iGNITE Technologies

BURP SUITE FOR PENTESTER

XSS VALIDATOR

Contents

Introduction

You might have used several online tools to detect XSS vulnerabilities and a few to validate them and thereby, at last, with all the generated outcomes you try to exploit the injection point manually or with burpsuite's fuzzing. But what, if we get all these things wrapped up in a single place. Today in this article, we'll learn one of the most important burp suite extensions i.e. XSS Validator, which thereby automates the detection and validation for XSS vulnerabilities in the web application.

Extensions & the BApp Store

Burp Extensions

You might have heard the term ***"Extension"***, probably for a browser, whether it is for chrome or firefox, so what are they?

Extensions are small programs scripted to enhance the functionalities over in an application. Thereby, the burp suite offers a feature to **customize its behavior** and to **extend the capabilities** it carries up, whether it is modifying the HTTP requests and responses, customizing the UI, or adding the custom Scanner checks, all it wraps up in the form of **Burp's Extensions**.

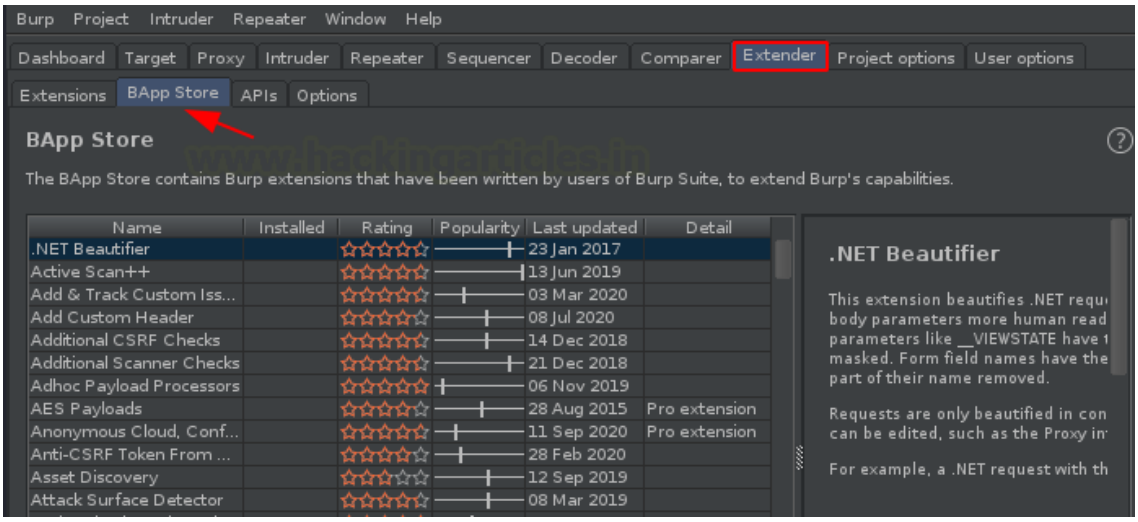
BApp Store

But where to find such burp extensions??

Over at burp suite, we're having one tab that is built only to manage the burp's extensions and i.e. the **Extender**. The Extender tab helps us to manage everything related to an extension, but in this, there is a sub-tab too, called the **BApp Store**, which is a **hub that contains** a variety of **"Burp Extensions"**.

There at the BApp store, we can **view the list of available apps**, **install a specific one**, and even we can **submit a user rating** for those we've already installed.

However, some extensions might have been removed from the BAPP Store, or even we need to set up ours in the burp suite. Therefore, for such scenarios burp provides us with an opportunity to **manually install an Extension** there.



About XSS Validator

Setting up the XSS Validator

XSS Validator commonly termed as Burp Intruder Extension is designed to detect and validate the most crucial Cross-Site Scripting vulnerability, which works collaboratively with the burp's intruder to capture a successful XSS drop out.

John Poulin the author of this extension, developed it in 2017 intending to automate the detection of XSS vulnerabilities in vulnerable web applications.

This extender is most common due to its minimal false positives and the in-build payload list, where every payload is bound up with a trigger value of "f7sdgfjFpoG".

Although being a validator, this extension also contributes as a Detector. However, to make the attack successful, the XSS Validator sends responses to a locally-running XSS-Detector server i.e. either Phantom.js and/or Slimer.js

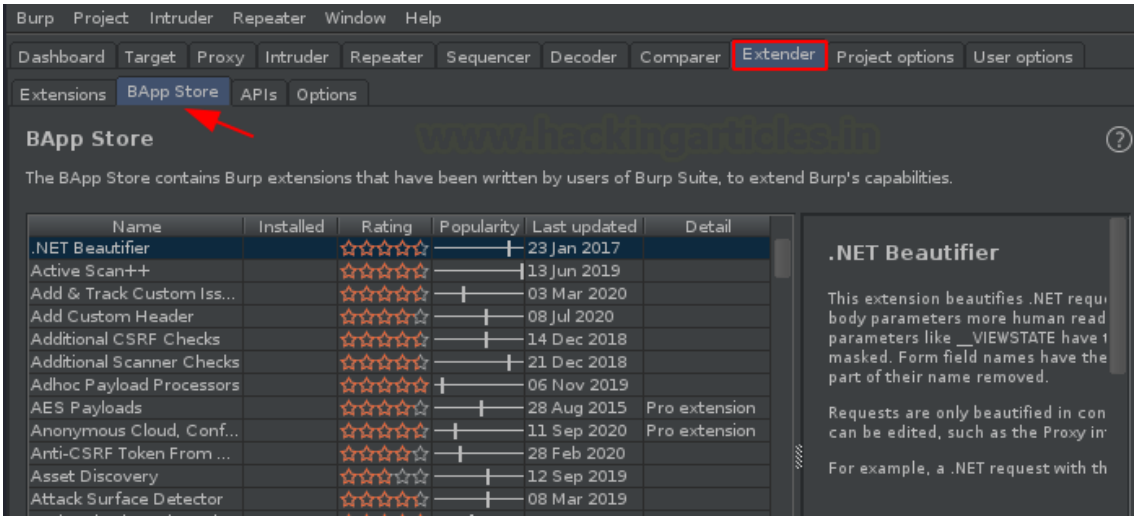
Let's explore the installation and the attack scenario of this XSS Extension to be more precise about its working.

Installing the Extension from BApp Store

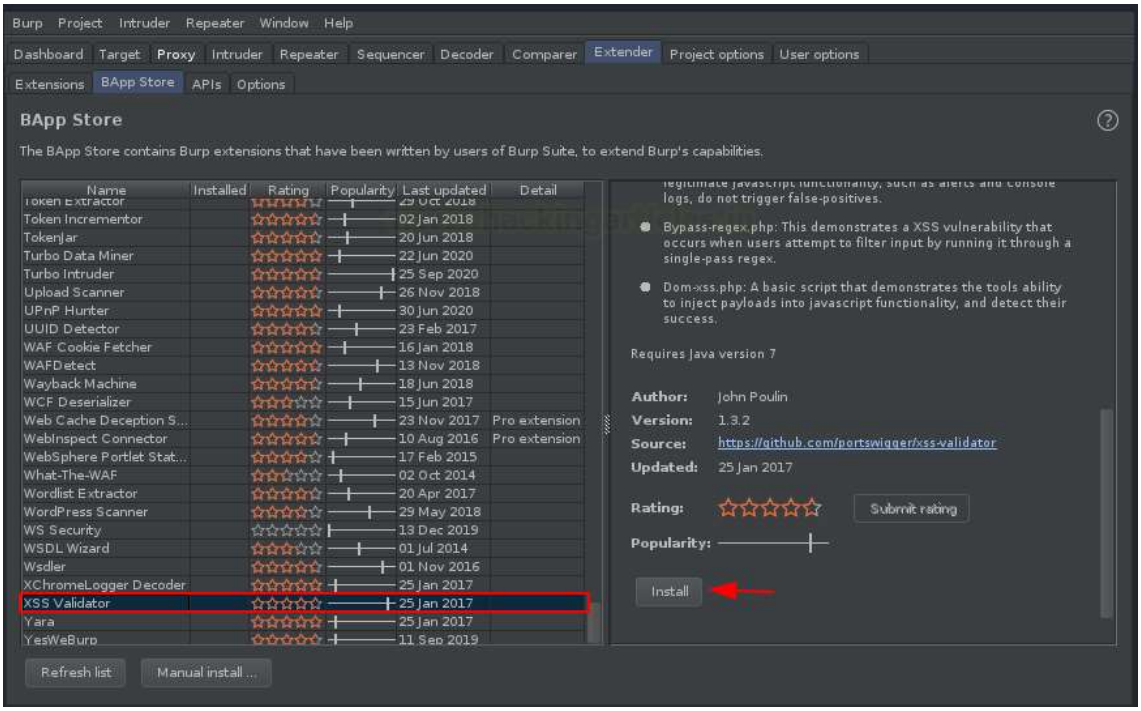
As we've already discussed, that the BApp store carries up to several extensions within itself, and thus being the most common, XSS Validator can be found there.

Thereby, this extension can thus be installed up with some simple steps.

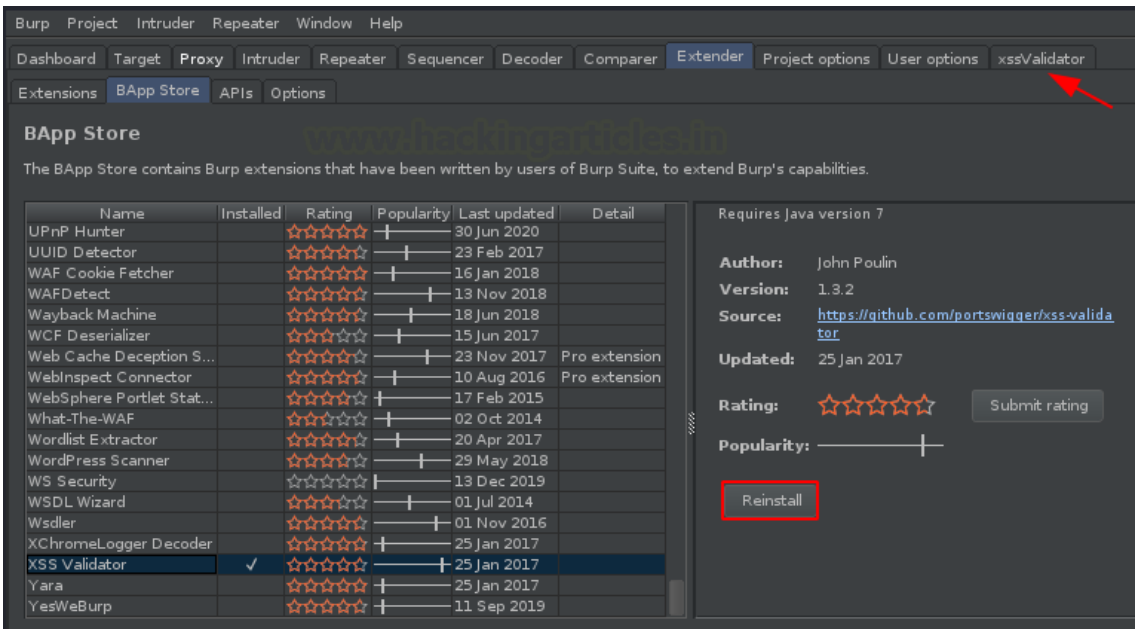
Over at the tabs provided at the Burp Suite monitor, navigate to the **Extender tab**, opt the **BApp sub-tab** there, such to check the list of the provided extensions.



Let's scroll down until we reach the end of the list, and with that, we'll get our extension placed.

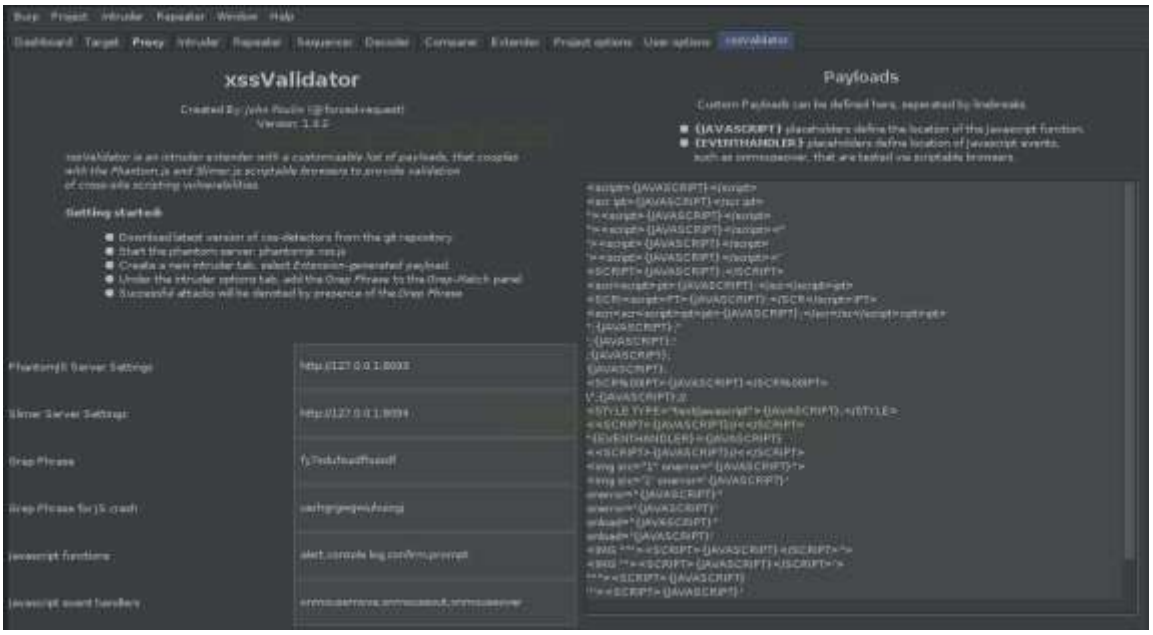


As soon as we hit the **install** button, it will start downloading, and within a few minutes, we'll have our **extension added** at the tabs panel as **"xssValidator".**



Let's click and check what it offers to us.

Over at the right side of the below image, we can see, a list with some **payloads is offered** to us, and along with it, at the left side, we got the **Grep Phrase** which is bound to trigger every payload added there. Above it, we're having the **PhantomJs Server setting**, which we'll use in the future scenario.



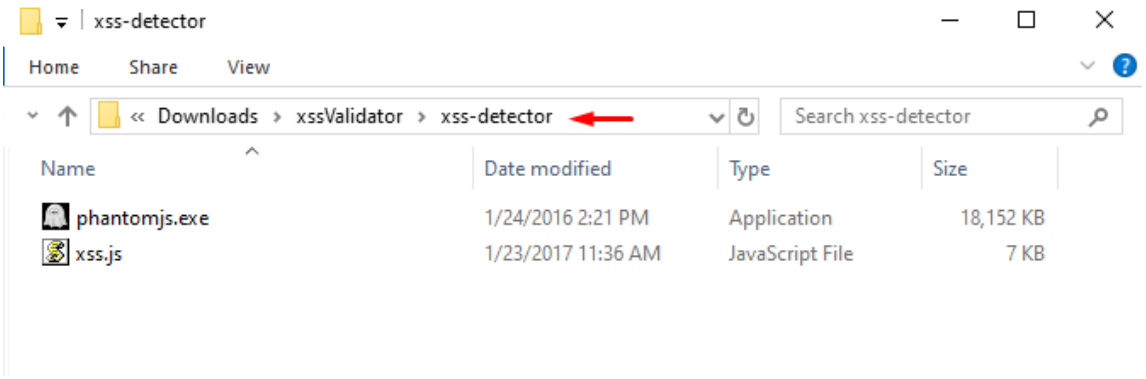
Installing Phantom.js as an XSS Detector

You might be wondering, what is this Phantom.js and why we are installing it??

Phantom.js is a command-line tool, basically a **headless browser** i.e. it does not contain any GUI interface, which thus runs itself silently in the background.

There are times when the **DOM-based XSS exists** and if we try to hit that, it executes up in the background but **does not get captured by the browser**, therefore to reduce this **false-negative** and to detect every possible XSS, we're thus installing this **Phantom.js**. You can download it from **here**.

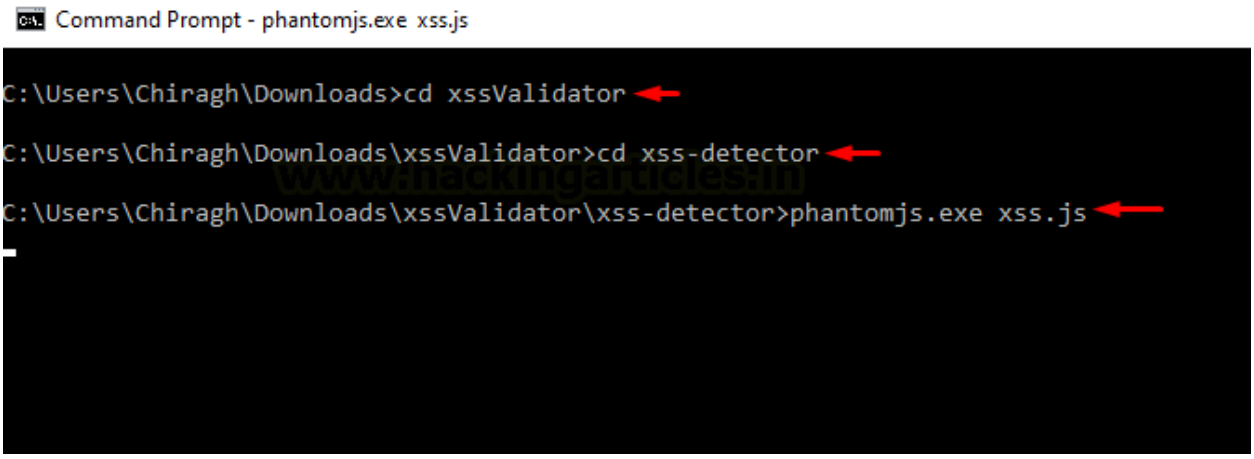
Now, to integrate our XSS validator with this Phantom.js, we need to **download the xss.js** file which thus could be done from **here**.



As soon as we do so, we simply need to **move our Phantom.js exe** to the XSS detector directory.

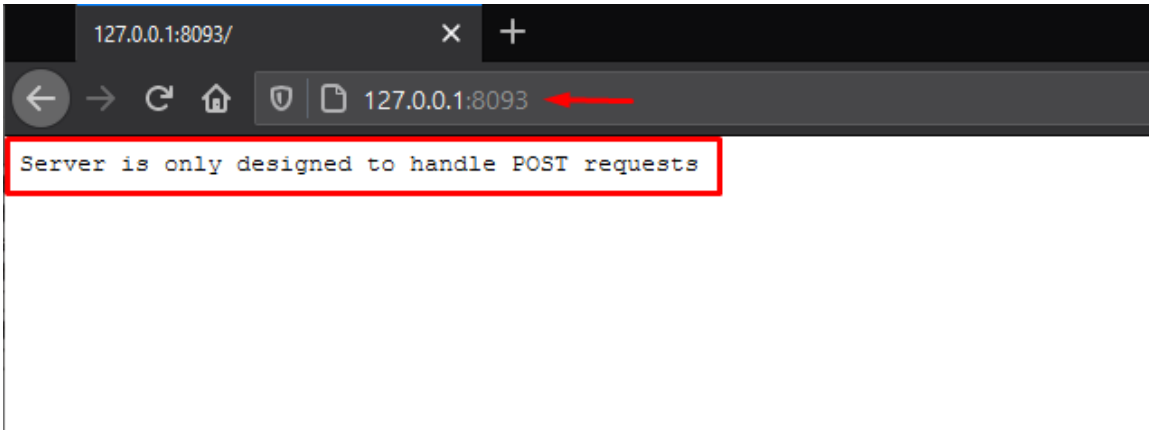
And now with all this, navigate the directory in the command prompt and execute the following command to initiate the server.

```
cd xssValidator
cd xss-detector
phantomjs.exe xss.js
```



Let's check whether our server is running or not by executing http://127.0.0.1:8093

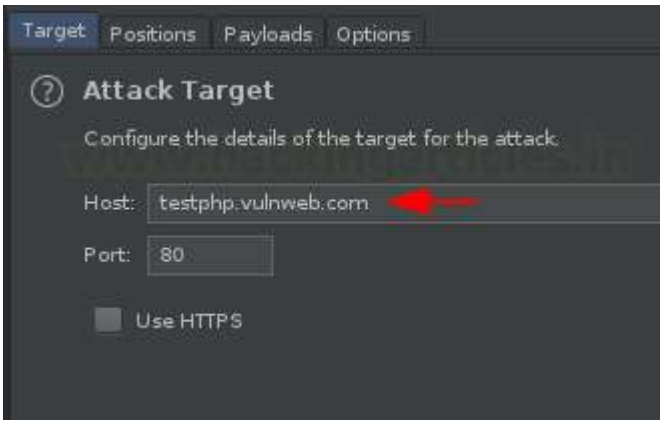
Cool!! From the below image, we can see that our detector server has been configured successfully.



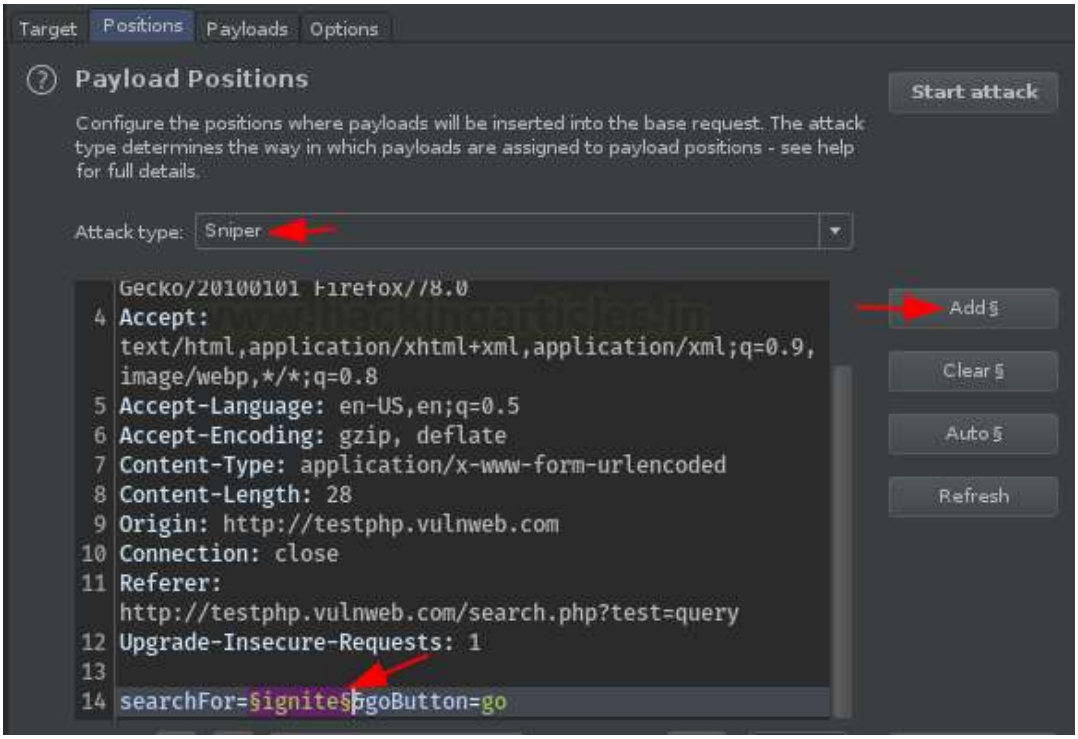
Fuzzing with XSS Validator

I guess you might be clear with the installation section, lets now do a **quick fuzzing** on the search field over at **test.vulnhub** with the XSS Collaborator.

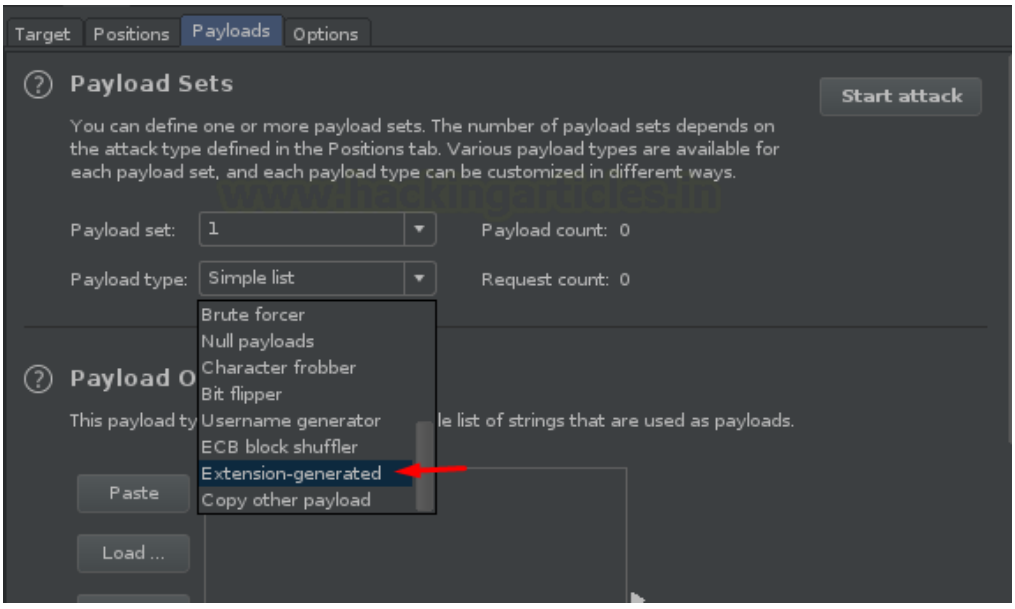
We'll capture the ongoing **HTTP Request** and thus will share it with the **Intruder** directly.



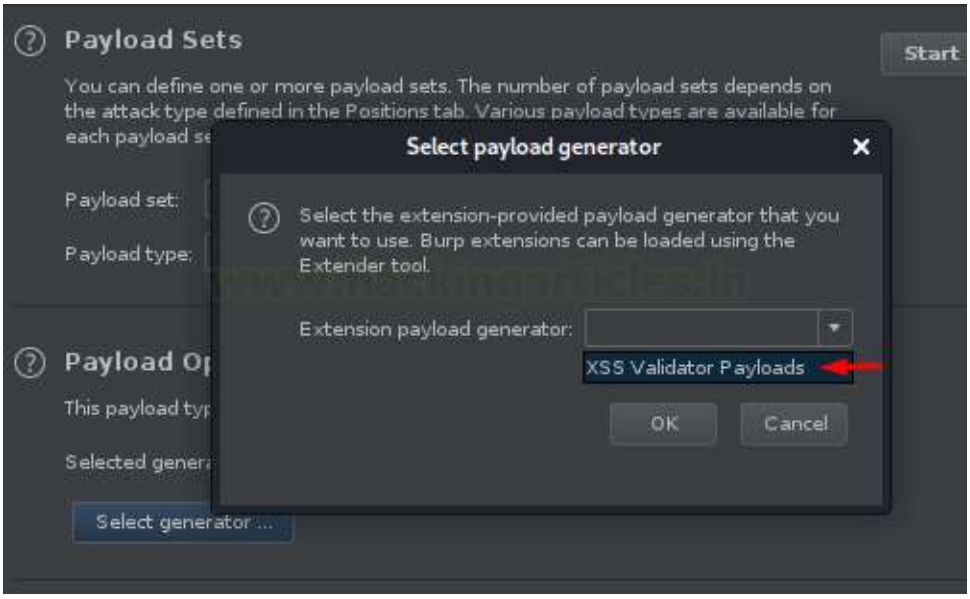
Time to set the payload position and the attack type, navigate right to **Positions tab**, select and hit the add button to set **"ignite"** as the injection point.



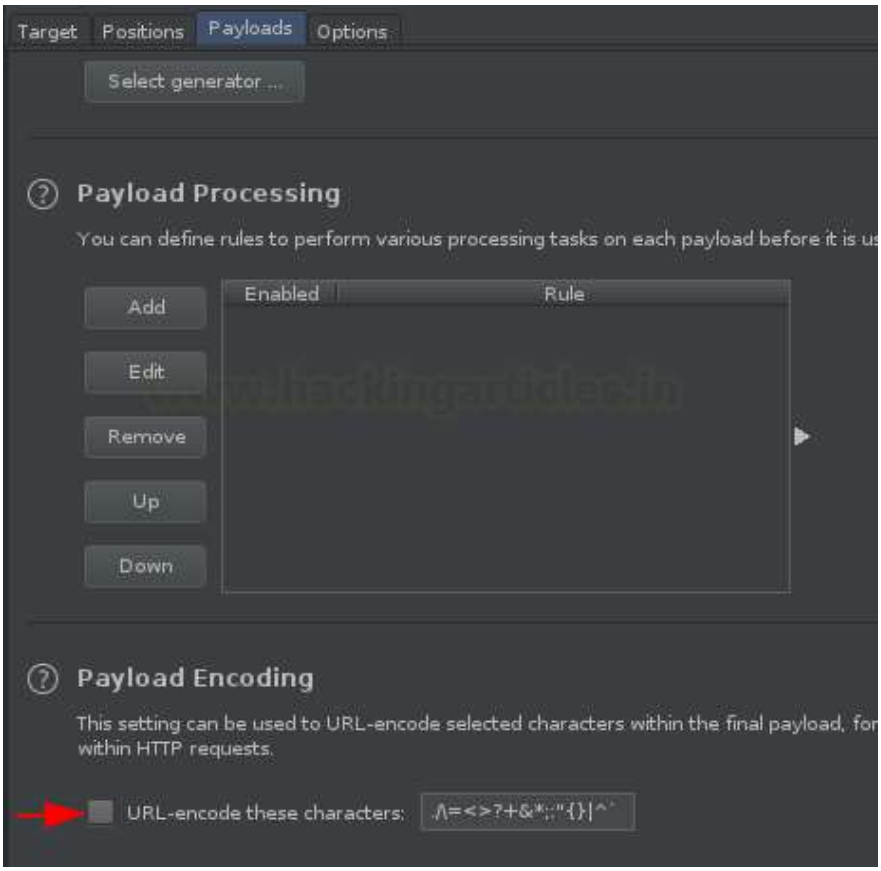
Now, here comes the most important part, rather than simply adding the payload file, we need to first set the payload type to **Extension-generated**



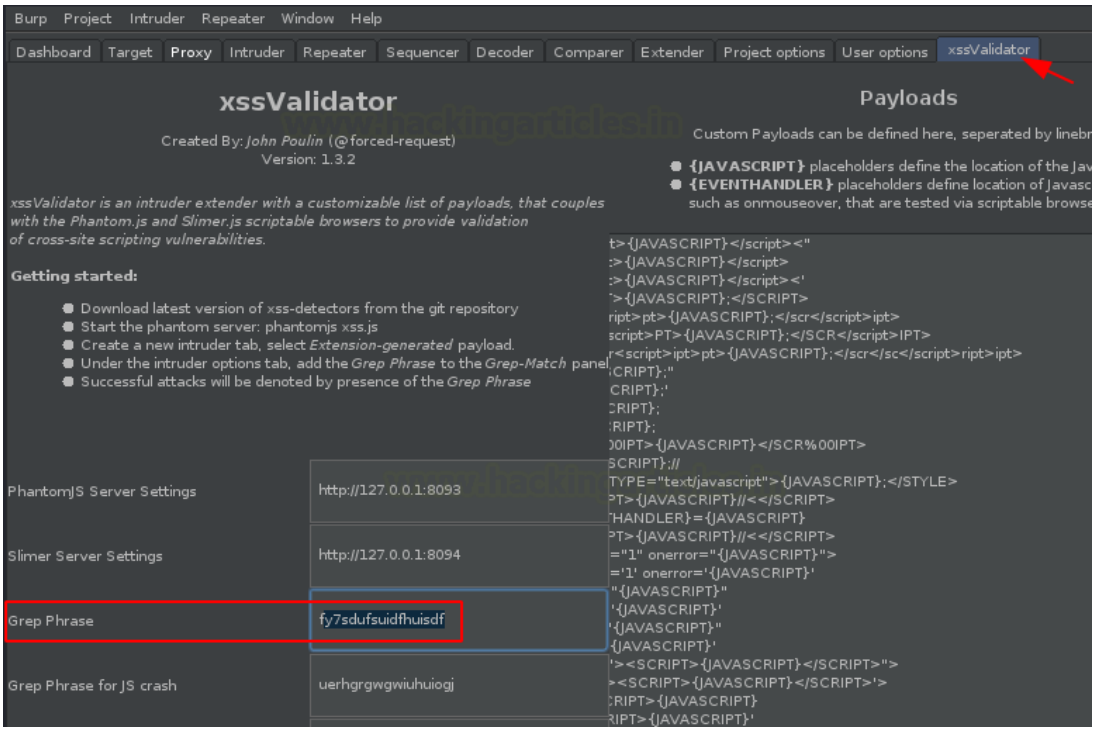
And with this, we now need to select our **generator** as **XSS Validator Payloads** from the Payload option.



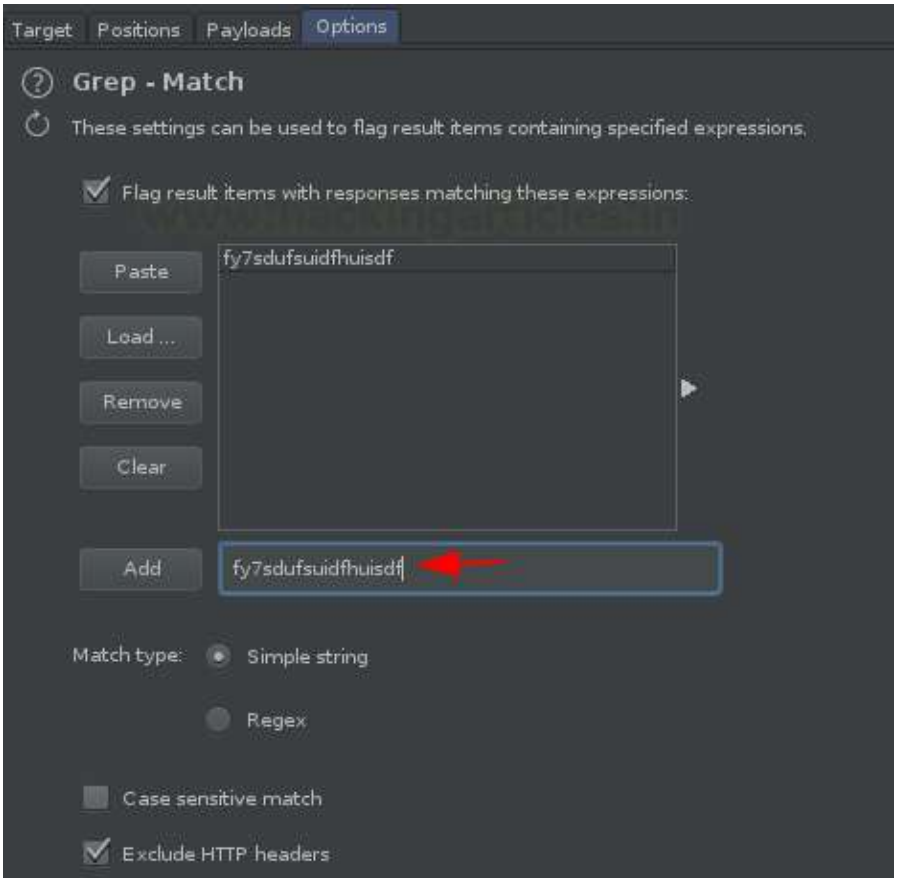
Let's uncheck the Payload Encoding option for this time.



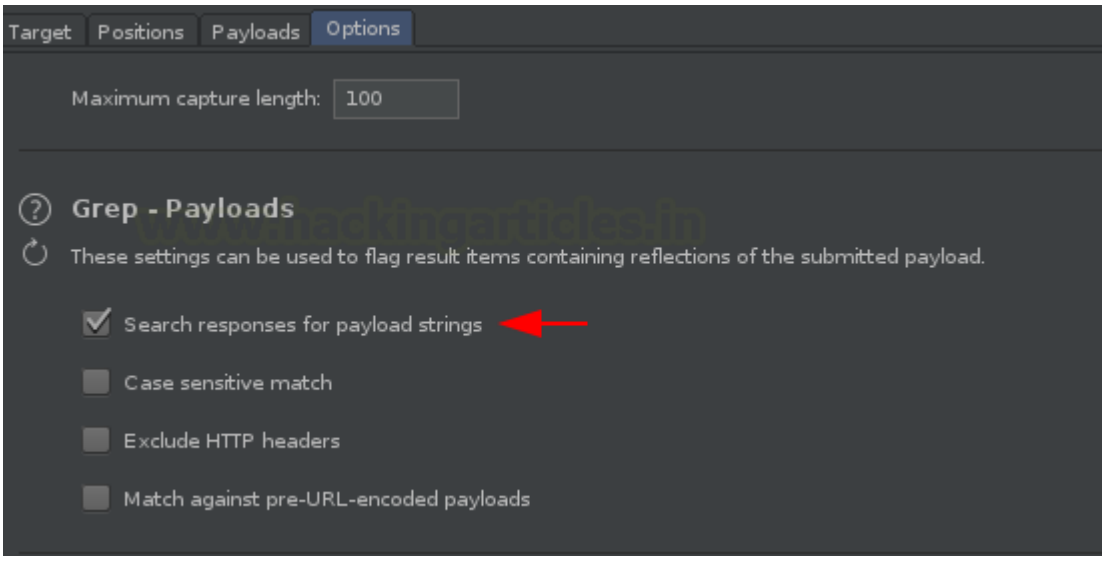
Now back from the XSS Validator, let's copy the **Grep Phrase,** that triggers back to every subsequent payload.



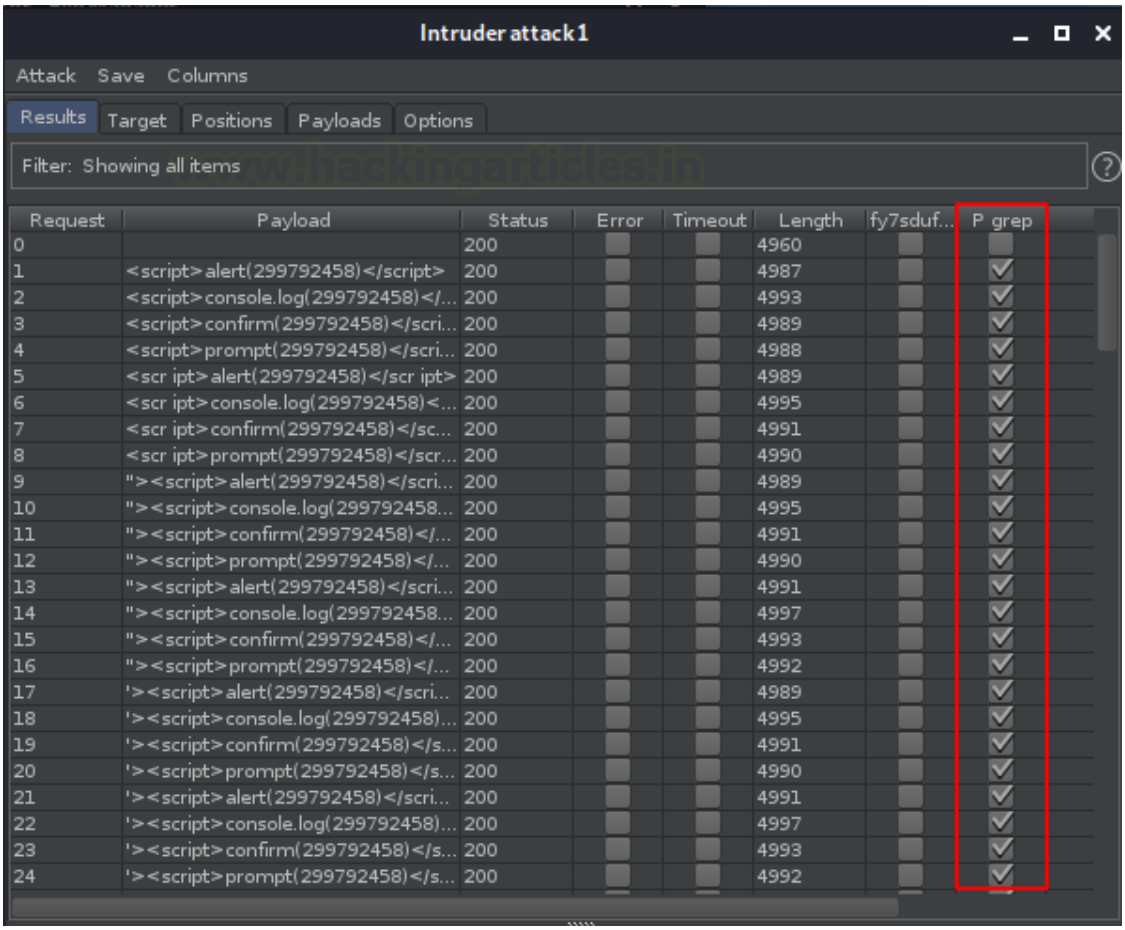
So, we are almost done, we just need to set copied phrase at the **Grep Match** in the **Options** tab to flag the result that **encounters a successful XSS**.



At last, Check the **"Search responses for payload strings"** box and fire up the **Attack** button.

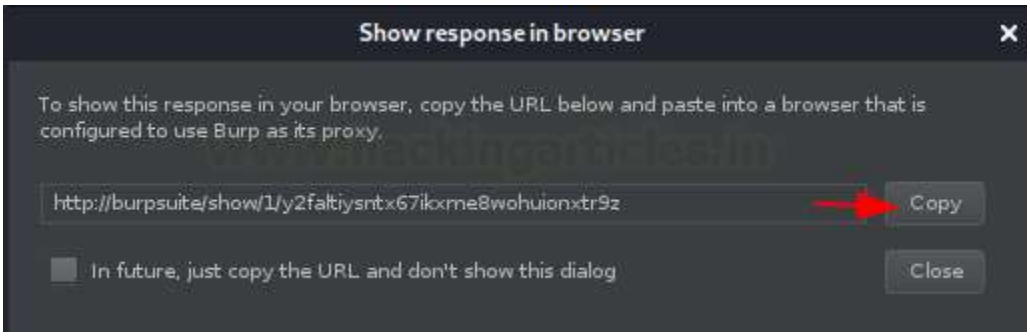


And there we go, from the below image you can see that almost all of our payloads got triggered out with a successful flag.

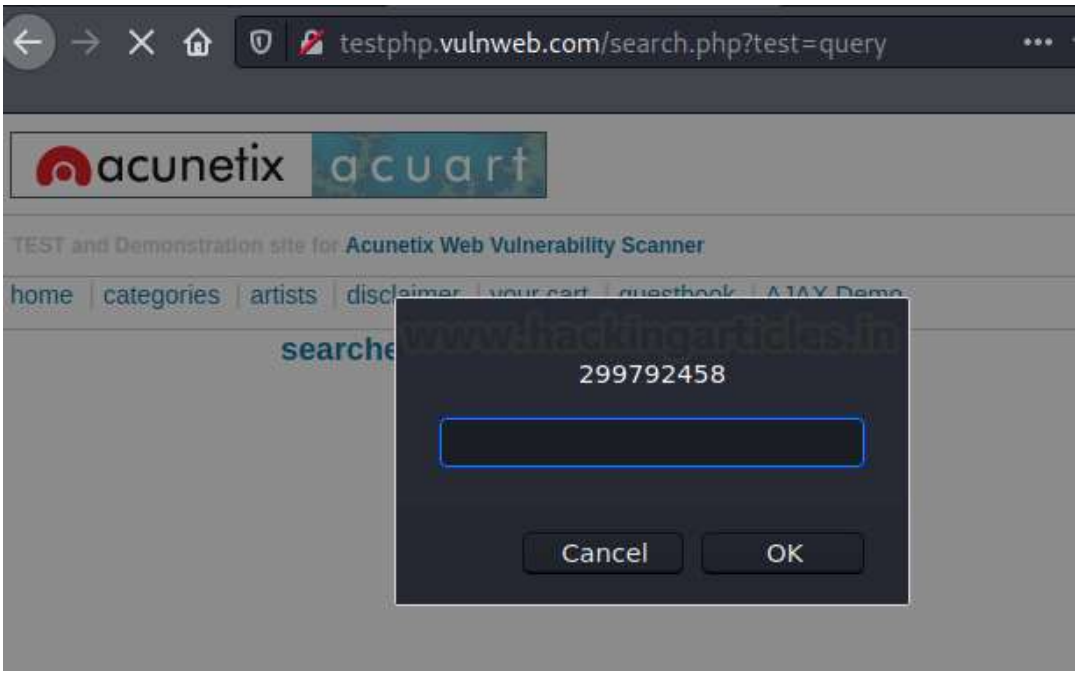


Request	Payload	Status	Error	Timeout	Length	fy7sduf...	P grep
0		200			4960		
1	<script>alert(299792458)</script>	200			4987		✓
2	<script>console.log(299792458)</...	200			4993		✓
3	<script>confirm(299792458)</scri...	200			4989		✓
4	<script>prompt(299792458)</scri...	200			4988		✓
5	<scr ipt>alert(299792458)</scr ipt>	200			4989		✓
6	<scr ipt>console.log(299792458)<...	200			4995		✓
7	<scr ipt>confirm(299792458)</sc...	200			4991		✓
8	<scr ipt>prompt(299792458)</scr...	200			4990		✓
9	"><script>alert(299792458)</scri...	200			4989		✓
10	"><script>console.log(299792458...	200			4995		✓
11	"><script>confirm(299792458)</...	200			4991		✓
12	"><script>prompt(299792458)</...	200			4990		✓
13	"><script>alert(299792458)</scri...	200			4991		✓
14	"><script>console.log(299792458...	200			4997		✓
15	"><script>confirm(299792458)</...	200			4993		✓
16	"><script>prompt(299792458)</...	200			4992		✓
17	'><script>alert(299792458)</scri...	200			4989		✓
18	'><script>console.log(299792458)...	200			4995		✓
19	'><script>confirm(299792458)</s...	200			4991		✓
20	'><script>prompt(299792458)</s...	200			4990		✓
21	'><script>alert(299792458)</scri...	200			4991		✓
22	'><script>console.log(299792458)...	200			4997		✓
23	'><script>confirm(299792458)</s...	200			4993		✓
24	'><script>prompt(299792458)</s...	200			4992		✓

To be more precise, let's check its output in the browser. **Right-click** on any **successful request**, opt for the option to **"Show response in the browser"** and **copy the generated URL** in thus pasting it in the browser.

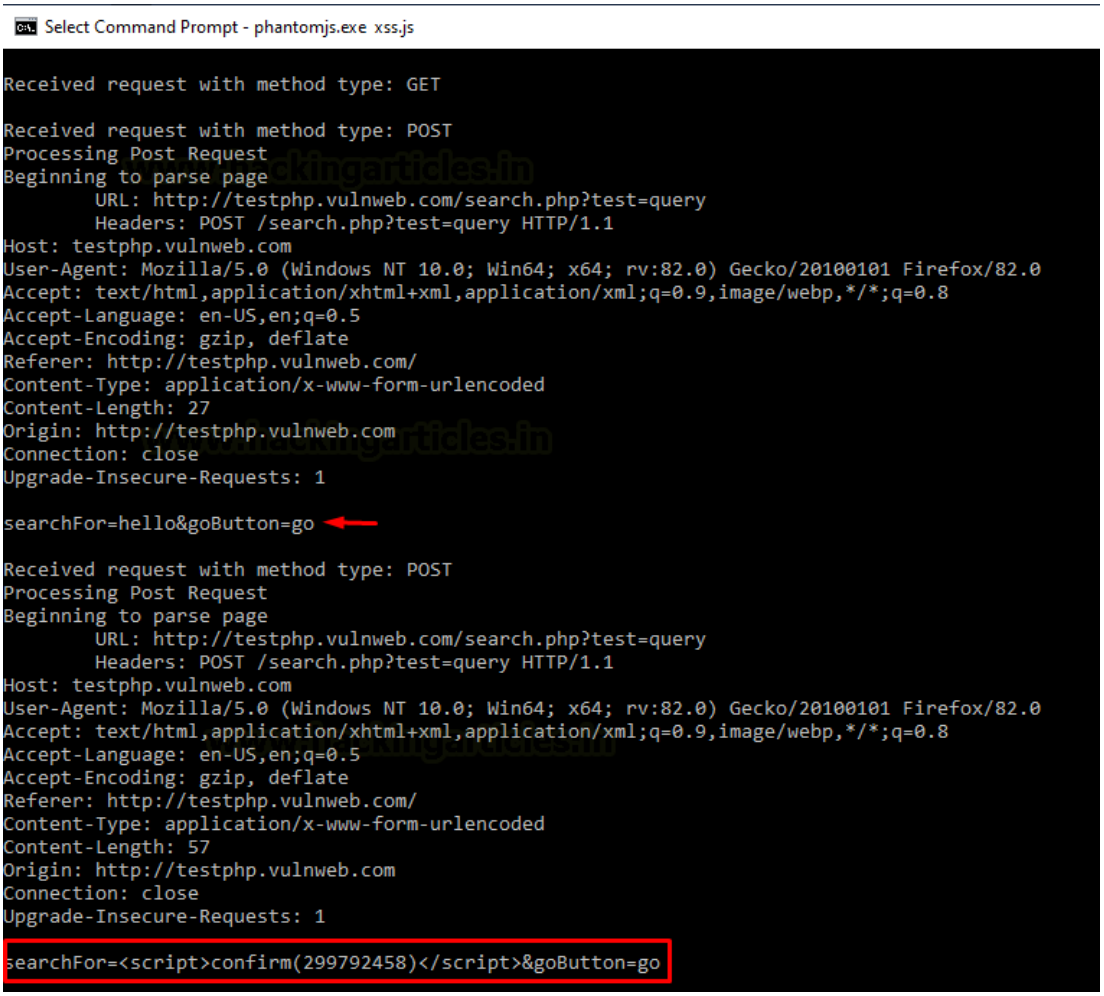


Great!! It's working, our payload hit at the correct destination.



Wait, but what about the phantom.js, did it capture something??

As soon as the attack starts up, the phantom.js tries to test every XSS injection with the HTTP requests shared by the XSS Validator.

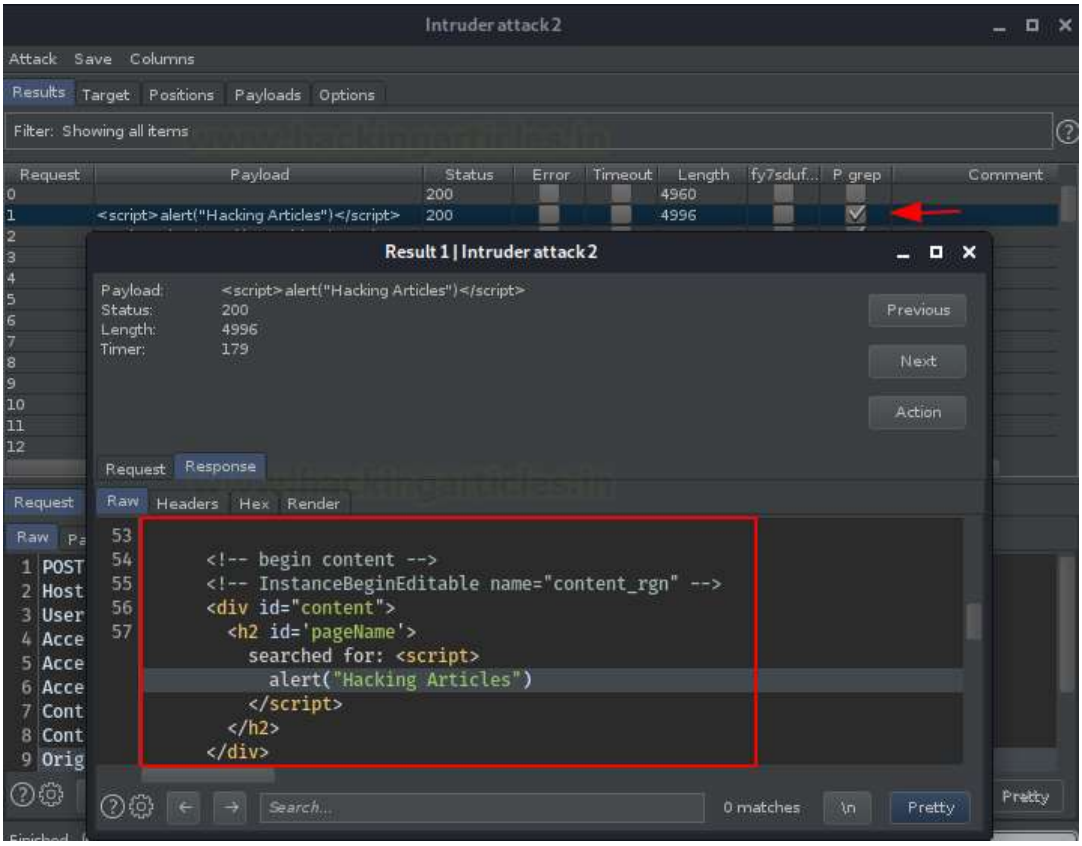


Customizing the Payload lists

Several payloads come pre-installed with the XSS validator, but what if, if we want to add our customized payload?? Yes, we can do so by simply **typing or by pasting the payload(s)** directly in the **Payload option** provided at the right-hand side of the extension.

Let's check whether this newly added **payload is triggering the grep phrase or not**. And for this, let's do the fuzzing again.

From the below screenshot, we can see that the grep phrase has been successfully triggered out by our payload.



Great!! From the below image, we can also see that the payload is executing as we desire.