《从0到1：CTFer成长之路》1.3 任意文件读取漏洞

原创

ZhShy23 于 2022-01-12 21:15:56 发布 3350 收藏

分类专栏： 《从0到1：CTFer成长之路》 文章标签： php 安全 web安全

本文链接：https://blog.csdn.net/weixin_43651049/article/details/122442526

文章目录

1.3.1 文件读取漏洞常见触发点

1.3.1.1 web语言

1. PHP

重点函数：

- file_get_contents()
- file()
- fopen()函数（及其文件指针操作函数fread()、fgets()等）
- 与文件包含相关的函数
  - include()
  - require()、
  - include_once()
  - require_once()
- 通过PHP读文件的执行系统命令
  - system()
  - exec()

> PHP开发技术越来越倾向于单入口、多层级、多通道的模式，其中涉及PHP文件之间的调用密集且频繁。开发者为了写出一个高复用性的文件调用函数，就需要将一些动态的信息传入（如可变的部分文件名）那些函数（见图1-3-1），如果在程序入口处没有利用switch等分支语句对这些动态输入的数据加以控制，攻击者就很容易注入恶意的路径，从而实现任意文件读取甚至任意文件包含。

php提供文件流—> 协议---->**Wrapper**---->**Filter**对Wrapper进行一定处理

实际情况：

文件路径前面可控，后面不可控

> 使用`\x00`截断，对应的URL编码是“`%00`”，当服务端存在文件上传功能时，也可以尝试利用zip或phar协议直接进行文件包含进而执行PHP代码。

文件路径后面可控，前面不可控

> 通过符号“`../`”进行目录穿越来直接读取文件，这种情况下无法使用Wrapper。如果服务端是利用include等文件包含类函数，将无法读取php文件中的PHP代码。

文件路径中间可控

> 与第一种情况类似，但无法利用Wrapper进行文件包含。

2.python

与PHP不同的是，Python的Web应用更多地倾向于通过其自身的模块启动服务，同时搭配中间件、代理服务将整个Web应用呈现给用户。用户和Web应用交互的过程本身就包含对服务器资源文件的请求，所以容易出现非预期读取文件的情况。因此，我们看到的层出不穷的Python某框架任意文件读取漏洞也是因为缺乏统一的资源文件交互的标准。
漏洞经常出现在框架请求静态资源文件部分，也就是最后读取文件内容的open函数，但直接导致漏洞的成因往往是框架开发者忽略了Python函数的feature，如os.path.join()函数：

```
>>> os.path.join("/a". "/b")
'/b'
```

很多开发者通过判断用户传入的路径不包含“.”来保证用户在读取资源时不会发生目录穿越，随后将用户的输入代入os.path.join的第二个参数，但是如果用户传入“/”，则依然可以穿越到根目录，进而导致任意文件读取。这是一个值得我们注意并深思的地方。

3.Java

除了Java本身的文件读取函数FileInputStream、XXE导致的文件读取，Java的一些模块也支持“file://”协议，这是Java应用中出现任意文件读取最多的地方，如Spring Cloud Config Server路径穿越与任意文件读取漏洞（CVE-2019-3799）、Jenkins任意文件读取漏洞（CVE-2018-1999002）等。

4.Ruby

在CTF线上比赛中，Ruby的任意文件读取漏洞通常与Rails框架相关。到目前为止，我们已知的通用漏洞为Ruby On Rails远程代码执行漏洞（CVE-2016-0752）、Ruby On Rails路径穿越与任意文件读取漏洞（CVE-2018-3760）、Ruby On Rails路径穿越与任意文件读取漏洞（CVE-2019-5418）。笔者在CTF竞赛中就曾遇到Ruby On Rails远程代码执行漏洞（CVE-2016-0752）的利用。

5.Node

目前，已知Node.js的express模块曾存在任意文件读取漏洞（CVE-2017-14849），但笔者还未遇到相关CTF赛题。CTF中Node的文件读取漏洞通常为模板注入、代码注入等情况。

1.3.1.2 中间件、服务件相关

1.Nginx错误配置

```
location /static {
 alias /home/myapp/static/;
}
```

如果配置文件中包含上面这段内容，很可能是运维或者开发人员想让用户可以访问static目录（一般是静态资源目录）。但是，如果用户请求的Web路径是 `/static../` ，拼接到alias上就变成了 `/home/myapp/static/../` ，此时便会产生目录穿越漏洞，并且穿越到了myapp目录。这时，攻击者可以任意下载Python源代码和字节码文件。
注意：漏洞的成因是location最后没有加“/”限制，Nginx匹配到路径static后，把其后面的内容拼接到alias，如果传入的是 `/static../` ，Nginx并不认为这是跨目录，而是把它当作整个目录名，所以不会对它进行跨目录相关处理。

2.数据库

> MySQL的load_file()函数可以进行文件读取，但是load_file()函数读取文件首先需要数据库配置FILE权限（数据库root用户一般都有），其次需要执行load_file()函数的MySQL用户/用户组对于目标文件具有可读权限（很多配置文件都是所有组/用户可读），主流Linux系统还需要Apparmor配置目录白名单（默认白名单限制在MySQL相关的目录下），可谓“一波三折”。即使这么严格的利用条件，我们还是经常可以在CTF线上比赛中遇到相关的文件读取题。
> 还有一种方式读取文件，但是与load_file()文件读取函数不同，这种方式需要执行完整的SQL语句，即load data infile。同样，这种方式需要FILE权限，不过比较少见，因为除了SSRF攻击MySQL这种特殊情形，很少有可以直接执行整条非基本SQL语句（除了SELECT/UPDATE/INSERT）的机会。

3. 软链接

> bash命令 `ln-s` 可以创建一个指向指定文件的软链接文件，然后将这个软链接文件上传至服务器，当我们再次请求访问这个链接文件时，实际上是请求在服务端它指向的文件。

4. FFmpeg

> 2017年6月，FFmpeg被爆出存在任意文件读取漏洞。同年的全国大学生信息安全竞赛实践赛（CISCN）就利用这个漏洞出了一道CTF线上题目（相关题解可以参考https://www.cnblogs.com/iamstudy/articles/2017_quanguo_ctf_web_writeup.html）。

5. Docker-API

> Docker-API可以控制Docker的行为，一般来说，Docker-API通过UNIX Socket通信，也可以通过HTTP直接通信。当我们遇见SSRF漏洞时，尤其是可以通过SSRF漏洞进行UNIX Socket通信的时候，就可以通过操纵Docker-API把本地文件载入Docker新容器进行读取（利用Docker的ADD、COPY操作），从而形成一种另类的任意文件读取。

1.3.1.3 客户端相关

客户端也存在文件读取漏洞，**大多是基于XSS漏洞读取本地文件。**

1. 浏览器/Flash XSS

> 一般来说，很多浏览器会禁止JavaScript代码读取本地文件的相关操作，如请求一个远程网站，如果它的JavaScript代码中使用了File协议读取客户的本地文件，那么此时会由于同源策略导致读取失败。但在浏览器的发展过程中存在着一些操作可以绕过这些措施，如Safari浏览器在2017年8月被爆出存在一个客户端的本地文件读取漏洞。

2. MarkDown语法解析器XSS

> 与XSS相似，Markdown解析器也具有一定的解析JavaScript的能力。但是这些解析器大多没有像浏览器一样对本地文件读取的操作进行限制，很少有与同源策略类似的防护措施。

1.3.2 文件读取漏洞常见读取路径

1.3.2.1 Linux

1.flag名称（相对路径）

对flag名称进行模糊测试

2. 服务器信息（绝对路径）

`/etc` 各种应用或系统配置文件，是进行文件读取的首要目标

`/etc`——各种应用或系统配置文件，是进行文件读取的首要目标

`/etc/passwd`——Linux系统保存用户信息及其工作目录的文件，权限是所有用户/组可读，一般被用作Linux系统下文件读取漏洞*存在性判断*的基准。

1. 系统存在哪些用户
2. 用户所属的组是什么
3. 工作目录是什么

`/etc/shadow`——linux系统保存用户信息及（可能存在）密码（hash）的文件，权限是root用户可读写，shadow组可读。一般情况下不可读

`/etc/apche2/*`——Apache配置文件，可以获知web目录、服务端口等信息

`/etc/nginx/*`——Nginx配置文件（Ubuntu等系统），可以获知web目录、服务端口等信息

`/etc/apparmor(.d)/*`——Apparmor配置文件，可以获知各应用系统调用的白名单、黑名单。*例如，通过读配置文件查看MySQL是否禁止了系统调用，从而确定是否可以使用UDF（User Defined Functions）执行系统命令。*

`/etc/(cron.d/*|crontab)`——定时任务文件。*有些CTF题目会设置一些定时任务，读取这些配置文件就可以发现隐藏的目录或其他文件。*

`/etc/environment`——环境变量配置文件之一。环境变量可能存在大量目录信息的泄露，甚至可能出现secret key泄露的情况。

`/etc/hostname`——主机名。

`/etc/hosts`——主机名查询静态表，包含指定域名解析IP的成对信息。通过这个文件，参赛者可以探测网卡信息和内网IP/域名。

`/etc/issue`——指明系统版本。

`/etc/mysql/*`——MySQL配置文件。

`/etc/php/*`——PHP配置文件。

`/proc目录`

/proc目录通常存储着进程动态运行的各种信息，本质上是一种虚拟目录。**注意：如果查看非当前进程的信息，pid是可以进行暴力破解的，如果要查看当前进程，只需`/proc/self/`代替`/proc/[pid]/`即可。**

对应目录下的cmdline可读出比较敏感的信息，如使用**mysql-uxxx-pxxxx**登录MySQL，会在cmdline中显示明文密码：

```
/proc/[pid]/cmdline                    （[pid]指向进程所对应的终端命令）
```

有时我们无法获取当前应用所在的目录，通过cwd命令可以直接跳转到当前目录：

```
/proc/[pid]/cwd/                       （[pid]指向进程的运行目录）
```

环境变量中可能存在secret_key，这时也可以通过environ进行读取：

```
/proc/[pid]/environ                    （[pid]指向进程运行时的环境变量）
```


其他目录

Nginx配置文件可能存在其他路径：

```
/usr/local/nginx/conf/*                （源代码安装或其他一些系统）
```

日志文件：

```
/var/log/*                             （经常出现 Apache2 的 Web 应用可读/var/log/apache2/access.log
                                       从而分析日志，盗取其他选手的解题步骤）
```

Apache默认Web根目录：

```
/var/www/html/
```

PHP session目录：

```
/var/lib/php(5)/sessions/              （泄露用户 session）
```

用户目录：

```
[user_dir_you_know]/.bash_history      （泄露历史执行命令）
[user_dir_you_know]/.bashrc            （部分环境变量）
[user_dir_you_know]/.ssh/id_rsa(.pub)  （ssh 登录私钥/公钥）
[user_dir_you_know]/.viminfo           （vim 使用记录）
```


[pid]指向进程所对应的可执行文件。有时我们想读取当前应用的可执行文件再进行分析，但在实际利用时可能存在一些安全措施阻止我们去读可执行文件，这时可以尝试读取**/proc/self/exe**。例如：

```
/proc/[pid]/fd/(1|2…)                  （读取[pid]指向进程的 stdout 或 stderror 或其他）
/proc/[pid]/maps                       （[pid]指向进程的内存映射）
/proc/[pid]/(mounts|mountinfo)         （[pid]指向进程所在的文件系统挂载情况。CTF 常见的是 Docker 环境
                                       这时 mounts 会泄露一些敏感路径）
/proc/[pid]/net/*                      （[pid]指向进程的网络信息，如读取 TCP 将获取进程所绑定的 TCP 端口
                                       ARP 将泄露同网段内网 IP 信息）
```

1.3.2.2 Windows

> Windows系统下的Web应用任意文件读取漏洞在CTF赛题中并不常见，但是Windows与PHP搭配使用时存在一个问题：可以使用“<”等符号作为通配符，从而在不知道完整文件名的情况下进行文件读取，这部分内容会在下面的例题中详细介绍。