从 ByteCTF 到 bypass_disable_function | Z3ratu1's blog

从 ByteCTF 到 bypass_disable_function 起初只是简单的看了一下去年 ByteCTF 中那个 bypass_disable_fuction 的题，然后发现原来 PHP 有这么多 bypass 的方.....

起初只是简单的看了一下去年 ByteCTF 中那个 bypass_disable_fuction 的题，然后发现原来 PHP 有这么多 bypass 的方案我不知道，以及一些零碎的其他知识点，统一学习了

然后马上蓝帽杯遇到一个 bypass 发现完全不会做，进行大型 update

putenv

在环境变量可控时执行的两大操作，需要 disable_function 里面给用 putenv

LD_PRELOAD

这个是最老的操作之一了，我掌握的也就这一种，通过 `LD_PRELOAD` 这个变量可以指定动态链接库 (.so 文件) 在 libc 之前被加载，编写一个恶意的动态链接库，使用 GNU 的特殊语法，在调用 main 函数之前对构造属性进行执行，由于是调的 C 库这边的函数，自然不受 disable_function 的影响。只要 PHP 启动新进程时，恶意动态链接库被加载，且其构造方法直接在一切之前运行，构造方法中写一个命令执行即可。
简单的说就是能劫持 PHP 启动的新进程

mail

PHP 使用 mail 函数的时候会用 execve 来启动 sendmail，启动进程时成功命令执行

imap_open

也是一个启动进程的函数

Imagick

这个库在远古版本是能直接命令执行的，能直接绕过 disable function，不过那个也太远古了。但是这个库也能启动新的进程，而非常有意思的一点是，LD_PRELOAD 那篇文章的作者却提到他没能在 imagick 中启动新的进程，不过 0CTF2019 中的这道 Wallbreaker_Easy 考的就是这个点（这个题用的 FPM 模式，估计用 FPM 也能打通），Imagick 再进行图片类型转换时，需要启动外部程序进行转换，这样就会启动新的进程，实现 LD_PRELOAD 的命令执行，这里放一个替换 mail 的 payload

```
$a=new Imagick();
$a->readImage('123.png');
$a->writeImage('123.wdp');
```

可以用之前的 bypass 脚本打通

iconv

这个是 ByteCTF 的考点，基本原理也是最后的底层实现调用的是 C 的动态链接库。
PHP 在使用 `iconv()` 时最后一路调用，进入 libc 函数 `iconv_open()` ，再一波操作调用到. so 文件的方法，实现 RCE
同样，系统不会无缘无故的调用我们自己上传的路径下的 `.so` 文件，其支持使用 `GCONV_PATH` 的自定义编码转换模块，所以可以 putenv 将该变量设置为目标路径，同时上传两个文件以建立该编码模块并实现利用

glibc iconv 文档 讲解了 gconv-modules 的基本语法

这里默认后缀是. so 所以不需要指定. so，默认认为动态链接库和 gconv-modules 在同一目录下，但是也允许相对目录，编码名后面加的 `//` 是 glibc 实现的问题，文档里说你听他的就行了

gconv-modules

```
module  PAYLOAD//    INTERNAL    ../../../../../../../../tmp/payload    2
module  INTERNAL    PAYLOAD//    ../../../../../../../../tmp/payload    2
```

payload.c，使用 `gcc payload.c -o payload.so -shared -fPIC` 编译为 payload.so

```
#define _GNU_SOURCE

#include
#include
#include

extern char** environ;

void gconv() {}

void gconv_init() {
    const char* cmdline = getenv("EVIL_CMDLINE");
    int i;
    for (i = 0; environ[i]; ++i) {
            if (strstr(environ[i], "LD_PRELOAD")) {
                    environ[i][0] = '\0';
            }
    }

    system(cmdline);
}
```

此时在 iconv 中遇到 `payload` 编码时，即会调用我们这个恶意的动态链接库完成命令执行

除了 `iconv()` 这个函数，所有能最终调用到 libc 的 `iconv_open()` 操作均能触发 RCE，比如 `iconv_strlen` ，或者 `php://filter` 的 `convert.iconv` 过滤器等
这道题的预期解应该就是用 php://filter
lfy 神仙还提到了可以再套一层 LD_PRELOAD 劫持 system 这个函数启动的进程方便执行命令，但是我感觉只要自己把 payload 里面写好一点靠环境变量作为参数执行命令应该也很方便吧？这里直接抄那个 LD_PRELOAD 的 payload 靠环境变量命令执行也很不错

写 / proc/self/mem

好像以前见到过类似的题目，偏二进制
PHP 主进程是 root 的，但是子进程是 www-data 的，/proc/self/mem 属于 www-data 且权限是 600，但 /proc/self 目录是 root 的，并且 www-data 无权限，所以正常情况下不能写入。
但是对于 Nginx+php，且为低版本的 php-fpm（PHP<5.6），/proc/self / 属于 www-data，可以通过写入 GOT 表的 RCE

二进制基础

二进制文件执行函数的时候要查两个表，一个 PLT 一个 GOT。PLT 是在编译时就确定下来了的，加载进内存的时候位于代码段。但由于动态链接之类的存在，在编译的时候并不能确定所有函数的地址，因为它甚至都还没加载进来，所以 PLT 表项并不存放函数的地址，而是指向 GOT 表的对应项，再由 GOT 表指向函数的真实地址。GOT 表就是在运行时当调用一个函数的时候临时去查询的，因为 PLT 处于代码段不可修改，所以查到之后回填进 GOT 表，下次调用该函数就可以直接查询 GOT 表获取到地址。期间还有各种复杂的操作，比如什么 GOT 表前几项是就是用来进行函数地址查询的函数之类的，还有一种程序运行时不是等需要用到再去填 GOT 表，而是直接全部加载完然后把 GOT 表也变得只读防止被修改之类的

利用

从上面已经可以看出来函数调用时其所用的是 GOT 表指向的地址，那么只要修改 GOT 表中某个函数，在执行那个函数的时候变成执行 system 之类的就搞定了（修改 GOT 表应该是 pwn 那边的常见操作才是。。。。）

抄一个攻击流程

写一下劫持 GOT 表的步骤，这里直接写 shellcode：

1. 读 / proc/self/maps 找到 php 和 libc 在内存中的基址
2. 解析 / proc/self/exe 找到 php 文件中 readfile@got 的偏移
3. 找个能写的地址写 shellcode
4. 向 readfile@got 写 shellcode 地址覆盖
5. 调用 readfile

利用脚本

攻击 PHP-FPM

先了解一下前置知识

PHP 运行类型

PHP 运行一般来说几个类型，CLI，php-cgi，php-fpm，Apache2.0handler

CLI 是 Command Line Interface，命令行情况下使用，不怎么常见

CGI 是 Common Gateway Interface，webserver 和其他软件通信的**协议**（感觉是配合 Nginx 做反带之类的时候使用的），有一个强化版本 fast-cgi。php 的 cgi 就是来一个请求 PHP 起一个解释器进程处理，处理完了关掉，就很慢很敷批，所以 fast-cgi 就是开启直接启动多个解释器进程常驻内存，等待接收请求，处理完了也不关，继续等下一次的请

php-cgi 是早期的 cgi 管理器，因为 cgi 不太行所以这个也不太行

php-fpm 是 fast-cgi 的管理器，一个 master 进程和一堆 worker 进程，master 接收请求分配给 worker，能动态调度启动 worker 进程，性能 upup

使用 Apache 搭建 PHP 的时候使用的是 Apache2handler，这个时候 Apache 是把 PHP 作为一个 module 加载进来的，属于是 Apache 自己内嵌 PHP 的解释器，就少了一步进程间通信，Apache 直接自己开 PHP 解释器进行处理

协议字段

在通信时有几个比较有意思的字段是可以指定的

`SCRIPT_FILENAME`，指定 PHP 执行的脚本文件路径，不过 php5.3.9 之后加入了 fpm 增加了 security.limit_extensions，只允许执行如下后缀的文件 `.php .php3 .php4 .php5 .php7`

`PHP_VALUE`，可以覆盖一些 php.ini 里面定义的属性，只能用于 `PHP_INI_ALL` 或 `PHP_INI_PERDIR` 类型的指令，具体看这个 php.ini 配置选项列表，好用的比如 `auto_prepend_file`，`open_basedir`

`PHP_ADMIN_VALUE`，和上面这个差不多，区别在于这个字段设置的属性不能在用户层面上被修改，也就是不能被 ini_set() 之类的函数在应用里被重写，也不能被. htaccess 这种配置文件覆盖，常用的有 `allow_url_include`，启用后支持 include url 形式的文件，经典 php://input 伪协议打通，以及 `extension_dir`，指定扩展的. so 直接 bypass disable_function 打通，`safe_mode`，在 PHP5.4 之后就被删除了的东西，启用后会限制某些函数的使用，比如 move_upload_file,copy 这些能把远程文件下到本地的函数，和一些 system，shell_exec 这类直接的命令执行函数，也不允许进行 dl 函数加载扩展文件 (.dll or .so)

（找不到资料，但是感觉也许可以把低安全等级的变量也设置成 PHP_ADMIN_VALUE 不让动态的修改，不过 `PHP_INI_SYSTEM` 这个类型的属性理论上只允许在 php.ini 里面设置，这里能改真是玄幻）

与之对位的还有 `PHP(_ADMIN)_FLAG` 这么个属性，其区别在于其值只能为布尔值

可惜的是这些属性并不能覆盖 disable_function

> PHP 配置值通过 php_value 或者 php_flag 设置，并且会覆盖以前的值。请注意 disable_functions 或者 disable_classes 在 php.ini 之中定义的值不会被覆盖掉，但是会将新的设置附加在原有值的后面。
> 使用 php_admin_value 或者 php_admin_flag 定义的值，不能被 PHP 代码中的 ini_set() 覆盖。

修改 PHP_(ADMIN_)VALUE 会直接使得当前处理请求的 fastCGI 进程收到影响，如果多次请求就可能污染掉所有的进程，在重启 fpm 之前可能所有进程都会受到影响

利用

因为之前说到使用 CGI 的时候，是 webserver 和 CGI 直接进行通信的，所以就存在进程间通信的问题，在 Nginx 的设置中有一个 fastcgi_pass 的设置，指定 fastcgi 所在的 ip 端口 (或 Unix socket)，PHP 的 fpm 设置中可以配置 fastcgi 的监听位置，可以通过配置使得 PHP 也能前后端分离
fpm 并不验证数据的来源，只要是发送到监听端口的数据就一律接受
PHP 在 fpm 配置中若将监听端口写成 0.0.0.0:9000，则接受来自任意 ip 的通信，可以通过伪装成 Nginx 服务器与 php-fpm 通信来执行命令，如果配置的是 127.0.0.1:9000 这种情况的可能就要靠 SSRF 打了
P 神的利用脚本

SSRF 也分两种，如果是监听的 127.0.0.1 这种 ip 地址就可以用 gopher 发 TCP 包打，如果直接监听 Unix socket 的话就得专门起一个 socket 去连接了

利用方案：1. 自己随便上传一个其他位置的马然后用 script_name 指定进行利用
2. 随便找一个 PHP 文件然后设置 auto_prepend_file 为 php://input，allow_url_include 为 On 打通

bypass_disable_function

直接拙着那个脚本打并不能绕过 Disable_function，只是做到了命令执行，因为这样子发过去的请求还是由原来的 PHP

直接抄看那个脚本的并不能绕过 disable_function，只是做到了命令执行。因为选择了发过去的请求还是由原来的 FPM 解释器进行解析

但是可以通过 FastCGI 协议去让 php-fpm 加载我们自定义的扩展 (.so 文件)，而这个扩展肯定是不受 disable_function 限制的，做到任意命令执行
即之前提到的 `extension_dir` 和 `extension`

蚁剑有一个玄幻的插件，直接生成一个调用 system 函数的. so 文件，然后输这么个命令 `php -n -S 127.0.0.1:port -t /var/www/html` 在目标机器上新开一个 web 服务，当然权限是 www-data 的，但是添加了 `-n` 参数指定不使用 php.ini，可以绕过其设置的 disable_function，并且还上传了一个 antproxy.php 文件将请求转发过去，实现比较方便的命令执行（不然应该是执行一次传一个. so 吧，要我说继续搞那种接受环境变量做命令的方法也挺好的）
这次这个蓝帽杯本身是一个打. so 扩展的 pwn 题，但是我看的 wp 成功的拿 fpm 打了一个. so 加载的非预期打通了

FFI

Foreign Function Interface，外部函数接口
PHP7.4 的新特性，可以直接调用外部代码，RCTF 2019 的 Nextphp 出的就是这个
需要 ffi.enable=true 才能任意使用，否则只能使用在对应文件中使用

参考链接

ByteCTF WP – 无需 mail bypass disable_functions
PHP 突破 disable_functions 常用姿势以及使用 Fuzz 挖掘含内部系统调用的函数
php 之 CGI、FastCGI、APACHE2HANDLER、CLI 运行模式的详解
攻击 PHP-FPM 实现 Bypass Disable Functions
RASP 攻防 —— RASP 安全应用与局限性浅析
bypass disable_function 多种方法 + 实例

浅析 php-fpm 的攻击方式

从一道 CTF 学习 Fastcgi 绕过姿势

PHP 连接方式介绍以及如何攻击 PHP-FPM

Fastcgi 协议分析 && PHP-FPM 未授权访问漏洞 && Exp 编写

PHP 绕过 open_basedir 列目录的研究