Python注入判断原理及实践

安全研究部　廖新喜

关键词：Python 注入 源码 语法树

摘要：Python 由于其简单、快速、库丰富的特点在国内使用的越来越广泛，但是一些不好的用法却带来了严重的安全问题。本文从 Python 源码入手，分析其语法树，跟踪数据流来判断是否存在注入点。

引言

Python 注入问题是说用户可以控制输入，导致系统执行一些危险的操作。它是 Python 中比较常见的安全问题，特别是把 Python 作为 Web 应用层时这个问题就更加突出，它包括代码注入、OS 命令注入、SQL 注入、任意文件下载等。

一、注入的场景

主要是在 Web 应用场景中，用户可直接控制输入参数，并且程序未做任何参数判断或者处理，直接就进入了危险函数中，导致执行一些危险的操作。主要的注入类型有：

（一）OS 命令注入

主要是程序中通过 Python 的 OS 接口执行系统命令，常见的危险函数有 os.system,os.popen,commands.getoutput,commands.getstatusoutput,subprocess 等一些接口。例如：def myserve(request,fullname):os.system('sudo rm -f %s'%fullname), fullname 是用户可控的，恶意用户只需利用 shell 的拼接符就可以完成一次很好的攻击。

（二）代码注入

是说在注入点可以执行一段代码，这个一般是由 Python 的序列话函数 eval 导致的，例如：def eval_test(request,login):login = eval(login), 如果恶意用户从外界传入 __import__('os').system('rm /tmp -fr') 就可以清空 tmp 目录。

（三）SQL 注入

在一般的 Python Web 框架中都对 SQL 注入做了防护，但是

千万别认为就没有注入风险，使用不当也会导致 SQL 注入。例如：

```
def getUsers(user_id):
sql = ‘select * from auth_user where id =%s’ %user_id
res = cur.execute(sql)
```

（四）任意文件下载

程序员编写了一个下载报表或者任务的功能，如果没有控制好参数就会导致任意文件下载，例如：def export_task(request,filename):return HttpResponse(fullname)。

二、判断原理

从以上四种情况来看，都有一个共同点，那就是危险函数中使用了可控参数。如 system 函数中使用到的('sudo rm -f %s'%fullname)，如 eval 中使用到的 login 参数，如 execute 函数中使用到的 user_id 参数，如 HttpResponse 中使用到的 fullname 参数，这些参数直接从函数中传进来，或者经过简单的编码、截断等处理直接进入危险函数，导致了以上危险行为。如果在执行危险函数前对这些可控参数进行一定判断，如必须是数字，路径必须存在，去掉某些特殊符号等则避免了注入问题。

有了这个基础理论，这些参数数据在传递的过程中到底有没有改变？怎么顺利地跟踪可控参数呢？接下来分析 Python 的语法树。

三、Python 语法树

很显然，在参数不停传递过程中，普通的正则表达式已经无能为力了。这个时候就可以体现 Python 库丰富的特点。Python 官方库中就提供了强大的 Python 语法分析模块 ast。我们可以利用根据 ast 优化后的 PySonar 模块，PySonar 相对于 ast 模块而言有性能上的提升，另外是以 Python 的 dict 来表示的。

（一）语法树的表示 - 文件

一个文件中可以有函数，类，它是模块的组成单位。大体结构如下：{ “body” :[{},{}],” filename” :” test.py” ,” type” :” module” }，这是文件 test.py 得到的语法树结构，body 里面包含两个 dict，实际里面会存放函数、类、全局变量或者导入等，它是递归嵌套的，type 字段表明类型，在这里是模块，filename 则是它的文件名。

（二）语法树的表示 - 函数

函数的作用就不用多说了，django 的 view 层基本都是以函数为单位的。下面来看一个函数的语法树，如图 1：

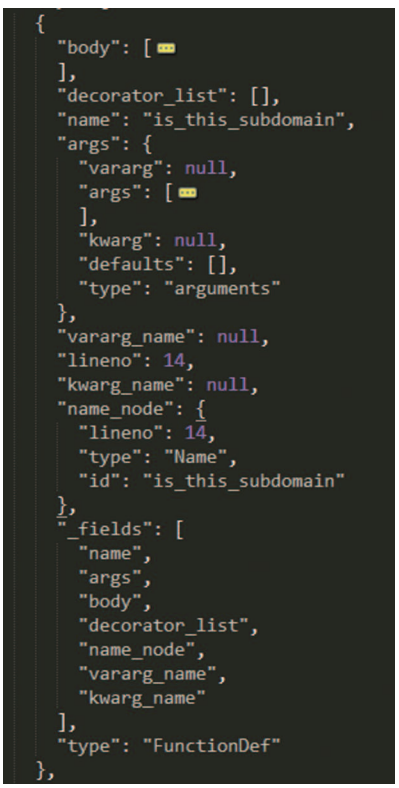

图 1 函数的语法树

我们简单分析一下这个结构，首先是 type，这里是 FunctionDef，说明这个结构体是一个函数。_fields 中的 name、args、body、decorator_list 等是函数的基本组成单位。name 是函数名称，上述函数名为 is_this_subdomain；args 是函数的参数，它包含普通参数 args，默认参数 kwarg；lineno 是标明该语句所在的文件的行数；decorator_list 则是函数的修饰器，上述为空。

（三）语法树的表示 - 类

在类的语法树中，包含 body、decorator_list、lineno、name、base 等字段 type 是 ClassDef，表明该结构为 class，body 中则包含着函数的结构体，base 则是继承的父类。

（四）语法树的表示 - 示例

接下来我们将以一个 if 结构片段代码作为示例，来解释 Python 源码到其语法树的对应关系。片段代码: if type not in ['RSAS", "BVS"]:return HttpResponse("2")，得到的语法树如图 2。

在这个语法树结构中，body 里包含着 if

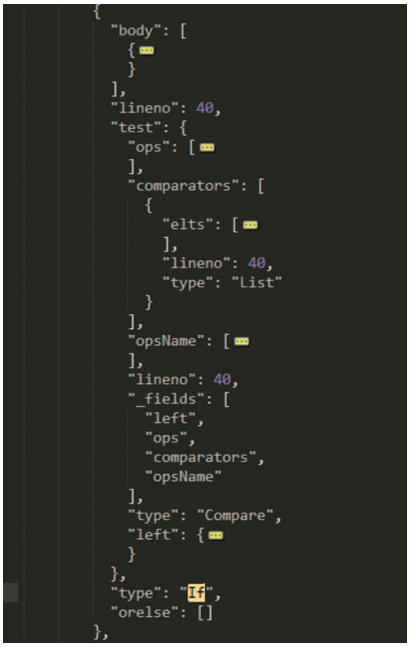

图 2 if 结构示意图

结构中的语句 return HttpResponse("2")。type 为 Compare 表示该结构体为判断语句，left 表示左值即源码中的 type。test 结构体则是用来进行 if 判断，test 中的 ops 对应着源码中的 not in，表示比较判断，comparators 则是被比较的元素。这样源码就和 Python 语法树一一对应起来，有了这些一一对应的基础，就有了判断 Python 注入问题的原型。

四、注入判断的实现

注入判断的核心就在于找到危险函数，并且判断其参数是可控的，找到危险函数这个只需要维护一个危险函数列表即可，当在语法树中发现了函数调用并且其名称在危险列表中就可以标记出该行代码。接下来的难点就在于跟踪该函数的参数，默认认为该危险函数的外层函数的参数是可控的，那就只需要分析这个外层函数参数的传递过程即可。首先分析哪些情况下，从一个参数赋值给另外一个参数其值还是可控的，下面列举了 5 种基本情况：

（1）属性取值：对一个变量取属性，比如 request 的 GET、POST、FILES 属性，属性的属性还是可控的，但是 request 的其他字段如 META、user、session、url 则得排查开外。

（2）字符串拼接：被拼接的字符串中包含可控参数，则认为赋值后的值也是可控的，需要考虑好各种拼接情况，如使用 +、% 等进行拼接。

（3）分片符取值：一般认为分片后的值

也是可控的。

（4）列表解析式，如果列表解析式基于某个可控因子进行迭代，则认为赋值后的列表也是可控的。

（5）简单的函数处理: a，处理函数是字符串操作函数（str, unicode, strip, encode 等）; b，简单的未过滤函数，也就是说这个函数的返回参数是可控的。

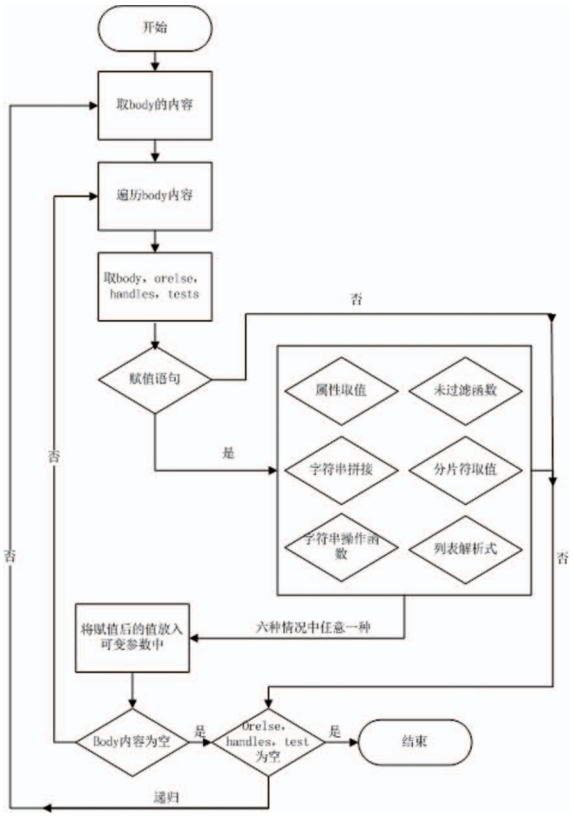

图 3 参数跟踪程序流程图

对外层函数中的所有代码行进行分析，判断是否是赋值类型，如果赋值类型的操作属于以上五种情况中任意一种，则将该赋值后的值放入可变参数列表中，具体的流程如图 3。

另外在分析的过程中还得排除下列情况，提前结束分析。第一种情况是 if 语句中有 os.path.exitst,isdigit 带可控参数并且含有 return 语句，如（if not os.path.isdir(parentPath)：return None）；第二种情况是将可控参数锁定在某个定值范围并直接返回的，如（if type not in ["R", "B"]：return HttpResponse("2")）。

五、结束语

对 Python 源码实现注入问题的自动审查，大大降低了人为的不可控性，使代码暴露出来的漏洞更少。当然目前来说，这个模块还是有一定局限性，对类的处理不够充分，没有分析导入的函数对属性的取值也不够细分等问题。

参考文献

Python 语法树 https://greentreesnakes.readthedocs.org/en/latest/nodes.html