【译】实战演练：看我是如何将LFI变为RFI的

bluesky / 2017-09-04 03:32:00 / 浏览数 5442

实战演练：看我是如何将LFI变为RFI的
前言
PHP文件包含漏洞的产生原因是在通过PHP的函数引入文件时，由于传入的文件名没有经过合理的校验，从而操作了预想之外的文件，就可能导致意外的文件泄露甚至恶意的
File Inclusion）漏洞了。
常见漏洞代码

```
if ($_GET['method']) {

  include $_GET['method'];

} else {

  include 'index.php';

}
```

一般情况下，程序的执行过程是当用户提交url为https://xianzhi.aliyun.com/sth.php?method=search.php时，
调用search.php里面的样式内容和功能。直接访问https://xianzhi.aliyun.com/search.php
则会包含默认的index.php里面的样式内容和功能。那么问题来了，如果我们提交https://xianzhi.aliyun.com/search.php?method=upload/1.jpg
，且1.jpg是由黑客上传到服务器上的一个图片，并在图片的末尾添加了恶意的php代码，那么恶意的代码就会被当前文件执行，以此触发本地文件包含漏洞。

有趣的发现
我和我的好朋友Mike
Brooks一直致力于对一些开源的Web框架进行代码审计工作，在对这些Web开源框架代码审计的过程中，我们找到了一种将本地文件包含漏洞（LFI）转换为远程文件包含

```
public class LoadRunner {
   static {
       System.out.println("Load runner'ed");
   }

   public static void main(String[] args) {

   }
}
```

我们首先编译这个java类，然后在代码中加载它，具体实现如下图所示：

现在，我们已经有了两个有趣的发现：一个是可以在加载的JAR文件中插入执行代码，另一个是在Web服务器上找到一个合适的文件路径来加载JAR包文件。因此，我们现在
Brooks想出了一个好主意。

文件描述符
一般情况下，大多数Web开源框架都会将上传的文件落地到服务器的某个磁盘上，但文件的路径是不可猜测的（通常使用GUID或其他随机标识符来表示），如果我们不知道

加载文件描述符
为了实现上面的方法，我们首先需要在程序中找到那些处理文件上传的请求，之后尝试通过Web开源框架中的LFI漏洞来查看所有PID文件描述符，并通过文件描述符来访问
Web框架直接在HTTP GET请求填充了FILES字典字段，以下是超简单的Flask应用程序：

```
# -*- coding: utf-8 -*-
import os

from flask import Flask, request


UPLOAD_FOLDER = "/tmp"

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

@app.route("/", methods=["GET"])
def show_me_the_money():
   x = request
   import code
   code.interact(local=locals())
```

```
if __name__ == "__main__":
   app.run()
```

在这个应用程序中，我们有一个单一的处理程序，该处理程序允许在URL上挂载HTTP GET请求。然后我们在Ubuntu VM中运行这个程序，并通过HTTP GET请求将文件上传到该服务器上。对于以前没有使用过import code技巧的人来说，这是一个很好的方法来调试Python代码和库，因为在code.interact被调用时你将进入到python的REPL环境中去。
以下是通过HTTP GET请求上传文件的简单脚本：

```
# -*- coding: utf-8 -*-
import requests

response = requests.get(
   "http://127.0.0.1:5000/",
   files={
       "upload_file": open("/tmp/hullo", "rb"),
   },
)
```

而在/tmp/hullo的文件中，我们可以看到很多的“Hello World”：

然后，我们首先运行服务器，之后上传文件，并进入Flask请求处理程序上下文中的python REPL环境，具体如下图所示：

通过使用请求处理程序的PID，我们可以查看到磁盘上打开的文件描述符：

然后我们返回到REPL并访问上传的文件：

现在该文件已经在Web服务器中被访问过了，此刻我们回到/proc目录，看看是否可以找到上传文件的内容：

果然，我们找到了上传的文件！因此，对于我们正在评估的Web应用程序来说，我们确认通过这种上传和引用文件的方法可以正常访问到我们驻留在Web服务器中的JAR包文
我们可以通过多次上传相同的文件来进一步减少文件路径搜索空间，因此我修改了文件上传的代码使得可以上传相同的九个文件：

```
# -*- coding: utf-8 -*-
import requests

response = requests.get(
   "http://127.0.0.1:5000/",
   files={
       "upload_file": open("/tmp/hullo", "rb"),
       "upload_file2": open("/tmp/hullo", "rb"),
       "upload_file3": open("/tmp/hullo", "rb"),
       "upload_file4": open("/tmp/hullo", "rb"),
       "upload_file5": open("/tmp/hullo", "rb"),
       "upload_file6": open("/tmp/hullo", "rb"),
       "upload_file7": open("/tmp/hullo", "rb"),
       "upload_file8": open("/tmp/hullo", "rb"),
   },
)
```

运行此脚本后，访问处理程序中的FILES字典，并查看请求处理程序PID目录中的fd目录内容，我们看到所有上传的文件都有打开的文件描述符：

因此，通过使用这种方法，我们可以保证具有特定号码的文件描述符终将会指向我们上传的文件。如果我们提交100个上传文件的请求，可能文件描述符50指向的就是我们的
如何实施攻击利用？
总而言之，为了引用上传文件以达到攻击的目的，这是一种大大减少搜索空间的方法，这在许多情况下可以使LFI成为RFI。如果你要使用此方法进行攻击利用，请考虑以下事
通过我们对多个Web框架（Django和Flask）的分析发现，当访问FILE字典时框架会延迟加载文件引用。因此，我们必须定位访问FILES字典的请求处理程序
。一旦请求处理程序访问了FILES字典，文件描述符将在请求处理期间一直保持打开状态。

默认情况下，其他框架可能会填充这些文件描述符，而这正是我们下一步将要研究的内容。

当处理请求体中上传的文件时，有些框架不区分不同的请求方式，这在一定程度上说明这种攻击方法不仅限于非幂等的HTTP verbs。

PID并不意味着随机化。无论我们的目标是什么（Ubuntu上的Apache，Fedora上的Nginx等），如果我们希望将其转化为漏洞，那么我们可以创建一个本地设置，并查看与

请求处理程序需要访问所有要处理的上传文件的FILES字典
。这就是说，如果处理程序中的功能期望上传的文件是PDF，以执行请求处理程序的中代码，而此时我们也准备上传一个JAR包文件，那么只要同时上传这两个文件即可，它

尝试找到加载文件描述符的请求处理程序，为了我们的代码审计工作能够顺利进行，我们发现有一个处理程序会逐行处理一个文件的全部内容，所以我们上传了一个巨大的文

请注意，如果您上传的文件较小，则可能只读入内存，并且不会打开任何文件描述符。当对Flask
Web框架进行测试时，我们发现1MB以下的文件会被直接加载到内存中，而1MB以上的文件则放在磁盘上。因此，我们需要在JAR包文件中额外填充任何可供攻击利用的有

后续更新
后续我们对多个框架缓慢加载文件描述符这一问题进行了深入的研究和分析。最后我们发现对于Flask和Django来说，并不是FILES
被缓慢加载，而是请求体的内容只有在被访问时才会被处理。因此，根据这个结论我们可以轻松定位那些访问HTTP请求体数据的任何请求处理程序。一旦请求处理程序访问
Django框架中访问请求体数据代码如下所示：

通过此访问请求填充的文件描述符如下所示：

Flask Web框架中访问请求体数据的代码如下所示：

通过此访问请求填充的文件描述符如下所示：

上一篇：数据库安全PPT 下一篇：无弹窗渗透测试实验
1. 0 条回复
   - 动动手指，沙发就是你的了！