COMPASS CTF Tutorial 3: Network Protocols and Web Vulnerabilities

COMPASS CTF 教程【3】: Web 漏洞利用专题

30016794 Zhao, Li (Research Assistant)

COMPuter And System Security Lab, Computer Science and Technology Department, College of Engineering (CE), SUSTech University.
南方科技大学 工学院 计算机科学与技术系 计算机与系统安全实验室

2023 年 8 月 7 日

任意文件读取漏洞

所谓文件读取漏洞，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。从整个攻击过程来看，它常常作为资产信息搜集的一种强力的补充手段，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都在这个漏洞触发点被攻击者窥探。[1]

文件读取漏洞常常意味着被攻击者的服务器即将被攻击者彻底控制。当然，如果服务器严格按照标准的安全规范进行部署，即使应用中存在可利用的文件读取漏洞，攻击者也很难拿到有价值的信息。文件读取漏洞在每种可部署 Web 应用的程序语言中几乎都存在。当然，此处的“存在”本质上不是语言本身的问题，而是开发者在进行开发时由于对意外情况考虑不足所产生的疏漏。[2]

任意文件读取漏洞

通常来讲，Web 应用框架或中间件的开发者十分在意代码的可复用性，因此对一些 API 接口的定义都十分开放，以求尽可能地给二次开发者最大的自由。而真实情况下，许多开发人员在进行二次开发时过于信任 Web 应用框架或中间件底层所实现的安全机制，在未仔细了解应用框架及中间件对应的安全机制的情况下，便轻率地依据简单的 API 文档进行开发，不巧的是，Web 应用框架或中间件的开发者可能未在文档中标注出 API 函数的具体实现原理和可接受参数的范围、可预料到的安全问题等。

业界公认的代码库通常被称为“轮子”，程序可以通过使用这些“轮子”极大地减少重复工作量。如果“轮子”中存在漏洞，在“轮子”代码被程序员多次迭代复用的同时，漏洞也将一级一级地传递，而随着对底层“轮子”代码的不断引用，存在于“轮子”代码中的安全隐患对于处在“调用链”顶端的开发者而言几乎接近透明。

任意文件读取漏洞

对于挖掘 Web 应用框架漏洞的安全人员来说，能否耐心对这条“调用链”逆向追根溯源也是一个十分严峻的挑战。
另外，有一种任意文件读取漏洞是开发者通过代码无法控制的，这种情况的漏洞常常由 Web Server 自身的问题或不安全的服务器配置导致。Web Server 运行的基本机制是从服务器中读取代码或资源文件，再把代码类文件传送给解释器或 CGI 程序执行，然后将执行的结果和资源文件反馈给客户端用户，而存在于其中的众多文件操作很可能被攻击者干预，进而造成诸如非预期读取文件、错误地把代码类文件当作资源文件等情况的发生。

Web 语言

不同的 Web 语言，其文件读取漏洞的触发点也会存在差异，我们以读取不同 Web 文件漏洞为例进行介绍，具体的漏洞场景请自行查阅，在此不再赘述。

PHP

PHP 标准函数中有关文件读的部分不再详细介绍，这些函数包括但可能不限于：**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**）。这些函数在 PHP 应用中十分常见，所以在整个 PHP 代码审计的过程中，这些函数会被审计人员重点关注。

Web 语言

这里或许有疑问，既然这些函数这么危险，为什么开发者还要将动态输入的数据作为参数传递给它们呢？因为现在 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（见图文件读取-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。

Web 语言

```php
public static function registerComposerLoader($composerPath)
{
    if (is_file($composerPath . 'autoload_namespaces.php')) {
        $map = require $composerPath . 'autoload_namespaces.php';
        foreach ($map as $namespace => $path) {
            self::addPsr0($namespace, $path);
        }
    }

    if (is_file($composerPath . 'autoload_psr4.php')) {
        $map = require $composerPath . 'autoload_psr4.php';
        foreach ($map as $namespace => $path) {
            self::addPsr4($namespace, $path);
        }
    }

    if (is_file($composerPath . 'autoload_classmap.php')) {
        $classMap = require $composerPath . 'autoload_classmap.php';
        if ($classMap) {
            self::addClassMap($classMap);
        }
    }
```

图: 文件读取-1

Web 语言

除了上面提到的标准库函数，很多常见的 PHP 扩展也提供了一些可以读取文件的函数。例如，**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 等。这些通过外部库函数进行任意文件读取的 CTF 题目不是很多，后续章节会对涉及的题目进行实例分析。

与其他语言不同，PHP 向用户提供的指定待打开文件的方式不是简简单单的一个路径，而是一个文件流。我们可以将其简单理解成 PHP 提供的一套协议。例如，在浏览器中输入 **http://host：port/xxx** 后，就能通过 HTTP 请求到远程服务器上对应的文件，而在 PHP 中有很多功能不同但形式相似的协议，统称为 **Wrapper**，其中最具特色的协议便是 php://协议，更有趣的是，PHP 提供了接口供开发者编写自定义的 **wrapper（stream_wrapper_register）**。

Web 语言

除了 Wrapper，PHP 中另一个具有特色的机制是 Filter，其作用是对目前的 Wrapper 进行一定的处理（如把当前文件流的内容全部变为大写）。
对于自定义的 Wrapper 而言，Filter 需要开发者通过 stream_filter_register 进行注册。而 PHP 内置的一些 Wrapper 会自带一些 Filter，如 php://协议存在图文件读取-2 中所示类型的 Filter。


List of Available Filters

Table of Contents

- String Filters
- Conversion Filters
- Compression Filters
- Encryption Filters

图: 文件读取-2

Web 语言

PHP 的 Filter 特性给我们进行任意文件读取提供了很多便利。假设服务端 include 函数的路径参数可控，正常情况下它会将目标文件当作 PHP 文件去解析，如果解析的文件中存在“<**?php**”等 PHP 的相关标签，那么标签中的内容会被作为 PHP 代码执行。

我们如果直接将这种含有 PHP 代码的文件的文件名传入 include 函数，那么由于 PHP 代码被执行而无法通过可视文本的形式泄露。但这时可以通过使用 Filter 避免这种情况的发生。

例如，比较常见的 Base64 相关的 Filter 可将文件流编码成 Base64 的形式，这样读取的文件内容中就不会存在 PHP 标签。而更严重的是，如果服务端开启了远程文件包含选项 allow_url_include，我们就可以直接执行远程 PHP 代码。

Web 语言

当然，这些 PHP 默认携带的 Wrapper 和 Filter 都可以通过 php.ini 禁用，读者在实际遇到时要具体分析，建议阅读 PHP 有关 Wrapper 和 Filter 的源代码，会更加深入理解相关内容。

在遇到的有关 PHP 文件包含的实际问题中，我们可能遇到三种情况： 文件路径前面可控，后面不可控； 文件路径后面可控，前面不可控； 文件路径中间可控。

对于第一种情况，在较低的 PHP 版本及容器版本中可以使用“\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。

Web 语言

对于第二种情况，我们可以通过符号“../”进行目录穿越来直接读取文件，但这种情况下无法使用 Wrapper。如果服务端是利用 include 等文件包含类的函数，我们将无法读取 PHP 文件中的 PHP 代码。
第三种情况与第一种情况相似，但是无法利用 Wrapper 进行文件包含。

Web 语言

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'
```

Web 语言

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。

除了 python 框架容易出这种问题，很多涉及文件操作的应用也很有可能因为滥用 open 函数、模板的不当渲染导致任意文件读取。比如，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，这就给了攻击者一个进行目录穿越的途径。

Web 语言

例如，CTF 线上比赛中，Python 开发者调用不安全的解压模块进行压缩文件解压，而导致文件解压后可进行目录穿越。当然，解压文件时的目录穿越的危害是覆写服务器已有文件。
另一种情况是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。这将在后面章节中详细分析。与 PHP 相同，Python 的一些模块可能存在 XXE 读文件的情况。此外，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，当然，其最大危害仍然是导致任意命令执行。

Web 语言

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。

Web 语言

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）。

Web 语言

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），CTF 中 Node 的文件读取漏洞通常为模板注入、代码注入等情况。

中间件/服务器相关

不同的中间件/服务器同样可能存在文件读取漏洞，我们以曾经出现的不同中间件/服务器上的文件读取漏洞为例来介绍。具体的漏洞场景请自行查阅，在此不再赘述。

Nginx 错误配置

Nginx 错误配置导致的文件读取漏洞在 CTF 线上比赛中经常出现，尤其是经常搭配 Python-Web 应用一起出现。这是因为 Nginx 一般被视为 Python-Web 反向代理的最佳实现。然而它的配置文件如果配置错误，就容易造成严重问题。例如：

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

中间件/服务器相关

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。

中间件/服务器相关

数据库

可以进行文件读取操作的数据库很多，这里以 MySQL 为例来进行说明。

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。

中间件/服务器相关

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。

中间件/服务器相关

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**）。

中间件/服务器相关

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。

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。

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。

客户端相关

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。

Linux

flag 名称（相对路径）

比赛过程中，有时 fuzz 一下 flag 名称便可以得到答案。注意以下文件名和后缀名，请根据题目及环境自行发挥。

```
../../../../../../../../../flag(.txt|.php|.pyc|.py …)
flag(.txt|.php|.pyc|.py …)
[dir_you_know]/flag(.txt|.php|.pyc|.py …)
../../../../../../../../../etc/flag(.txt|.php|.pyc|.py …)
../../../../../../../../../tmp/flag(.txt|.php|.pyc|.py …)
../flag(.txt|.php|.pyc|.py …)
../../../../../../../../../root/flag(.txt|.php|.pyc|.py …)
../../../../../../../../../home/flag(.txt|.php|.pyc|.py …)
../../../../../../../../../home/[user_you_know]/flag(.txt|.php|.pyc|.py …)
```

Linux

服务器信息（绝对路径）

下面列出 CTF 线上比赛常见的部分需知目录和文件。建议在阅读后亲自翻看这些目录，对于未列出的文件也建议了解一二。

(1) /etc 目录

/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。

(2)/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。

Linux

(3)/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。
(4)/etc/apache2/*
/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。

Linux

(5)/etc/nginx/*

/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。

(6)/etc/apparmor(.d)/*

/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。

(7)/etc/(cron.d/*|crontab)

/etc/(cron.d/*|crontab) 是定时任务文件。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。

Linux

(8)/etc/environment
/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。
(9)/etc/hostname
/etc/hostname 表示主机名。
(10)/etc/hosts
/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。

Linux

(11)/etc/issue
/etc/issue **指明系统版本**。
(12)/etc/mysql/*
/etc/mysql/***是 MySQL 配置文件**。
(13)/etc/php/*
/etc/php/***是 PHP 配置文件**。

Linux

(14) /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）
```

Linux

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）
```

Linux

(15) 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/
```

Linux

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）
```

Linux

[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）

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。

参考文献

[1] Nu1L. 从 0 到 1：CTFer 成长之路 [EB/OL].
https://book.douban.com/subject/35200558/.

[2] MI L. 4.7. 文件读取 [EB/OL].
https://websec.readthedocs.io/zh/latest/vuln/fileread.html.