PyCon China 2020

PyConChina 2020
PYTHON 中国开发者大会 2020

Python For Good

Python “业务逻辑安全”&& “业务代码安全”

李嘉旭 | 360政企网络安全专家

个人介绍

- 一名安服仔
- 热爱挖漏洞与撸猫
- 360政企网络安全专家
- 热爱安全开发 工具研发
- 大二学生一枚

1.业务逻辑安全

	业务漏洞	常规漏洞
优点	不易被检测设备发现，很难触发安全设备，往往较易挖掘到	大部分危害较大 可以获取到大量的敏感信息或者服务器权限
缺点	部分业务漏洞危害较小,大部分无法获取到服务器权限	很容易触发到安全设备 防御简单，在WEB框架中 除非故意写 否则很难做出来

业务逻辑安全

产生和危害

程序员想错了思路

对于当前业务的不了解

如果业务出错那就会出现一个好玩的情况 如果A公司的A网站出现问题 那么A公司的B网站出问题的几率有百分之八十 因为程序员没变 开发该业务的逻辑没变

业务逻辑安全

可能为这样讲 你没太听明白 让我们举个例子

业务逻辑安全

前后端分离是目前WEB开发的一大热点，举个一家较小的外包公司的例子

首先看看他们开发网站其中一处业务的逻辑

忘记密码 —— 用户点击忘记密码 —— 进入忘记密码页面 —— 用户输入手机号 —— ajax将手机号发送到接口处 —— 发送验证码给手机号 —— 用户将得到的验证码输入 —— 后端进行判断 如果正确就返回 {'status':0} 错误返回{'status':1} —— 前段验证status 如果为0就跳转到密码修改界面 为1就返回验证码输入错误

合格，没毛病

业务逻辑安全

这时候黑客来了 看看这个逻辑如何被黑客利用

后端进行判断 如果正确就返回 {'status':0} 错误返回{'status':1}

- 前段验证status 如果为0就跳转到密码修改界面 为1就返回验证码输入错误
- 黑客直接抓取返回包 将返回包发成正确返回 欺骗前端

1.1 重灾区 “注册”and“登陆”

业务逻辑安全

小明作为一个BUG开发工程师 接到公司项目要求写一个注册和登陆

代码地址：

https://github.com/HTFTIMEONE/py_Vulnerability_drone/blob/main/pycon2020/%E7%94%A8%E6%88%B7%E5%90%8D%E7%88%86%E7%A0%B4.py

先让我们看看他第一个出现问题的地方

其中有段代码如下

```
sql1 = "select * from newuser where username=:username"
sql2 = "insert into newuser(username,password,email,phone,name) values(:username,:password,:email,:phone,:name)"
result = db.session.execute(sql1,{"username":username})
if len(result.fetchall()) > 0:
    return "该用户已经被注册"
else:
    result = db.session.execute(sql2,{"username":username,"password":password,"email":email,"phone":phone,"name":name}
    if result:
        return "注册成功"
    else:
        return "注册失败"
```

合格，没毛病

业务逻辑安全

但是机智的黑客又发现了问题

黑客将这个注册的数据包抓住 掏出了自己的字典 开始不停的重发 修改username字段 进行爆破 就这样 黑客成功爆破出来小明网站注册的1000个用户

那么可能你在想 这有什么危害呢
假如你现在的登陆界面有验证码 黑客进行爆破会难度变大 但是黑客找到了在线的验证码接收平台 如果按照正常爆破 他需要爆破10000个用户 这个接口当然不是免费的 这时候黑客刚刚得到的1000个用户就得到用处的 这样就可以精准爆破
1.这时候黑客又来到 登陆界面 抓住登陆的数据包 将username 不停的爆破 爆破的字典就是刚那1000个用户 密码设置成123456 最后黑客成功登陆了近100个用户的账号
2.这已经算信息泄露了

业务逻辑安全

小明立马修复了漏洞 并且迅速重新上线

上线之后 第一天就有近百万的注册量 领导高兴坏了

第二天有100亿的注册量 领导感到一丝不对劲

第三天有1000亿的注册量 领导跑路了

小明重新看了代码 终于发现了问题 发现自己的验证码逻辑产生了问题

代码地

址:https://github.com/HTFTIMEONE/py_Vulnerability_drone/blob/main/pycon2020/%E9%AA%8C%E8%AF%81%E7%A0%81%E9%80%BB%E8%BE%91%E9%94%99%E8%AF%AF.py

```
    new_captcha = CaptchaTool()
    img, code = new_captcha.get_verify_code()
    session["code"] = code
    return render_template('register.html',img_stream=img.decode('ascii'))
```

看代码我们可以得知

这段代码的逻辑是将code储存到session中 然后再进行对比的地方再将用户输入的内容和session中的code进行对比 我们会发现这个session会一直储存 除非我们刷新整个页面 否则这个code就不会被改变 我们就可以一直不停的请求

业务逻辑安全


```
POST /reister HTTP/1.1
Host: 192.168.3.71:5000
Content-Length: 54
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.3.71:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/88.0.4302.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apn
g,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://192.168.3.71:5000/reister
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: session=eyJjb2RlIjoiNDgxNSJ9.X7kKBg.Us-attHX-JDYSSVUOFwCKJM3thE
Connection: close

username=4&password=4&email=4&phone=4&name=4&code=4815
```

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 12
Vary: Cookie
Server: Werkzeug/1.0.1 Python/3.8.2
Date: Sat, 21 Nov 2020 12:39:37 GMT

注册成功
```

```
POST /reister HTTP/1.1
Host: 192.168.3.71:5000
Content-Length: 54
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.3.71:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/88.0.4302.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apn
g,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://192.168.3.71:5000/reister
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: session=eyJjb2RlIjoiNDgxNSJ9.X7kKBg.Us-attHX-JDYSSVUOFwCKJM3thE
Connection: close

username=5&password=4&email=5&phone=5&name=5&code=4815
```

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 12
Vary: Cookie
Server: Werkzeug/1.0.1 Python/3.8.2
Date: Sat, 21 Nov 2020 12:41:33 GMT

注册成功
```

业务逻辑安全

小明很快就修复了这个漏洞 在注册栏添加了手机号验证

结果放上去第一天小明就发现注册了5个用户 但是却花掉了1万的短信费用 小明觉得此时有蹊跷

小明查看了服务器的请求日志 发现有个IP在不停的请求 短信接口

项目地址： https://github.com/HTFTIMEONE/py_Vulnerability_drone/blob/main/pycon2020/%E7%9F%AD%E4%BF%A1%E8%BD%B0%E7%82%B8.py

看代码小明立马发现了问题所在 原来是没有限制用户注册的时候 使用接口的次数 小明也很快做了限制 要求单个手机号一分钟内只能发送一次

业务逻辑安全

修复之后 发现确实没有了 但是没过几天 又开始了 小明去查看了web日志才知道到底是怎么回事

首先看段代码

a2.py > ...

```
1  session = "131**799518"
2  sessions = "131**799518 "
3  if session==sessions:
4      print("这是相等的")
5  else:
6      print("不想等")
```

执行结果是不相等 所以如果不在输入手机号的地方做过滤 那么就可以绕过我们前面所写的代码

黑客输入手机号+一个空格 就绕过了我们的限制 而这个手机号+空格 到了api接口处就被api自动处理了

下次请求 手机号+俩个空格 再下次 手机号+三个空格 以此类推

```
{"status":"error","code":411,"msg":"empty sms signature"}
  ➜  ~ curl -d 'appid=37697&to=131[redacted]799518&content=[redacted]您的短信验证码：4438，请在10分钟内输入。&signature=9a0[redacted]edaf25d176459d899f2' https://api.mysubmail.com/message/send.json
{"status":"success","send_id":"0a09ffd46fa4343004486c1010b1df8","fee":1,"sms_credits":"28","transactional_sms_credits":"0"}
➜  ~ curl -d 'appid=37697&to=131[redacted]799518 &content=[redacted]您的短信验证码：4438，请在10分钟内输入。&signature=9a[redacted]edaf25d176459d899f2' https://api.mysubmail.com/message/send.json
```

注册问题通用的解决方案

- 加上CSRF_TOKEN 对每个请求进行验证
- 限制用户在十秒内对请求次数 如果次数过多可以判断此用户非正常用户
- 最好是对每个请求过来对参数在前端中进行加密处理

业务逻辑安全

接下来让我们看看登陆上面的问题

代码如下： https://github.com/HTFTIMEONE/py_Vulnerability_drone/

在登陆处 遇见最多的无疑是爆破穷举问题

我们可以添加验证码来有效防御 但是如果验证码也像我们前面所写那样失去效果 或者验证码过于单调被黑客轻易识别 那么问题依然种种 那么接下来推荐多个解决方案

- 对每个请求设置token
- 限制用户在10秒内的请求次数
- 添加难度大的验证码
- 对password处在js中进行加密（有可能会被逆向出来

这个办法不是特别靠谱）

那你巨几把棒哦

业务逻辑安全

忘记密码

- 密码重置链接过弱 导致黑客猜解出重置链接生成方式 随意重置密码
- 手机验证码位数少 并且无验证次数限制
- 逻辑产生 问题 可以绕过密码重置前续步骤
- 密码重置的时候 接受验证的账户可被用户篡改

业务逻辑安全

登陆的注意事项:

很多开发测试人员 都会开通测试账号 来测试自身系统的稳定性 那么就会导致出现一些问题

大部分的测试账号 账号密码 都可能为

test:test

demo:demo

t1:t1

123:123

等等 黑客往往都非常喜欢这些账户

1.2 "专业"的信息泄露接口

业务逻辑安全

信息泄露 是我们能常听见的 比如某某著名网站存在泄露 泄露大量用户

例如今年的微博泄露 华住泄露 数据量极大 危害极高 也导致企业和用户损失重大

我将它分为 直接信息泄露 和间接信息泄露

直接信息泄露：设计缺陷和管理不当 导致的直接信息泄露

间接信息泄露：通过其他漏洞进入后台或者得到某个特定功能的权限而得到的信息

在这里我们只讲直接信息泄露 我将它大致的分为三个：

1.越权

2.未授权

3.GitHub仓库管理不当

业务逻辑安全

有一天 项目又来了 小明这次负责个人信息管理的地方

小明 一想 这个好啊 拿手绝活 半个小时不到 小明就写完了

代码地址：https://github.com/HTFTIMEONE/py_Vulnerability_drone/

```
def infos():
    id = request.args.get('id')
    sql1 = "select * from newuser where id=:id"
    result = db.session.execute(sql1,{"id":id})
    userinfo = result.fetchall()
    print(userinfo)
    data = {'username':userinfo[0][1],'email':userinfo[0][3],'phone':userinfo[0][5],'name':userinfo[0][5]}
    return render_template('info.html',**data)
```

那么这种就属于平行越权 因为是A用户获取了B用户的个人数据

还有一种越权叫垂直越权 是用户身份使用了管理员权限的功能

越权这种漏洞 出现都是对于权限的限制不清楚 或者根本就没有限制

业务逻辑安全

小明很快便修复了这个漏洞 这是领导又下任务了 让写一个管理平台 专门管理所有用户的信息的管理平台

小明很快也写好了

项目地址：https://github.com/HTFTIMEONE/py_Vulnerability_drone/

结果又被别人脱数据了 这次并没有很大的请求量

从代码中可以看出来 在用户信息管理的地方 没有进行限制 导致 可以直接未授权去访问

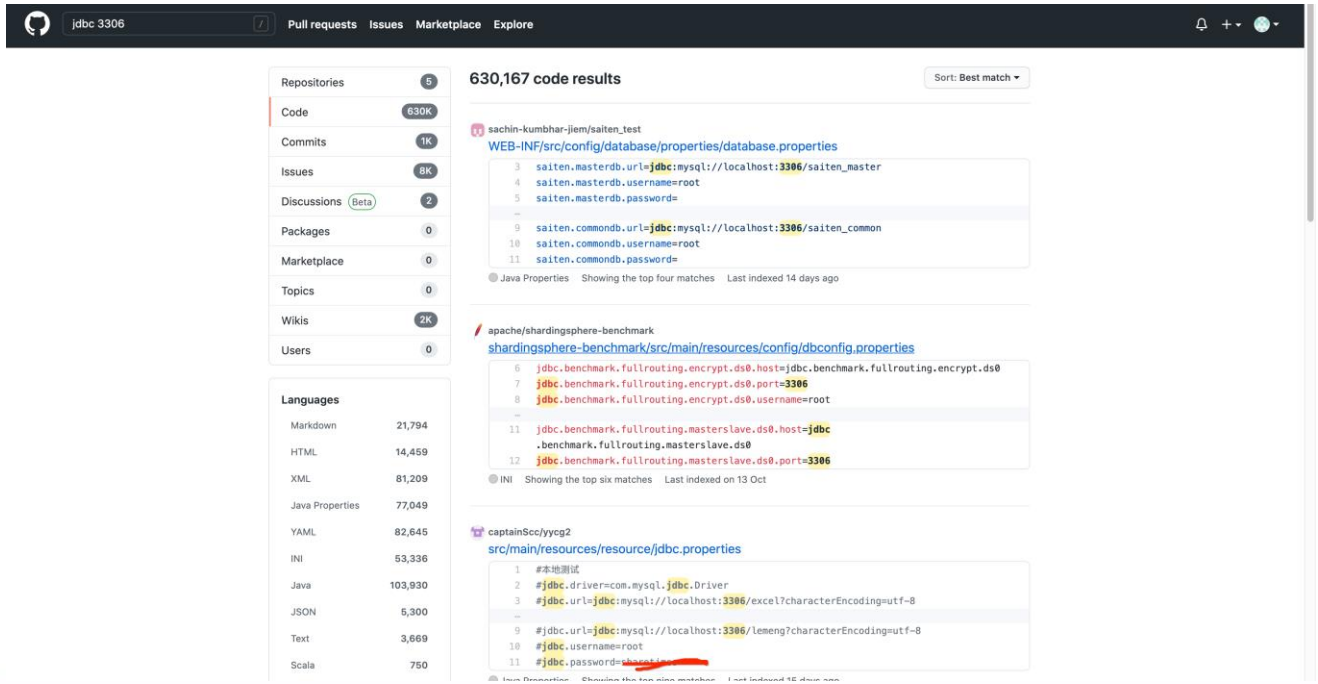
其实这种未授权 大部分是出现在文件上传接口 文件下载接口 这些地方

业务逻辑安全

小明想着 这次终于没黑客搞了 不说了 上传GitHub 保存起来

结果上传之后 信息又泄露了 这次更严重 黑客直接链接上了 数据库 小明知道这事不对 突然发现 原来是将 配置文件也发到了GitHub上面

小明通过搜索关键词 不搜不知道 一搜吓一跳 没想到那么多和他一样的人 在GitHub上面发了自己的配置文件



1.3 "羊毛党"的青春故事

业务逻辑安全

薅羊毛 分为俩种薅羊毛

1.好的薅羊毛 就是领领你发的优惠券 走正常流程

2.坏的薅羊毛 坏的薅羊毛 就是硬薅 不管你身上的毛够不够 他们往往喜欢破坏你正常的流程

或者利用自身代码设计缺陷

我将这种硬薅分成了四个问题 来进行讲解

1.优惠券问题

2.订单问题

3.支付问题

4.撞库问题

业务逻辑安全

优惠券ID较弱

项目地址：https://github.com/HTFTIMEONE/py_Vulnerability_drone/

他的问题其实很简单 就是生成的优惠卷 我们可以看到是 1234 这样的ID 很容易就被用户猜解出来 或者是有规律的生成

例如 123abc

或者将123进行md5加密 1234进行md5加密

除了md5加密 更多是用base64加密

以此类推的方式就可以算出来规律

业务逻辑安全

优惠卷重复使用 就是当A优惠卷使用完毕之后 发现他依然可用导致黑客抓包之后 可以一直使用该商品购物卷

项目地址：https://github.com/HTFTIMEONE/py_Vulnerability_drone/

```
POST /shop HTTP/1.1
Host: 172.20.10.10:5001
Content-Length: 14
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://172.20.10.10:5001
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/88.0.4302.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apn
g,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://172.20.10.10:5001/shop
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie:
session=eyJjb2RlIjoiNzIwMyIsInVpZCI6MiwidXNlcm5hbWUiOiJhZG1pbiJ9.X7pqTA.EMHOShljdBMY
kzYRVAioOhv2UPk
Connection: close

id=3&pay=50&yhj=123
```

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 12
Vary: Cookie
Server: Werkzeug/1.0.1 Python/3.8.2
Date: Sun, 22 Nov 2020 13:44:41 GMT

购买成功
```

业务逻辑安全

支付问题 已经是个老生常谈的问题 比如最简单的 0 元购物

项目地址：https://github.com/HTFTIMEONE/py_Vulnerability_drone/

我们通过抓包 发现可以修改支付的金额

```
POST /shop HTTP/1.1
Host: 172.20.10.10:5001
Content-Length: 14
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://172.20.10.10:5001
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/88.0.4302.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apn
g,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://172.20.10.10:5001/shop
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie:
session=eyJjb2RlIjoiNzIwMyIsInVpZCI6MiwidXNlcm5hbWUiOiJhZG1pbiJ9.X7pqTA.EMHOShljdBMY
kzYRVAioOhv2UPk
Connection: close

id=3&pay=0.01&yhj=123
```

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 12
Vary: Cookie
Server: Werkzeug/1.0.1 Python/3.8.2
Date: Sun, 22 Nov 2020 13:44:41 GMT

购买成功
```

我们可以通过修改金额为0.01 直接进行购买

业务逻辑安全

当然除了 修改成0.01 我们可以修改成负数 先让我们看一段代码

```
1   b = -100
2   a = 1 - b
3   print(a)
```

每天一个坐牢小技巧

得出的结果为101 负负得正 以此如果我们使用负数去支付 可以不减反增我们的余额

业务逻辑安全

来源：乌云

1 和银行交易时，做数据签名，对用户金额和订单签名。

2 敏感参数不要明文放在URL中

3 服务端效验客户端提交的参数

4 在服务端计算金额的时候，一定要判断是否为正数。

5 支付过程中加一个服务器生成的key，用户校验参数有没有被串改。

6 如果一定需要用URL传递相关参数，建议进行后端的签名验证

7 订单金额和充值接口返回的数据进行校验

8 提交订单时后台判断单价是否与数据库中相符，如不符则返回错误。

9 支付时应从服务器拉取数据，而不是直接读客户端的值！！

业务逻辑安全

咱们这里讲一个羊毛党最爱的办法 在几年前盛行的1元购 羊毛党最喜欢的就是撞库

咱们了解一下什么是撞库 首先这个撞库 不是跑到你的机房 撞你家的数据库

举个例子 首先大家在很多平台都有自己的账号 这些账号我相信大部分的密码是一致的

假设A网站 被黑客干掉了他成功找到了一堆账户 然后他去B网站 用从A得到的用户账户密码 去登陆 B网站

这就是撞库 他利用的就是人们一般设置密码 都是一致的 来进行撞库

那么这个就需要动用我们小脑瓜子了

1.添加验证码 并且把验证码机制拉满 很难进行机识（推荐）

2.对密码在传输过程中进行加密 比如在前段js 进行加密

3.添加csrf_token

2 代码安全

代码安全

在python中常规漏洞 出现次数确实很少 但是有时候错误的写法也会不可避免的出现 或者 那么如何规范写法 彻底防御常规漏洞 是我们较为需要的

2.1 WEB框架问题

代码安全

我们从俩个框架去聊 Django 和Flask 我们知道他俩均为MVC框架

那么他们就肯定涉及到俩点 数据库查询 视图渲染 而这俩点往往是黑客最喜欢攻击到

先看数据库查询 数据库查询最多到无疑就是SQL注入 但是SQL注入只需要你写法无误 并且使用到为最新版到框架 均为特别大大问题

在看视图渲染 视图渲染最多的问题就俩种 SSTI XSS

修复的办法是使用官方推荐的视图模版 不要让用户可以控制html标签属性

那么最好的解决办法就是使用官方最新的版本 去做修改

2.2 代码写法问题

代码安全

让我们从SQL注入的例子来看

首先数据库内只含有俩个用户 admin和demo

后端代码如下:

我们传值username=ad'or+1=1#

```
sql = "select * from users where username=:username"
result = db.session.execute(sql,{"username":username})
```

```
None
```

```
sql=text("SELECT * from users WHERE username = '%s';" %username)
result = db.session.execute(sql)
```

最后: SELECT * from users WHERE username = 'ad'or 1=1#';"

```
(1, 'admin', 'admin123')
```

代码安全

CSRF防御

Django中CSRF防御 只需要在前端的表单立马添加{% csrf_token %}

```
<p>欢迎用户{{ username }}</p>
<form method="POST">
 {% csrf_token %}
 <p>商品: {{ shopinfo }}</p>
 <p>单价: {{ payinfo }}</p>
 <input name="pay" value="{{payinfo}}" type="hidden">
 优惠券: <input name="yhj">
 <input type="submit" value="购买">
```

搞定了，就是这么简单

代码安全

CSRF防御

Flask中CSRF防御

在自己的py立马添加

CSRFProtect(app)

```
app = Flask(__name__)
CSRFProtect(app)
```

在前段表单添加

<input type="hidden" name="csrf_token" value="{{ csrf_token() }}"/>

```
优惠券: <input name="yhj">
<input type="submit" value="购买">
<input type="hidden" name="csrf_token" value="{{ csrf_token() }}"/>
```

搞定了，就是这么简单

代码安全

csrf 防御方法也很简单 只需要你去按照官方文档去写就行了

所以一般我们写法只要不是 过于追求简易 在数据库操作中 使用orm 是一个不错的想法

xss防御其实很简单 只需要你去使用官方推荐的官方视图即可 同时 ssti防御也是如此

其实对于这些常见的web框架 防御方法很简单 就是使用官方文档中推荐的内容去

PyCon China 2020

Python For Good

PyConChina 2020 | PYTHON 中国开发者大会 2020

THANK YOU

HTFTIME

扫一扫上面的二维码图案，加我微信