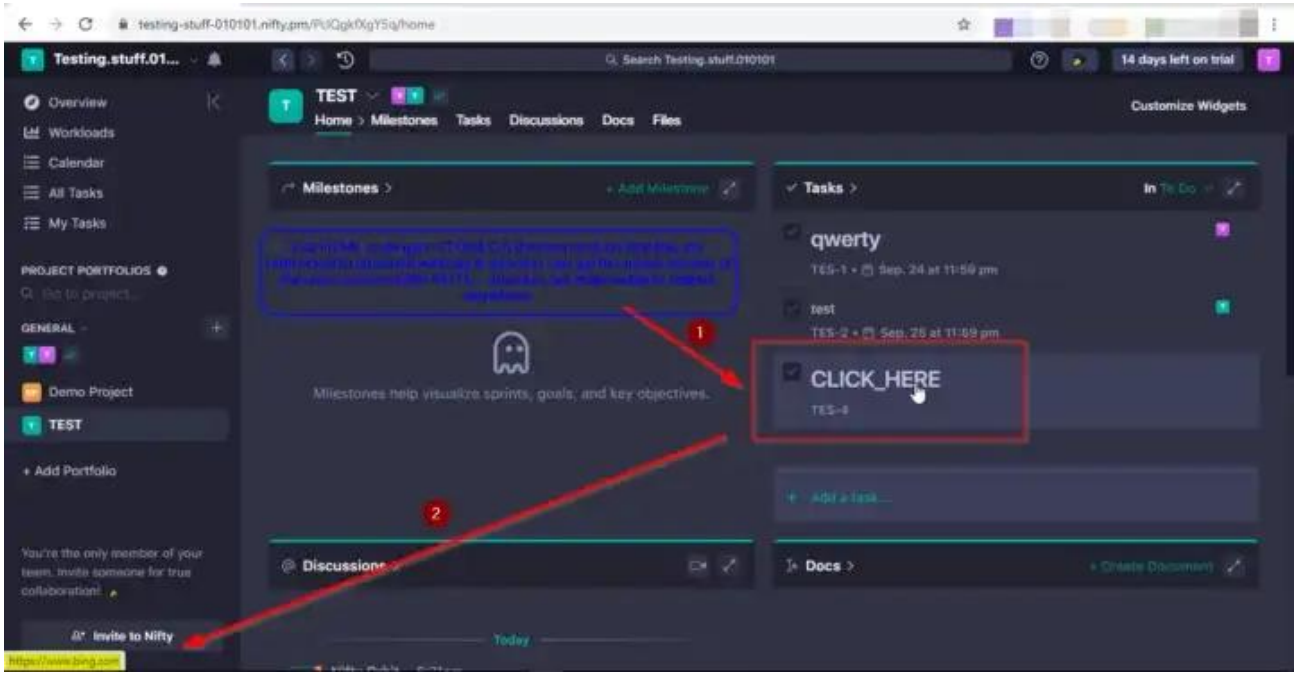
This vulnerability can have many consequences, like disclosure of a user's session cookies that could be used to impersonate the victim, or, more generally, it can allow the attacker to modify the page content seen by the victims.

> *Detailed Steps:*

Step 1: Add the following HTML CODE in the "Add Task" Field & Add that task, as shown in the following screenshots.
<h1>CLICK_HERE</h1>



Step 2: HTML CODE, get stored in the task, now if any one click on that task (CLICK_HERE button), then victim easily redirect to attackers malicious website (Bing.com), as shown in the following screenshots.



And BOOM! it redirected and got saved in tasks.

> *How to Prevent HTML Injection?*

The 2 broad strategies to mitigate HTML Injection attack are:

Input Validation: It is a validation process to make sure that the input matches the business requirement. The unwanted data entered in the application/field should be rejected. There are two ways to implement input validation:
1. Whitelisting good input
2. Blacklisting bad input
The strategies are explained in greater detail below. Whitelisting is the better solution, but also more difficult to implement. If it is not possible to implement whitelisting, then blacklisting should be done. Implementing both whitelisting and blacklisting adds two layers of defense, and is hence the strongest.
Whitelisting:
Create a whitelist of characters required by the application. Once this whitelist is ready, the application should disallow all requests containing any character apart from those in the whitelist.
Blacklisting:
The application should not accept any script, special character or HTML codes in the fields whenever not required. It should prevent special characters that may prove to be harmful. Some of the main characters used in scripts that must be prevented are as follows:

< > ( ) ‘ “ / \ * ; : = { } `(backtick) % + ^ ! — \x00-\x20 (x is a hexadecimal notation — it includes Space, Tab, CarriageReturn, and LineFeed.)
Apart from input validation, Output encoding (encoding or escaping) can also be applied to the application.

Output Encoding: It is a process to convert untrusted input into a safe form where the input is displayed as data to the user without executing as code in the browser.

Encoding: It is transforming data from one format into another format.
Example: URL Encoding, HTML encoding, etc.
Escaping: There is another alternative to Output Encoding i.e. Escaping. In escaping you either truncate/exclude the unwanted special character or add an escape character in front or back of a special character. This addition of special character changes the meaning of the script/payload (malicious code) entered and hence prevents the malicious code to be executed.
Example: Adding \ (backslash) before or after any special character.

> *The vulnerability is now patched.*
>
> ***CVE ID:*** *CVE-2020–26049*

Cheers!

Happy Hunting#Togtherwehitharder#bugbounty#webapplication