SUCTF2019，python源码分析，漏洞原理

fz41 / 2019-08-31 09:57:00 / 浏览数 3304 安全技术 CTF 顶(0) 踩(0)

前言

前段时间打的SUCTF2019中有一个题目叫Pythongin思路大概来源于黑帽大会

https://i.blackhat.com/USA-19/Thursday/us-19-Birch-HostSplit-Exploitable-Antipatterns-In-Unicode-Normalization.pdf

不怎么明白漏洞的原理，就准备复现一下，在复现的过程中出现了很多坑，来记录一下。

踩坑过程

整个SUCTF2019的源码都已经开源了，地址如下

https://github.com/team-su/SUCTF-2019

具体题目的分析过程就不再赘述了，感觉师傅们分析的一个比一个详细,我在文末也放了几个师傅的分析的writeup

搭建好docker的环境，直接按照文档中的命令可以直接复现成功

```
docker build -t dockerflask .

docker run -p 3000:80 dockerflask
open http://localhost:3000
```

然后按照题目的payload可以直接复现成功

然而问题来了，我比较懒，我是直接把文件放到了我的sublime 然后，使用一个官方的payload
居然没有打成功怀疑是我的环境和编辑器的编码出现了问题，然后放到我的WSL系统里面运行（这里说一句题外话，最近刚刚给自己的电脑安装了WSL win下的ubuntu，感觉很好用 ，想用linux的时候不必再去开虚拟机了）大家可以去尝试一下，同时也顺便美化一下自己的终端。

再win下的环境是

在linux下的环境是

```
/mnt/d/CTF/SUCTF ■ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

在win下的环境是

```
C:\Users\11466>python3
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

可以看版本不一样，同时python最后更新的时间也不一样问题就出现在这里了。

对源码做出的简单的修改，用来测试payload

```
def getUrl2(url):
   host = parse.urlparse(url).hostname
   if host == 'suctf.cc':
       return "■■ your problem? 111"
   parts = list(urlsplit(url))
   host = parts[1]
   if host == 'suctf.cc':
       return "■■ your problem? 222 " + host
   newhost = []
   for h in host.split('.'):
       newhost.append(h.encode('idna').decode('utf-8'))
   parts[1] = '.'.join(newhost)
   #■■ url ■■■■
   finalUrl = urlunsplit(parts).split(' ')[0]
   host = parse.urlparse(finalUrl).hostname
```

```
   if host == 'suctf.cc':
       return "success"
   else:
       return "■■ your problem? 333"
if __name__=="__main__":
   # get_unicode()
   # try:
   #     print_unicode()
   # except:

   #     print("something_error")
   url = "file://suctf.c■sr%2ffffffflag @111"
   print(url)
   print(getUrl2(url))
   # print(getUrl(url))
   # get_unicode()
```

可以出运行不同的结果：

在win下

```
file://suctf.c■sr%2ffffffflag @111
Traceback (most recent call last):
 File "1.py", line 72, in <module>
   print(getUrl2(url))
 File "1.py", line 45, in getUrl2
   host = parse.urlparse(url).hostname
 File "E:\python3\lib\urllib\parse.py", line 368, in urlparse
   splitresult = urlsplit(url, scheme, allow_fragments)
 File "E:\python3\lib\urllib\parse.py", line 461, in urlsplit
   _checknetloc(netloc)
 File "E:\python3\lib\urllib\parse.py", line 407, in _checknetloc
   "characters under NFKC normalization")
ValueError: netloc 'suctf.cc/usr%2ffffffflag @111' contains invalid characters under NFKC normalization
```

在WSL下

```
/mnt/d/CTF/NUCA ■ python3 1.py
file://suctf.c■sr%2ffffffflag @111
success
```

原来没怎么分析过python的源码

但是从报错信息上可以找到，问题就出现在`python3\lib\urllib\parse.py`

于是就来简单分析了下parse.py的源码

源码对比

在win下是比较新的一个python，很明显对于这类漏洞已经修补

在WSL下的是一个比较就得python

使用在win下找到`E:\python3\lib\urllib\parse.py`

在WSL下找到 `/usr/lib/python3.6/urllib/parse.py`

```
/mnt/d/CTF/SUCTF  python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path
['', '/usr/lib/python36.zip', '/usr/lib/python3.6', '/usr/lib/python3.6/lib-dynload', '/home/fangzhang/.local/lib/python3.6/si
>>>
/mnt/d/CTF/SUCTF cd /usr/lib/python3.6/urllib/
/usr/lib/python3.6/urllib
```

使用文本对比工具

得到一下结果：

```
--- E:\python3\Lib\urllib\parse.py
+++ /usr/lib/python3.6/urllib
@@ -390,21 +390,6 @@
        if wdelim >= 0:                    # if found
            delim = min(delim, wdelim)     # use earliest delim position
    return url[start:delim], url[delim:]   # return (domain, rest)
-
-def _checknetloc(netloc):
-    if not netloc or netloc.isascii():
-        return
-    # looking for characters like \u2100 that expand to 'a/c'
-    # IDNA uses NFKC equivalence, so normalize for this check
-    import unicodedata
-    netloc2 = unicodedata.normalize('NFKC', netloc)
-    if netloc == netloc2:
-        return
-    _, _, netloc = netloc.rpartition('@') # anything to the left of '@' is okay
-    for c in '/?#@:':
-        if c in netloc2:
-            raise ValueError("netloc '" + netloc2 + "' contains invalid " +
-                             "characters under NFKC normalization")

def urlsplit(url, scheme='', allow_fragments=True):
    """Parse a URL into 5 components:
@@ -424,6 +409,7 @@
    i = url.find(':')
    if i > 0:
        if url[:i] == 'http': # optimize the common case
+            scheme = url[:i].lower()
            url = url[i+1:]
            if url[:2] == '//':
                netloc, url = _splitnetloc(url, 2)
@@ -434,8 +420,7 @@
                url, fragment = url.split('#', 1)
            if '?' in url:
                url, query = url.split('?', 1)
-            _checknetloc(netloc)
-            v = SplitResult('http', netloc, url, query, fragment)
+            v = SplitResult(scheme, netloc, url, query, fragment)
            _parse_cache[key] = v
            return _coerce_result(v)
        for c in url[:i]:
@@ -458,7 +443,6 @@
        url, fragment = url.split('#', 1)
    if '?' in url:
        url, query = url.split('?', 1)
-    _checknetloc(netloc)
    v = SplitResult(scheme, netloc, url, query, fragment)
    _parse_cache[key] = v
    return _coerce_result(v)
@@ -600,7 +584,7 @@
    # if the function is never called
    global _hextobyte
    if _hextobyte is None:
-        _hextobyte = {(a + b).encode(): bytes.fromhex(a + b)
+        _hextobyte = {(a + b).encode(): bytes([int(a + b, 16)])
                      for a in _hexdig for b in _hexdig}
    for item in bits[1:]:
        try:
@@ -750,7 +734,7 @@
_ALWAYS_SAFE = frozenset(b'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
                         b'abcdefghijklmnopqrstuvwxyz'
                         b'0123456789'
-                         b'_.-~')
+                         b'_.-')
_ALWAYS_SAFE_BYTES = bytes(_ALWAYS_SAFE)
_safe_quoters = {}

@@ -782,17 +766,14 @@
```

```
    Each part of a URL, e.g. the path info, the query, etc., has a
    different set of reserved characters that must be quoted.

-    RFC 3986 Uniform Resource Identifiers (URI): Generic Syntax lists
+    RFC 2396 Uniform Resource Identifiers (URI): Generic Syntax lists
     the following reserved characters.

     reserved    = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" |
-                  "$" | "," | "~"
+                  "$" | ","

     Each of these characters is reserved in some component of a URL,
     but not necessarily in all of them.
-
-    Python 3.7 updates from using RFC 2396 to RFC 3986 to quote URL strings.
-    Now, "~" is included in the set of reserved characters.

     By default, the quote function is intended for quoting the path
     section of a URL.  Thus, it will not encode '/'.  This character
```

可以明显得看出，主要是多了一个处理得函数

```
-def _checknetloc(netloc):
-    if not netloc or netloc.isascii():
-        return
-    # looking for characters like \u2100 that expand to 'a/c'
-    # IDNA uses NFKC equivalence, so normalize for this check
-    import unicodedata
-    netloc2 = unicodedata.normalize('NFKC', netloc)
-    if netloc == netloc2:
-        return
-    _, _, netloc = netloc.rpartition('@') # anything to the left of '@' is okay
-    for c in '/?#@:':
-        if c in netloc2:
-            raise ValueError("netloc '" + netloc2 + "' contains invalid " +
-                             "characters under NFKC normalization")
```

同时也可以看出，这次主要更新得地方。

如下对上面得代码进行分析

源码分析

unicode规范化处理

如下引用一下https://python3-cookbook.readthedocs.io

关于unicode得规范化处理，有如下说明

```
unicode■■■■■■■■■■■■■■■■■■■■■■■■
   NFC
   Unicode ■■■■■■ C■■■■■■■ normalization-type■■■■■■■ Unicode ■■■■■
   NFD
   Unicode ■■■■■■ D■
   NFKC
   Unicode ■■■■■■ KC■
   NFKD
   Unicode ■■■■■■ KD■
```

在Unicode中，某些字符能够用多个合法的编码表示。为了说明，考虑下面的这个例子：

```
>>> s1 = 'Spicy Jalape\u00f1o'
>>> s2 = 'Spicy Jalapen\u0303o'
>>> s1
'Spicy Jalapeño'
>>> s2
'Spicy Jalapeño'
>>> s1 == s2
False
>>> len(s1)
```

```
14
>>> len(s2)
15
>>>
```

这里的文本"Spicy Jalapeño"使用了两种形式来表示。
第一种使用整体字符"ñ"(U+00F1)，第二种使用拉丁字母"n"后面跟一个"~"的组合字符(U+0303)。

在需要比较字符串的程序中使用字符的多种表示会产生问题。
为了修正这个问题，你可以使用unicodedata模块先将文本标准化：

```
>>> import unicodedata
>>> t1 = unicodedata.normalize('NFC', s1)
>>> t2 = unicodedata.normalize('NFC', s2)
>>> t1 == t2
True
>>> print(ascii(t1))
'Spicy Jalape\xf1o'
>>> t3 = unicodedata.normalize('NFD', s1)
>>> t4 = unicodedata.normalize('NFD', s2)
>>> t3 == t4
True
>>> print(ascii(t3))
'Spicy Jalapen\u0303o'
>>>
```

`normalize()` 第一个参数指定字符串标准化的方式。
NFC表示字符应该是整体组成(比如可能的话就使用单一编码)，而NFD表示字符应该分解为多个组合字符表示。

Python同样支持扩展的标准化形式NFKC和NFKD，它们在处理某些字符的时候增加了额外的兼容特性。比如：

```
>>> s = '\ufb01' # A single character
>>> s
'■'
>>> unicodedata.normalize('NFD', s)
'■'
# Notice how the combined letters are broken apart here
>>> unicodedata.normalize('NFKD', s)
'fi'
>>> unicodedata.normalize('NFKC', s)
'fi'
>>>
```

漏洞分析

根据以上分析

主要的修复方式就是通过对url中的unicode进行规范化处理了，现在通过具体的例子来分析一哈。

```
import unicodedata
netloc2 = unicodedata.normalize('NFKC', netloc)
if netloc == netloc2:
   return
```

用我们的WSL的环境(也就是没有打上补丁的环境)进行测试

```
>>> from urllib.parse import urlsplit
>>> u = "https://example.com\uFF03@bing.com"
#■■■■■■■
>>> SplitResult(scheme='https', netloc='example.com■@bing.com', path='', query='', fragment='')
#■■■■■■■■■
>>>import unicodedata
>>>u2 = unicodedata.normalize('NFKC', u)
>>> urlsplit(u2)
SplitResult(scheme='https', netloc='example.com', path='', query='', fragment='@bing.com')
#■■■■■■■■■■
>>>u3 = u.encode("idna").decode("ascii")
>>> urlsplit(u3)
SplitResult(scheme='https', netloc='example.com', path='', query='', fragment='@bing.com')
```

以上就是漏洞的原理，不同的编码经处理之后，经过urlsplit() 处理之后，得到的的netloc是不一样的

IDNA（Internationalizing Domain Names in Applications）IDNA是一种以标准方式处理ASCII以外字符的一种机制，它从unicode中提取字符，并允许非ASCII码字符以允许使用的ASCII字符表示。

unicode转ASCII发生在IDNA中的TOASCII操作中。如果能通过TOASCII转换时，将会以正常的字符呈现。而如果不能通过TOASCII转换时，就会使用“ACE标签”，“ACE

所以在新的urlsplit函数中会增加一个判断，如果规范化处理的结果和原来的结果一样，才能返回真确的值。

题目分析

```
def getUrl():
    url = request.args.get("url")
    host = parse.urlparse(url).hostname
    if host == 'suctf.cc':
        return "■■ your problem? 111"
    parts = list(urlsplit(url))
    host = parts[1]
    if host == 'suctf.cc':
        return "■■ your problem? 222 " + host
    newhost = []
    for h in host.split('.'):
        newhost.append(h.encode('idna').decode('utf-8'))
    parts[1] = '.'.join(newhost)
    #■■ url ■■■■
    finalUrl = urlunsplit(parts).split(' ')[0]
    host = parse.urlparse(finalUrl).hostname
    if host == 'suctf.cc':
        return urllib.request.urlopen(finalUrl, timeout=2).read()
    else:
        return "■■ your problem? 333"

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80)
```

根据以上分析，题目就比较简单了，只需要满足hostname`encode('idna').decode('utf-8'))` 处理之前不是`suctf.cc` 处理之后是`suctf.cc`就好了

后记

漏洞不难理解，只是觉得应该记录一下

看到了一句话,摘自某位大佬：

> 如果翻译器对程序进行了彻底的分析而非某种机械的变换, 而且生成的中间程序与源程序之间已经没有很强的相似性, 我们就认为这个语言是编译的.
> 彻底的分析和非平凡的变换, 是编译方式的标志性特征.
>
> 如果你对知识进行了彻底的分析而非某种机械的套弄, 在你脑中生成的概念与生硬的文字之间已经没有很强的相似性, 我们就认为这个概念是被理解的.
> 彻底的分析和非凡的变换, 是获得真知的标志性特征.

与君共勉。

参考链接

参考很多链接，不过我觉得，遇到问题看官方的文档和源码更有效果

https://xz.aliyun.com/t/6042#toc-29

https://www.anquanke.com/post/id/184858#h3-13

https://i.blackhat.com/USA-19/Thursday/us-19-Birch-HostSplit-Exploitable-Antipatterns-In-Unicode-Normalization.pdf

https://bugs.python.org/issue36216

https://python3-cookbook.readthedocs.io/zh_CN/latest/c02/p09_normalize_unicode_text_to_regexp.html

上一篇：DNS安全皮毛（二） 下一篇：浅谈ctf中phpinfo需要关注的点
1. 0 条回复
   - 动动手指，沙发就是你的了！

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