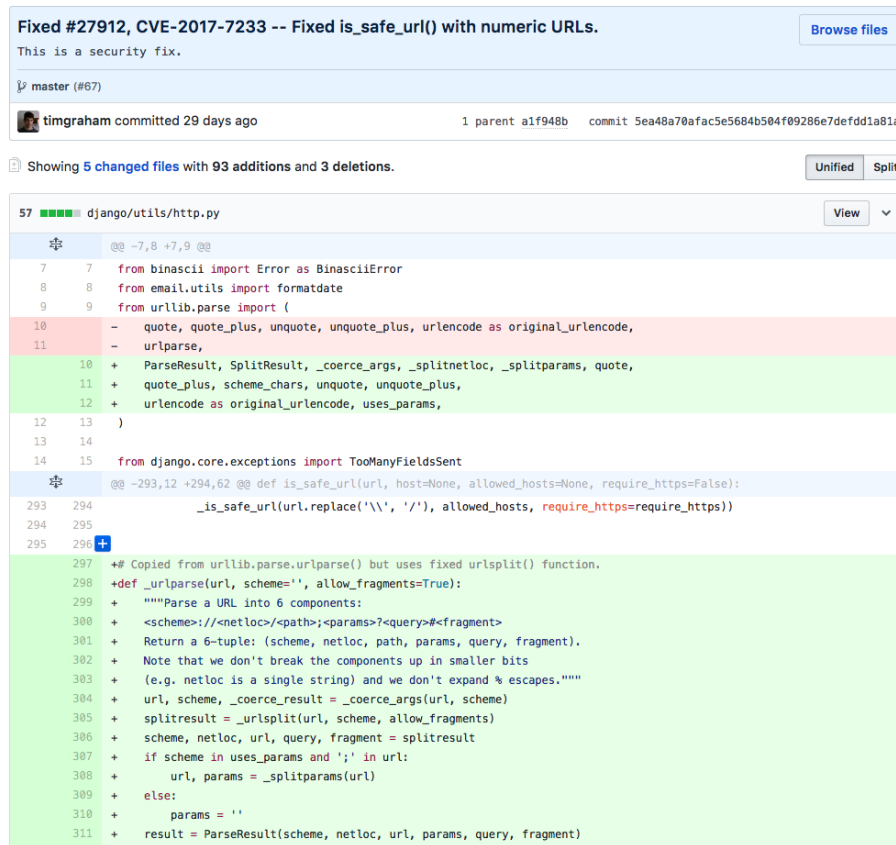Django两则CVE-2017-7233和CVE-2017-7234url跳转漏洞分析

nearg1e / 2017-06-19 07:15:47 / 浏览数 4855 安全技术 漏洞分析 顶(0) 踩(0)

Django官方News&Event在4月4日发布了一个安全更新，修复了两个URL跳转的漏洞，一个是urlparse的锅，另一个来自国内的安全研究员 phithon@长亭,都非常漂亮。因为有复现Django漏洞的习惯，晚上抽了点时间复现了一下。有趣的点还挺多。把两个漏洞的分析整合在一起，凑了篇文章。（还是研究漏洞

CVE-2017-7233分析 – Django is_safe_url() URL跳转过滤函数Bypass

国外安全研究员roks0n提供给Django官方的一个漏洞。

关于is_safe_url函数

Django自带一个函数：`django.utils.http.is_safe_url(url, host=None, allowed_hosts=None, require_https=False)`，用于过滤需要进行跳转的url。如果url安全则返回ture，不安全则返回false。文档如下：

```
print(is_safe_url.__doc__)

Return ``True`` if the url is a safe redirection (i.e. it doesn't point to
a different host and uses a safe scheme).

Always returns ``False`` on an empty url.

If ``require_https`` is ``True``, only 'https' will be considered a valid
scheme, as opposed to 'http' and 'https' with the default, ``False``.
```

让我们来看看常规的几个用法：

```
from django.utils.http import is_safe_url

In [2]: is_safe_url('http://baidu.com')
Out[2]: False

In [3]: is_safe_url('baidu.com')
Out[3]: True

In [5]: is_safe_url('aaaaa')
Out[5]: True
```

```
In [8]: is_safe_url('//blog.neargle.com')
Out[8]: False

In [7]: is_safe_url('http://google.com/adadadadad','blog.neargle.com')
Out[7]: False

In [13]: is_safe_url('http://blog.neargle.com/aaaa/bbb', 'blog.neargle.com')
Out[13]: True
```

可见在没有指定第二个参数host的情况下,url如果非相对路径，即`HttpResponseRedirect`函数会跳往别的站点的情况，`is_safe_url`就判断其为不安全的url,如果指定了

urllib.parse.urlparse的特殊情况

问题就出在该函数对域名和方法的判断，是基于`urllib.parse.urlparse`的,源码如下(django/utils/http.py):

```
def _is_safe_url(url, host):
if url.startswith('///'):
return False
url_info = urlparse(url)
if not url_info.netloc and url_info.scheme:
return False
if unicodedata.category(url[0])[0] == 'C':
return False
return ((not url_info.netloc or url_info.netloc == host) and
(not url_info.scheme or url_info.scheme in ['http', 'https']))
```

我们来看一下urlparse的常规用法及几种urlparse无法处理的特殊情况。

```
>>> urlparse('http://blog.neargle.com/2017/01/09/chrome-ext-spider-for-probe/')
ParseResult(scheme='http', netloc='blog.neargle.com', path='/2017/01/09/chrome-ext-spider-for-probe/', params='', query='', fr

>>> urlparse('ftp:99999999')
ParseResult(scheme='', netloc='', path='ftp:99999999', params='', query='', fragment='')

>>> urlparse('http:99999999')
ParseResult(scheme='http', netloc='', path='99999999', params='', query='', fragment='')

>>> urlparse('https:99999999')
ParseResult(scheme='', netloc='', path='https:99999999', params='', query='', fragment='')

>>> urlparse('javascript:222222')
ParseResult(scheme='', netloc='', path='javascript:222222', params='', query='', fragment='')

>>> urlparse('ftp:aaaaaaa')
ParseResult(scheme='ftp', netloc='', path='aaaaaaa', params='', query='', fragment='')

>>> urlparse('ftp:127.0.0.1')
ParseResult(scheme='ftp', netloc='', path='127.0.0.1', params='', query='', fragment='')

>>> urlparse('ftp:127.0.0.1')
ParseResult(scheme='ftp', netloc='', path='127.0.0.1', params='', query='', fragment='')
```

可以发现当scheme不等于http，且path为纯数字的时候,urlparse处理例如`aaaa:2222222223`的情况是不能正常分割开的，会全部归为path。这时`url_info.netloc`
`== url_info.scheme == ""`,则`((not url_info.netloc or url_info.netloc == host) and (not url_info.scheme or`
`url_info.scheme in ['http',`
`'https']))`为true。（这里顺便提一下,[django官方News&Event](#)中提到的poc:"http:99999999"是无法bypass的，在前面的判断`if not`
`url_info.netloc and url_info.scheme:`都过不了。）例如下面几种情况：

```
>>> is_safe_url('http:555555555')
False
>>> is_safe_url('ftp:23333333333')
True
>>> is_safe_url('https:2333333333')
True
```

使用IP Decimal Bypass is_safe_url

但是既然是url跳转漏洞，我们就需要让其跳转到指定的url里，`https:2333333333`这样的url明显是无法访问的，而冒号之后必须纯数字,`http:127.0.0.1`是无法pypass
`127.0.0.1 == 2130706433,8.8.8.8 ==`
134744072(转换器:[http://www.ipaddressguide.com/ip),而'http:2130706433'是在浏览器上是可以访问到对应的ip及服务的，即'http:2130706433,%E8%80%8C'http:2](#)
= [http://127.0.0.1/'。](#)

这里我们选用`https:1029415385`作为poc，这是一个google的ip,这个url可以bypass`is_safe_url`并跳转到google.com。

漏洞验证与影响

我们来写一个简单的环境:

```
from django.http import HttpResponseRedirect
from django.utils.http import is_safe_url

def BypassIsUrlSafeCheck(request):
url = request.GET.get("url", '')
if is_safe_url(url, host="blog.neargle.com"):
return HttpResponseRedirect(url)
else:
return HttpResponseRedirect('/')
```

然后访问:`http://127.0.0.1:8000/bypassIsUrlSafeCheck?url=https:1029415385`, 如图,url被重定向到了google.com。
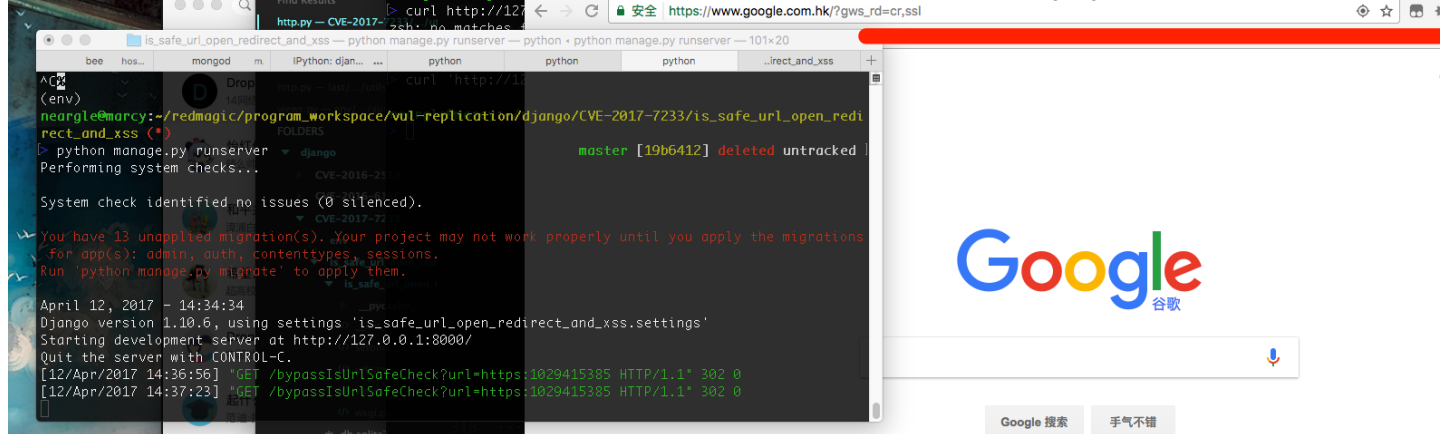
并非只有开发者自己使用`is_safe_url`会受到影响，Django默认自带的admin也使用了这个函数来处理next GET | POST参数，当用户访问`/admin/login/?next=https:1029415385`进行登录时，登录后同样会跳转到google.com,退出登录时同样使用到了该函数。

```
def _get_login_redirect_url(request, redirect_to):
### Ensure the user-originating redirection URL is safe.
if not is_safe_url(url=redirect_to, host=request.get_host()):
return resolve_url(settings.LOGIN_REDIRECT_URL)
return redirect_to

@never_cache
def login(request, template_name='registration/login.html',
redirect_field_name=REDIRECT_FIELD_NAME,
authentication_form=AuthenticationForm,
extra_context=None, redirect_authenticated_user=False):
......
return HttpResponseRedirect(_get_login_redirect_url(request, redirect_to))
......
```
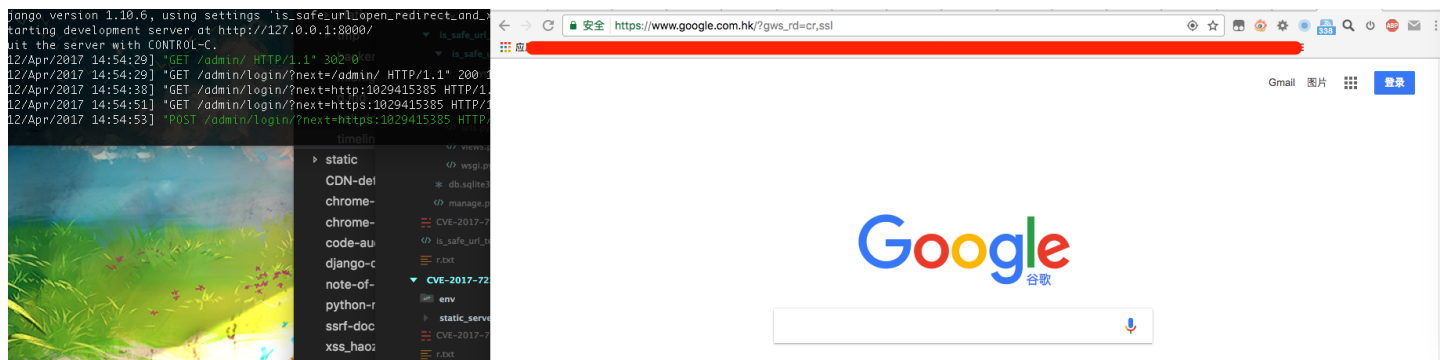

修复

django修复了代码，自己重构了一下`urlparse`函数,修复了`urlparse`函数的这个漏洞。

```
### Copied from urllib.parse.urlparse() but uses fixed urlsplit() function.
def _urlparse(url, scheme='', allow_fragments=True):
"""Parse a URL into 6 components:
```

```
<scheme>://<netloc>/<path>;<params>?<query>#<fragment>
Return a 6-tuple: (scheme, netloc, path, params, query, fragment).
Note that we don't break the components up in smaller bits
(e.g. netloc is a single string) and we don't expand % escapes."""
url, scheme, _coerce_result = _coerce_args(url, scheme)
splitresult = _urlsplit(url, scheme, allow_fragments)
scheme, netloc, url, query, fragment = splitresult
if scheme in uses_params and ';' in url:
url, params = _splitparams(url)
else:
params = ''
result = ParseResult(scheme, netloc, url, params, query, fragment)
return _coerce_result(result)
```

关于官方提到的 possible XSS attack

[django官方News&Event](#)中提到的这个漏洞可能会产生XSS，我认为除非程序员把接受跳转的url插入的到<script
type="text/javascript"
src=""></script>等特殊情况之外，直接使用产生XSS的场景还是比较少的。如果你想到了其他的场景还请赐教，祝好。

CVE-2017-7234 django.views.static.serve url跳转漏洞

漏洞详情

来自 @Phithon 的一个漏洞。

问题出现在：`django.views.static.serve()`函数上。该函数可以用来指定web站点的静态文件目录。如:

```
urlpatterns = [
url(r'^admin/', admin.site.urls),
url(r'^staticp/(?P<path>.*)$', serve, {'document_root': os.path.join(settings.BASE_DIR, 'staticpath')})
]
```

这样django项目根目录下staticpath中的所有文件，就可以在staticp/目录中访问。e.g. `http://127.0.0.1:8000/staticp/test.css`

这种方法是不被django官方推荐在生成环境使用的，对安全性和性能都有一定影响。

问题代码如下 (django/views/static.py)：

```
path = posixpath.normpath(unquote(path))
path = path.lstrip('/')
newpath = ''
for part in path.split('/'):
if not part:
### Strip empty path components.
continue
drive, part = os.path.splitdrive(part)
head, part = os.path.split(part)
if part in (os.curdir, os.pardir):
### Strip '.' and '..' in path.
continue
newpath = os.path.join(newpath, part).replace('\\', '/')
if newpath and path != newpath:
return HttpResponseRedirect(newpath)
```

path既我们传入的路径,如果传入的路径为`staticp/path.css`,则`path=path.css`。跟踪代码可知，path经过了unquote进行url解码，后来又`replace('\\'`
`'/')`,进入HttpResponseRedirect，很诡异的逻辑看起来很有问题。一般遇到这类型的函数我们会先试着找看看,任意文件读漏洞，但是这个对'.'和&

我们的最终目的是`HttpResponseRedirect('//evil.neargle.com')`或者`HttpResponseRedirect('[http://evil.neargle.com](http://evil.neargle.com)')`,那么需
`!=`
`newpath`,那么path里面就必须带有'\',好的现在的我们传入`'/staticp/%5C%5Cblog.neargle.com'`,则`path='\\\\blog.neargle.com'newpath`

修复

```
11     from django.http import (                              11     from django.http import (
12         FileResponse, Http404, HttpResponse, HttpResponseNotModified,   12         FileResponse, Http404, HttpResponse, HttpResponseNotModified,
13  -      HttpResponseRedirect,
14  )                                                          13  )
15     from django.template import Context, Engine, TemplateDoesNotExist, loader   14     from django.template import Context, Engine, TemplateDoesNotExist, loader
                                                                15  +from django.utils._os import safe_join
16     from django.utils.http import http_date, parse_http_date   16     from django.utils.http import http_date, parse_http_date
17     from django.utils.translation import gettext as _, gettext_lazy   17     from django.utils.translation import gettext as _, gettext_lazy
                                                                18
⊹   @@ -33,25 +33,11 @@ def serve(request, path, document_root=None, show_indexes=False):
33         but if you'd like to override it, you can create a template called   33         but if you'd like to override it, you can create a template called
34         ``static/directory_index.html``.                   34         ``static/directory_index.html``.
35         """                                                 35         """
36  -      path = posixpath.normpath(path)                     36  +      path = posixpath.normpath(path).lstrip('/')
37  -      path = path.lstrip('/')                             37  +      fullpath = safe_join(document_root, path)
38  -      newpath = ''
39  -      for part in path.split('/'):
40  -          if not part:
41  -              # Strip empty path components.
42  -              continue
43  -          drive, part = os.path.splitdrive(part)
44  -          head, part = os.path.split(part)
45  -          if part in (os.curdir, os.pardir):
46  -              # Strip '.' and '..' in path.
47  -              continue
48  -          newpath = os.path.join(newpath, part).replace('\\', '/')
49  -      if newpath and path != newpath:
50  -          return HttpResponseRedirect(newpath)
51  -      fullpath = os.path.join(document_root, newpath)
52         if os.path.isdir(fullpath):                         38         if os.path.isdir(fullpath):
53             if show_indexes:                                 39             if show_indexes:
54  -              return directory_index(newpath, fullpath)    40  +              return directory_index(path, fullpath)
55             raise Http404(_("Directory indexes are not allowed here."))   41             raise Http404(_("Directory indexes are not allowed here."))
56         if not os.path.exists(fullpath):                    42         if not os.path.exists(fullpath):
57             raise Http404(_('"%(path)s" does not exist') % {'path': fullpath})   43             raise Http404(_('"%(path)s" does not exist') % {'path': fullpath})
⊹
```

嗯，官方表示自己也不知道为什么要写这串代码，删了这一串代码然后用safe_url函数代替。

ps.

浏览器不仅仅支持十进制来代替点分十进制的IP，也可以使用十六进制和8进制来代替。`http://`■■■■■ == `http://`■■■ == `http://0x`■■■■ == `http://0`■■■(例如:`http://127.0.0.1` == `http://2130706433` == `http://0x7F000001` == `http://017700000001`),十六进制非纯数字所以不可用来bypass urlparse，但是八进制还是可以的。

urls

- https://github.com/django/django/commit/5ea48a70afac5e5684b504f09286e7defdd1a81a
- https://www.djangoproject.com/weblog/2017/apr/04/security-releases/
- https://docs.python.org/3/library/urllib.parse.html

点击收藏 | 0 关注 | 0

1. 1 条回复

admln 2017-06-20 10:06:06

不错给楼主点个赞

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](#)

[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](#)