PyConChina 2019

编写安全的Python代码

邓良驹

2019.10.19

思考题

```
if user.balance >= product.price:
    user.balance -= product.price
```

?

目录 CONTENTS

常见不安全代码

小心 eval

```
import sys

def run(s):
    try:
        v = eval(s)
    except Exception as e:
        print(e)

if __name__ == "__main__":
    run(sys.argv[1])
```

```
└─○ python test_eval.py "__import__('os').system('/bin/sh')"
sh-3.2$
sh-3.2$
```

```
└─○ python test_eval.py "print(globals())"
{'__name__': '__main__', '__doc__': None, '__package__': No
>, '__spec__': None, '__annotations__': {}, '__builtins__':
s': <module 'sys' (built-in)>, 'run': <function run at 0x10
```

还可以构造更复杂的输入使Python进程产生段错误，从产生一个core dump文件。

应对：

在生产环境中，任何情况下都不要使用eval。

小心 input()

```
secret = "123456"

def sign_up():
    name = input("please enter your
name:")
    print("Your name is: ", name)

if __name__ == "__main__":
    sign_up()
```

```
└─○ python test_input.py
please enter your name:secret
('Your name is: ', '123456')
```

```
└─○ python test_input.py
please enter your name:globals()
('Your name is: ', {'__builtins__': <module '__
_name__': '__main__', 'sign_up': <function sign
```

应对：
使用较新版本的Python3。Python2中可改用raw_input。

小心类型溢出

```
>>> range(100**100)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OverflowError: range() result has too many items
```

```
>>> range(100**100)
range(0, 10000000000…)
```

```
>>> xrange(100**100)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OverflowError: Python int too large to convert to C long
```

https://www.cvedetails.com/cve/CVE-2017-1000158/

应对：

使用较新版本的Python3，而不使用发行版OS自带的旧版Python。捕获并处理溢出错误，可以减少风险。在重要的位置做好防御式编程，检查好入参的类型与合法的上下限。

小心 assert

```
def do_sth(request, user):
    assert user.is_admin, "Permission denied!"
    user.balance += 100
```

应对：

Assert 语句不应用于业务逻辑条件检查，只应用于程序员之间的沟通，如单元测试、数据边界检查、API调用约束说明等。

小心 pickle

```
import os
import pickle

class ShellExp:
    def __reduce__(self):
        return (os.system, 'ls -a / &&
/bin/sh',))

if __name__ == "__main__":
    shellcode = pickle.dumps(ShellExp())
    pickle.loads(shellcode)
```

应对：
绝不对不可信/未认证数据进行unpickle，使用更安全的JSON或YAML做序列化。必须使用pickle时在沙盒环境执行。

小心 PyYAML

```
import yaml

class ShellExp:
    def __init__(self, args):
        import os
        os.system(args)

if __name__ == "__main__":
    payload =
"!!python/object/apply:__main__.ShellExp ['/bin/sh']"
    yaml.load(payload)
```

```
└─○ python test_yaml.py
test_yaml.py:14: YAMLLoadWarning: ca
g.pyyaml.org/load for full details.
  yaml.load(payload)
sh-3.2$
sh-3.2$
```

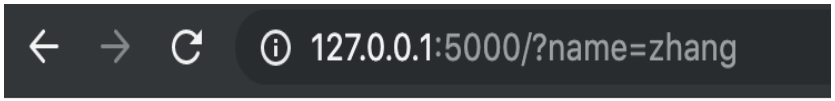
应对：

使用yaml.safe_load，必要时编写自定义 Loader 做更严格的检查。

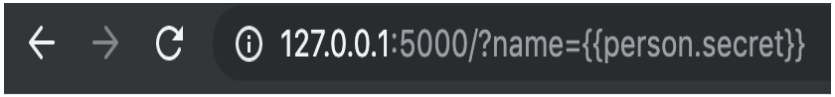
对不可信来源的序列化检查后操作。

小心 服务端模板注入(SSTI)

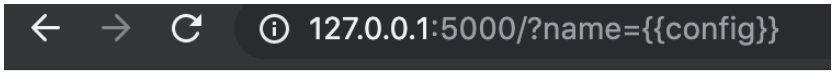
```
from flask import Flask
from flask request, render_template_string

app = Flask(__name__)

@app.route('/')
def hello():
    name = request.args.get("name", "World")
    person = {"name": name, "secret": "123456"}
    tpl = "<p>Hello, %s." % name
    return render_template_string(tpl,
person=person)

if __name__ == "__main__":
    app.run(debug=True
```



Hello, zhang.



Hello, 123456.

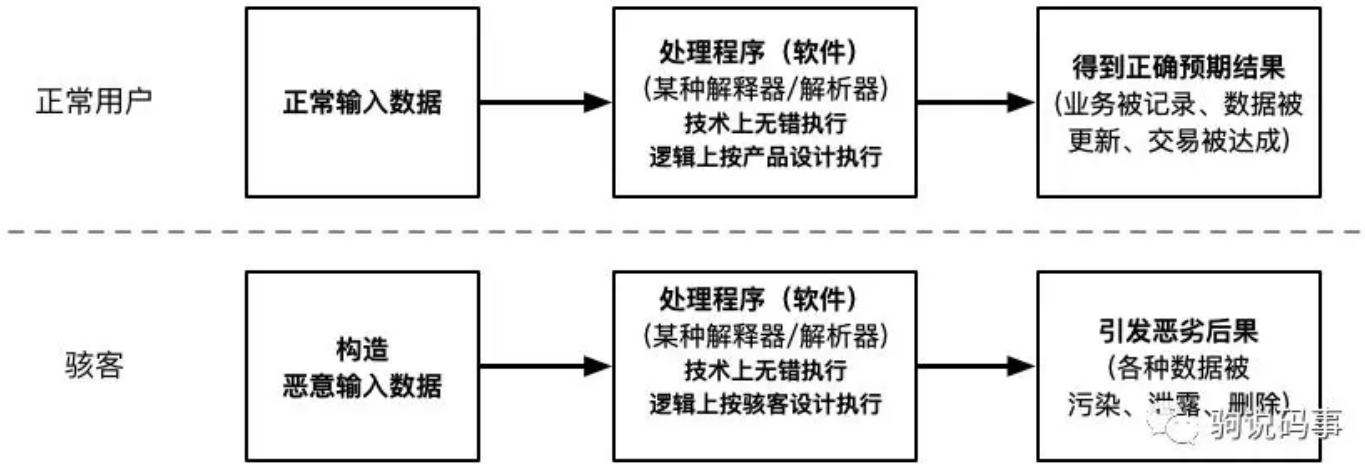


Hello, <Config {'ENV': 'production', 'DEBUG': True,
'PERMANENT_SESSION_LIFETIME': datetime.time
'SESSION_COOKIE_DOMAIN': None, 'SESSION_CC
'SESSION_REFRESH_EACH_REQUEST': True, 'MA
None, 'TRAP_HTTP_EXCEPTIONS': False, 'EXPLAI
'JSONIFY_PRETTYPRINT_REGULAR': False, 'JSON

应对：

不要使用%s，而应在模板中直接使用{{var}}。

小心 注入攻击




```
query = “SELECT * FROM accounts WHERE uid = ‘” +
request.args.get(“uid”) + ”’”
resuts = mysql.execute(query)
```

```
http://example.com/accounts?uid=' or '1'='1
```

```
SELECT * FROM accounts WHERE uid = '' or '1'='1'
```

应对：

对外部输入进行完善的检查、过滤、转义操作后，再进行执行。

参考资料：“驹说码事”《注入的原理与防御措施》

https://mp.weixin.qq.com/s/Zd81qlcWJi4H1AD9WzPX7w

小心 PyPI 依赖包

不要以为 Star 多的包就不存在漏洞；
更不要以为 PyPI 源中的包就不存在恶意代码；

不要以为你import的就是你实际要import的*；

每个 PyPI 包都可能存在前述所有风险点，
有的甚至是故意、恶意为之。

应对：
谨慎选择第三方 PyPI 包，尽量少导入 PyPI 包；
利用 https://pyup.io/ 等服务保持检查和更新依赖；
利用 Chef InSpect 落实代码安全规范的检查。
*参考资料：“驹说码事” 《如何import一个不存在的对象》
https://mp.weixin.qq.com/s/0_ivKVDU-nKf3r-c96sqrA

利用 Bandit 检查代码

记不住那么多点？交给Bandit！

B101	assert_used
B102	exec_used
B103	set_bad_file_permissions
B104	hardcoded_bind_all_interfaces
B105	hardcoded_password_string
B106	hardcoded_password_funcarg
B107	hardcoded_password_default
B108	hardcoded_tmp_directory
B110	try_except_pass
B112	try_except_continue
B201	flask_debug_true
B301	pickle
……	

```
└─○ bandit -r pycon2019
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.7.2
Run started:2019-10-18 18:12:52.154742

Test results:
>> Issue: [B307:blacklist] Use of possibly insecure function - consider using sa
   Severity: Medium   Confidence: High
   Location: pycon2019/test_eval.py:7
   More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls
6           try:
7               v = eval(s)
8           except Exception as e:

--------------------------------------------------
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: '123456'
   Severity: Low   Confidence: Medium
   Location: pycon2019/test_input.py:3
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b105_hardcoded_pas
2
3       secret = "123456"
4
5       def sign_up():

--------------------------------------------------
>> Issue: [B322:blacklist] The input method in Python 2 will read from standard
ode. This is similar, though in many ways worse, then using eval. On Python 2, u
   Severity: High   Confidence: High
   Location: pycon2019/test_input.py:6
   More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls
5       def sign_up():
6           name = input("please enter your name:")
7           print("Your name is: ", name)
```

https://github.com/PyCQA/bandit

https://bandit.readthedocs.io/en/latest/

总结：如何规避代码风险

如何尽量规避风险

1. 默认不相信外部输入，需要进行检查；
2. 内部逻辑也应该做检查，例如购买东西以后余额应该降低而不是升高；
3. 奥卡姆剃刀原则，用能达到目的的最简单的设计、配置，减少缺陷的可能性；
4. 最小权限原则，仅用恰好够用的权限去执行代码，减少越权漏洞；
5. 定时清理缓存数据，以及非业务代码的固有数据，减少入侵风险；
6. 在发布代码前，使用 Bandit 工具检查代码，规避最常见的不安全写法；
7. 敏感信息一定加密后使用；
8. 不要将敏感数据驻留在内存中，包括locals()、globals()可以访问到的；
9. 使用虚拟环境隔离每个项目，对不可信数据和代码在沙盒中执行；
10. 保持操作系统、Python解释器、PyPI包的更新，尤其应关注安全更新；
11. 定期关注Python相关的CVE报告 https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=python；
12. 定期关注 OWASP https://www.owasp.org/ 的安全报告，并依此排查所负责代码中是否存在相关缺陷；
13. 多重防御，安全的编码 + 安全的系统配置 + 充分的测试 + 安全的操作规范。

THANK YOU

事故出于麻痹
安全来于警惕

代码千万行，安全第一行；
编码不安全，老板两行泪！