Challenge Name: Rookie

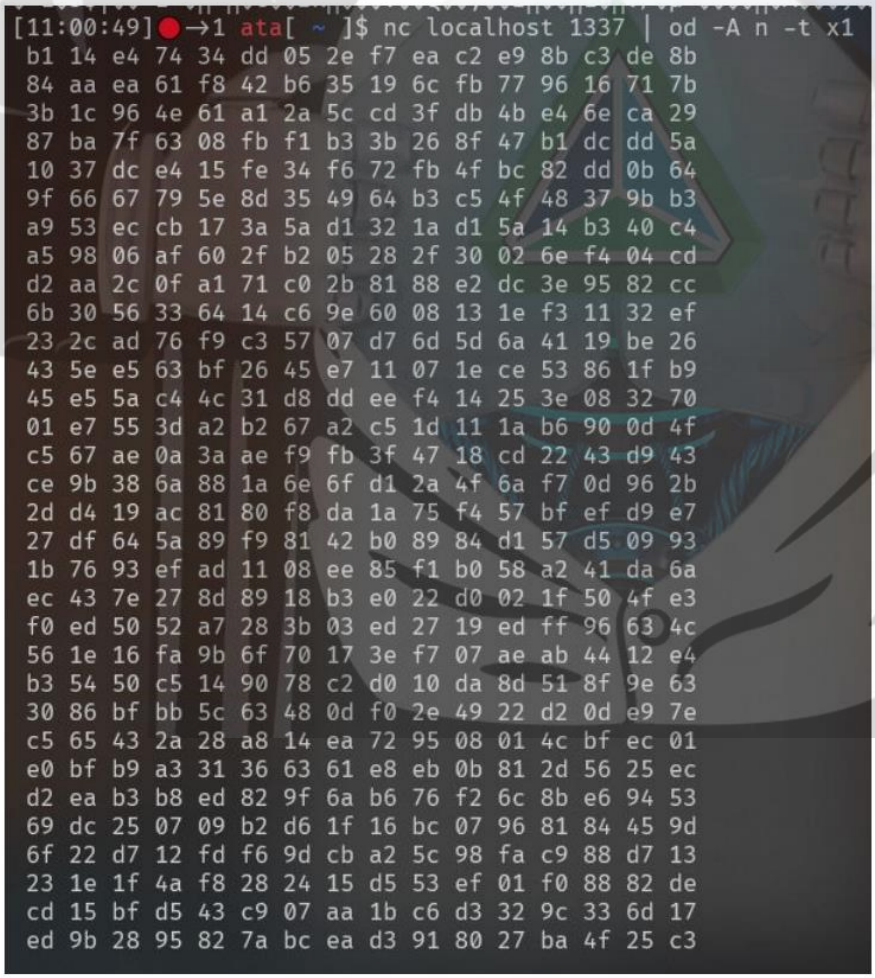
Category: Reverse

Flag: STMCTF{dOub1E_sUdo_t0_b4ckd00r_0r_ju5t_a_r0okIe_M1st4ke?}

For this challenge, a private key, and a binary is provided, as well as a connection address.

Connecting to the remote address directly is not very helpful.

With each packet sent, server responds with 512 byte responses



Looking at the main function of the binary, you can see it expects argc to be 2, and executes atoi on argv[1]. This value is then used to create a socket.

```
signal(17, (__sighandler_t)((char *)&size + 1));
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v19);
if ( a4 == 2 )
{
  v12 = atoi(a1[1]);
  ::fd = socket(2, 1, 0);
  if ( ::fd >= 0 )
  {
    addr.sa_family = 2;
    *(_DWORD *)&addr.sa_data[2] = 0;
    *(_WORD *)addr.sa_data = htons(v12);
    addr_len = 16;
    v11 = 0;
```

Running the binary again with ltrace, and giving a cli parameter, its easy to see it creates a bind listen. At the given port.

```
[11:02:23] ata[ /mnt/c/VM/Share/stmctf/rookie ]$ ltrace bin/rookie-server 1337
_ZNSt8ios_base4InitC1Ev(0x7fb37d0062f1, 0xffff, 0x7fffe6018090, 128)     = 0
__cxa_atexit(0x7fb37c5039e0, 0x7fb37d0062f1, 0x7fb37d006008, 6)          = 0
_ZNSt8ios_base4InitC1Ev(0x7fb37d0062fc, 0xffff, 0x7fffe6018090, 160)     = 2
__cxa_atexit(0x7fb37c5039e0, 0x7fb37d0062fc, 0x7fb37d006008, 160)         = 0
_ZNSt8ios_base4InitC1Ev(0x7fb37d0062fd, 0xffff, 0x7fffe6018090, 192)     = 3
__cxa_atexit(0x7fb37c5039e0, 0x7fb37d0062fd, 0x7fb37d006008, 192)         = 0
_ZNSt8ios_base4InitC1Ev(0x7fb37d0062fe, 0xffff, 0x7fffe6018090, 224)     = 4
__cxa_atexit(0x7fb37c5039e0, 0x7fb37d0062fe, 0x7fb37d006008, 224)         = 0
signal(SIGINT, 0x7fb37ce02ebc)                                           = 0
signal(SIGABRT, 0x7fb37ce02ebc)                                          = 0
signal(SIGTERM, 0x7fb37ce02ebc)                                          = 0
signal(SIGCHLD, 0x1)                                                     = 0
_ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEC1Ev(0x7fffe6017ed0, 0x7fffe6017be0, 0, 0) = 0x7fffe6017ee0
atoi(0x7fffe60182b1, 0x7fffe6017be0, 0, 0)                               = 1337
socket(2, 1, 0)                                                          = 3
htons(1337, 1, 0, 0x7fb37bfdecc0)                                        = 0x3905
bind(3, 0x7fffe6017eb0, 16, 0x7fffe6017eb0)                              = 0
listen(3, 5, 16, 0x7fb37bfdecc0)                                         = 0
accept(3, 0x7fffe6017ec0, 0x7fffe6017e90, 0x7fffe6017ec0
```

Ltrace showing bind listen

Connecting to this port with netcat gives a similar result to the host address given in the challenge text. Each connection is being forked, and we need ltrace -f to track each children.

```
fork()                                                                   = 317
close(4)                                                                 = 0
accept(3, 0x7fffe6017ec0, 0x7fffe6017e90, 0x7fffe6017ec0
```

Fork is not hooked by ltrace without -f flag

Running again with ltrace -f to track forks, it looks like openssl is being spawned as a child process.

```
[pid 333] <... getenv resumed> )                                    = nil
[pid 333] CRYPTO_mem_ctrl(1, 106, 0×7f8a98079993, 19 <unfinished ...>
[pid 332] <... __freading resumed> )                                = 0
[pid 332] __freading(0×7fffdcbec680, 0, 4, 2880 <unfinished ...>
[pid 333] <... CRYPTO_mem_ctrl resumed> )                           = 0
[pid 333] getenv("OPENSSL_FIPS" <unfinished ...>
[pid 332] <... __freading resumed> )                                = 0
[pid 332] fflush(0×7fffdcbec680 <unfinished ...>
[pid 333] <... getenv resumed> )                                    = nil
[pid 333] signal(SIGPIPE, 0×1 <unfinished ...>
[pid 332] <... fflush resumed> )                                    = 0
[pid 332] fclose(0×7fffdcbec680 <unfinished ...>
[pid 333] <... signal resumed> )                                    = 0
[pid 333] OPENSSL_init_crypto(0×7640, 0, 0, 0 <unfinished ...>
[pid 332] <... fclose resumed> )                                    = -1
[pid 332] __errno_location()                                        = 0×7fffdce914c0
[pid 332] +++ exited (status 0) +++
[pid 331] --- SIGCHLD (Child exited) ---
[pid 331] <... wait resumed> )                                      = 332
[pid 331] wait(0 <unfinished ...>
[pid 333] <... OPENSSL_init_crypto resumed> )                       = 1
[pid 333] UI_create_method(0×7f8a98071880, 1, 0×7fffc2e6719c, 1)    = 0×7fffc2e6c130
[pid 333] UI_method_set_opener(0×7fffc2e6c130, 0×7f8a9802fbd0, 245, 0)  = 0
[pid 333] UI_method_set_reader(0×7fffc2e6c130, 0×7f8a9802fb50, 245, 0)  = 0
[pid 333] UI_method_set_writer(0×7fffc2e6c130, 0×7f8a9802fad0, 245, 0)  = 0
[pid 333] UI_method_set_closer(0×7fffc2e6c130, 0×7f8a9802fab0, 245, 0)  = 0
[pid 333] qsort(0×7f8a982921c0, 112, 32, 0×7f8a98047ad0 <unfinished ...>
[pid 333] strcmp("ca", "ciphers")                                   = -8
[pid 333] strcmp("asn1parse", "ca")                                 = -2
[pid 333] strcmp("cms", "crl")                                      = -5
[pid 333] strcmp("crl2pkcs7", "dgst")                               = -1
[pid 333] strcmp("cms", "crl2pkcs7")                                = -5
[pid 333] strcmp("crl", "crl2pkcs7")                                = -50
[pid 333] strcmp("asn1parse", "cms")                                = -2
[pid 333] strcmp("ca", "cms")                                       = -12
[pid 333] strcmp("ciphers", "cms")                                  = -4
```

Ltrace -f output

Looking trough the defined strings, its easy to notice openssl rsautl is being used to encrypt whatever is being send.

```
.rodata:000000... 00000008 C openssl
.rodata:000000... 00000007 C rsautl
.rodata:000000... 00000009 C -encrypt
.rodata:000000... 00000007 C -inkey
.rodata:000000... 00000010 C cert/public.pem
.rodata:000000... 00000007 C -pubin
.rodata:000000... 00000005 C echo
```

Defined strings showing openssl rsautl call

Following references to the address, this is the function using openssl.

```
1  void __fastcall __noreturn sub_28E2(int *a1)
2  {
3    char *file; // [rsp+10h] [rbp-40h]
4    const char *v2; // [rsp+18h] [rbp-38h]
5    const char *v3; // [rsp+20h] [rbp-30h]
6    const char *v4; // [rsp+28h] [rbp-28h]
7    const char *v5; // [rsp+30h] [rbp-20h]
8    const char *v6; // [rsp+38h] [rbp-18h]
9    __int64 v7; // [rsp+40h] [rbp-10h]
10   unsigned __int64 v8; // [rsp+48h] [rbp-8h]
11
12   v8 = __readfsqword(0x28u);
13   close(0);
14   close(2);
15   dup(*a1);
16   close(a1[1]);
17   close(*a1);
18   file = "openssl";
19   v2 = "rsautl";
20   v3 = "-encrypt";
21   v4 = "-inkey";
22   v5 = "cert/public.pem";
23   v6 = "-pubin";
24   v7 = 0LL;
25   execvp("openssl", &file);
26   exit(0);
27 }
```

Renaming this function to openssl_execvp to ease our future work.

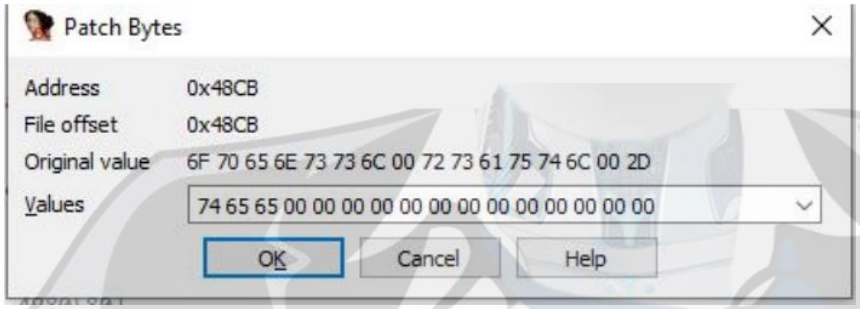
This function has 2 references, one echoes input string and pipes it to openssl, and other passes given arguments to execvp and pipes stdout to openssl.

```
1  __pid_t __fastcall echo_openssl(__int64 a1)
2  {
3    int pipedes[2]; // [rsp+18h] [rbp-38h]
4    char *file; // [rsp+20h] [rbp-30h]
5    const char *v4; // [rsp+28h] [rbp-28h]
6    __int64 v5; // [rsp+30h] [rbp-20h]
7    __int64 v6; // [rsp+38h] [rbp-18h]
8    unsigned __int64 v7; // [rsp+48h] [rbp-8h]
9
10   v7 = __readfsqword(0x28u);
11   if ( pipe(pipedes) == -1 )
12     return 0;
13   if ( fork() == 0 )
14   {
15     close(1);
16     close(2);
17     dup(pipedes[1]);
18     close(pipedes[0]);
19     close(pipedes[1]);
20     v6 = 0LL;
21     file = "echo";
22     v4 = "-n";
23     v5 = a1;
24     execvp("echo", &file);
25     exit(1);
26   }
27   if ( fork() == 0 )
28     openSSL_execvp(pipedes);
29   close(pipedes[0]);
30   close(pipedes[1]);
31   wait(0LL);
32   return wait(0LL);
33 }
```

```
__int64 __fastcall proc_pipeto_openssl(__int64 a1,
{
  char **argv; // [rsp+8h] [rbp-38h]
  __WAIT_STATUS stat_loc; // [rsp+2Ch] [rbp-14h]
  int v6; // [rsp+34h] [rbp-Ch]
  unsigned __int64 v7; // [rsp+38h] [rbp-8h]

  argv = a3;
  v7 = __readfsqword(0x28u);
  if ( pipe((int *)&stat_loc.__iptr + 1) == -1 )
    return 1LL;
  if ( fork() == 0 )
  {
    close(1);
    close(2);
    dup(v6);
    execvp(*argv, argv);
    exit(1);
  }
  if ( fork() == 0 )
    openSSL_execvp((int *)&stat_loc.__iptr + 1);
  close(SHIDWORD(stat_loc.__iptr));
  close(v6);
  wait((__WAIT_STATUS)&stat_loc);
  wait((__WAIT_STATUS)&stat_loc);
  return LODWORD(stat_loc.__uptr);
}
```

To test it locally, we can patch out openssl part. We can modify the execvp char** as { “tee”, 0} or {“tee”, “”,””,””,””,””}, so whatever is piped to the stdin gets echoed out directly, instead of being encrypted.



Patching openssl as tee

```
.rodata:00000000000048CB aOpenssl        db 'tee',0,0,0,0,0      ; DATA XREF: openSSL_execvp+5A↑o
.rodata:00000000000048D3 aRsautl         db 0,0,0,0,0,0,0        ; DATA XREF: openSSL_execvp+65↑o
.rodata:00000000000048DA aEncrypt        db 0,0,0,0,0,0,0,0,0,0  ; DATA XREF: openSSL_execvp+70↑o
.rodata:00000000000048E3 aInkey          db 0,0,0,0,0,0,0,0      ; DATA XREF: openSSL_execvp+7B↑o
.rodata:00000000000048EA aCertPublicPem  db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.rodata:00000000000048EA                                         ; DATA XREF: openSSL_execvp+86↑o
.rodata:00000000000048FA aPubin          db 0,0,0,0,0,0,0        ; DATA XREF: openSSL_execvp+91↑o
.rodata:0000000000004901 asc_4901        db 0Ah,0                ; DATA XREF: sub_2911+2B↑o
.rodata:0000000000004901                                         ; sub_2911+47↑o
.rodata:0000000000004903 aEcho           db 'echo',0             ; DATA XREF: echo_openssl+9F↑o
```

Final patch

Address	Length	Original bytes	Patched bytes
000000000000...	0x2	6F 70	74 65
000000000000...	0x4	6E 73 73 6C	00 00 00 00
000000000000...	0x6	72 73 61 75 74 6C	00 00 00 00 00 00
000000000000...	0x8	2D 65 6E 63 72 79 70 74	00 00 00 00 00 00 00 00
000000000000...	0x6	2D 69 6E 6B 65 79	00 00 00 00 00 00
000000000000...	0xf	63 65 72 74 2F 70 75 62 6C 69 63 2E 70 65 6D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000000000...	0x6	2D 70 75 62 69 6E	00 00 00 00 00 00

Patched bytes After applying the patch, rerunning the binary, and connecting to the socket, we can see its a bind shell.

```
[11:30:45] →1 ata[ ~ ]$ nc localhost 1337
ata@DESKTOP-5K61N76:/mnt/c/VM/Share/stmctf/rookie$
```

Now, after restoring some of the functionality, it should be easier to reverse engineer the rest of the binary

Main function is forking into a new function at 30E5

```
        v15 = (unsigned int)fd | 0x3E800000000LL;
        v14 = fork();
        if ( !v14 )
          sub_30E5(v15, v16);
        close(fd);
      }
      v4 = 1;
    }
    else
```

This function is calling sub_2D07 and forking pty session, calling sub_23A6, then splitting execution.

```
*(_QWORD *)fd = a1;
v6 = a2;
v15 = __readfsqword(0x28u);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v12);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v13);
sub_2D07((unsigned int)a1, 180LL);
v2 = &amaster;
v8 = forkpty(&amaster, 0LL, 0LL, 0LL);
sub_23A6(&amaster);
if ( v8 )
{
  if ( v8 < 0 )
    exit(122);
  sub_2D6C(fd, (unsigned int)amaster);
}
if ( fd[0] >= 0 )
{
  while ( fd[0] )
  {
    v9 = sub_441B(v2);
    bzero(&s, 0x200uLL);
    v9 = read(fd[0], &s, 0x200uLL);
    if ( v9 == -1 )
    {
      v4 = __cxa_allocate_exception(4uLL);
      *v4 = 1;
      __cxa_throw(v4, (struct type_info *)&`typeinfo for'int, 0LL);
    }
    v11 = sub_468B(&s);
    v2 = *(int **)fd;
    v10 = sub_45BE(*(_QWORD *)fd, v6, v11);
  }
  shutdown(0, 1);
  close(0);
  wait(0LL);
  wait(0LL);
  dword_2062F8 = 1;
  exit(1);
}
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v13);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v12);
```

2D07 is just setting socket timeout to 3 minutes. 23A6 is dropping root privileges, but only effective uid. This process can restore root privileges with setuid(0).

```
int __fastcall sub_2D07(int a1, int a2)
{
  __int64 optval; // [rsp+10h] [rbp-20h]
  __int64 v4; // [rsp+18h] [rbp-18h]
  unsigned __int64 v5; // [rsp+28h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  optval = a2;
  v4 = 0LL;
  return setsockopt(a1, 1, 20, &optval, 0x10u);
}
```

```
int sub_23A6()
{
  __gid_t groups; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  groups = 1000;
  setgroups(1uLL, &groups);
  setgid(groups);
  setuid(0);
  return seteuid(0x3E8u);
}
```

Function called from ptyfork session looks complicated at first, but a quick read-through shows it just redirects stdin/stdout to the client socket.

```
int v9; // [rsp+2Ch] [rbp-1D4h]
struct termios termios_p; // [rsp+30h] [rbp-1D0h]
fd_set readfds; // [rsp+70h] [rbp-190h]
fd_set writefds; // [rsp+F0h] [rbp-110h]
char v13; // [rsp+170h] [rbp-90h]
__int64 v14; // [rsp+1F0h] [rbp-10h]
unsigned __int64 v15; // [rsp+1F8h] [rbp-8h]

v15 = __readfsqword(0x28u);
termios_p.c_lflag &= 0xFFFF7FB4;
termios_p.c_oflag &= 0xFFFFFFFA;
termios_p.c_iflag &= 0xFFFFFA84;
termios_p.c_cflag &= 0xFFFFFECF;
termios_p.c_cflag |= 0x30u;
tcsetattr(a2, 0, &termios_p);
do
{
  if ( dword_2062F8 )
    break;
  memset(&readfds, 0, sizeof(readfds));
  v4 = 0;
  v5 = (unsigned __int64)&writefds;
  memset(&writefds, 0, sizeof(writefds));
  v6 = 0;
  v7 = (unsigned __int64)&v13;
  memset(&v13, 0, 0x80uLL);
  v8 = 0;
  v9 = (unsigned __int64)&v14;
  readfds.fds_bits[a2 / 64] |= 1LL << (a2 % 64);
  readfds.fds_bits[*a1 / 64] |= 1LL << (*a1 % 64);
  select(a2 + 1, &readfds, &writefds, 0LL, 0LL);
  if ( readfds.fds_bits[a2 / 64] & (1LL << (a2 % 64)) )
  {
    if ( read(a2, &buf, 1uLL) == -1 )
      break;
    write(*a1, &buf, 1uLL);
  }
  if ( writefds.fds_bits[a2 / 64] & (1LL << (a2 % 64)) )
  {
    if ( read(*a1, &v2, 1uLL) != -1 )
      write(a2, &v2, 1uLL);
  }
}
while ( *a1 );
exit(121);
}
```

Restoring function names so far, we reach an io/read loop on the socket. First, its calling sub_441B, reading 512 bytes from the socket, and calling 468B with data read as its parameter, and passing its return value to 45BE.

```
*(_QWORD *)fd = a1;
v6 = a2;
v15 = __readfsqword(0x28u);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v12);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v13);
setSocketTimeout(a1, 180);
v2 = &amaster;
v8 = forkpty(&amaster, 0LL, 0LL, 0LL);
termDropPriv();
if ( v8 )
{
  if ( v8 < 0 )
    exit(122);
  sub_2D6C(fd, (unsigned int)amaster);
}
if ( fd[0] >= 0 )
{
  while ( fd[0] )
  {
    v9 = sub_441B(v2);
    bzero(&s, 0x200uLL);
    v9 = read(fd[0], &s, 0x200uLL);
    if ( v9 == -1 )
    {
      v4 = __cxa_allocate_exception(4uLL);
      *v4 = 1;
      __cxa_throw(v4, (struct type_info *)&`typeinfo for'int, 0LL);
    }
    v11 = sub_468B(&s);
    v2 = *(int **)fd;
    v10 = sub_45BE(*(_QWORD *)fd, v6, v11);
  }
  shutdown(0, 1);
  close(0);
  wait(0LL);
  wait(0LL);
  dword_2062F8 = 1;
  exit(1);
}
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v13);
```

441B calls 41FB and encrypts a string using openssl, sending it over to the stdout (and to fd because of ptyfork)

```
__int64 sub_441B()
{
  char v1; // [rsp+0h] [rbp-210h]
  unsigned __int64 v2; // [rsp+208h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  sub_41FB(&v1);
  echo_openssl((__int64)&v1);
  return 0LL;
}
```

41FB is just creating the interactive bash-like line, we can rename this function and its parents about what it's doing.

```
unsigned __int64 __fastcall sub_41FB(char *a1)
{
  const char *v1; // rax
  char *v3; // [rsp+10h] [rbp-1D0h]
  char *v4; // [rsp+10h] [rbp-1D0h]
  char *v5; // [rsp+18h] [rbp-1C8h]
  char v6; // [rsp+20h] [rbp-1C0h]
  char buf; // [rsp+40h] [rbp-1A0h]
  char name; // [rsp+C0h] [rbp-120h]
  unsigned __int64 v9; // [rsp+1C8h] [rbp-18h]

  v9 = __readfsqword(0x28u);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v6);
  v3 = getenv("SUDO_USER");
  gethostname(&name, 0xFEuLL);
  getcwd(&buf, 0x7FuLL);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::append(&v6, "\x1B[1;32m");
  if ( v3 )
  {
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::append(&v6, v3);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::append(&v6, "/:");
    v5 = getenv("USER");
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::append(&v6, v5);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::append(&v6, ":/");
  }
  else
  {
    v4 = getenv("USER");
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::append(&v6, v4);
  }
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::append(&v6, "@");
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::append(&v6, &name);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::append(&v6, "\x1B[37m:\x1B[94m");
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::append(&v6, &buf);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::append(&v6, "\x1B[37m$ ");
  v1 = (const char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(&v6);
  strncpy(a1, v1, 0x200uLL);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v6);
  return __readfsqword(0x28u) ^ v9;
}
```

468B is splitting string around whitespaces, and returning a char**, which has all the arguments.

```
_QWORD *__fastcall sub_468B(char *a1)
{
  int v2; // [rsp+18h] [rbp-18h]
  int v3; // [rsp+1Ch] [rbp-14h]
  _QWORD *ptr; // [rsp+20h] [rbp-10h]
  char *i; // [rsp+28h] [rbp-8h]

  v2 = 64;
  v3 = 0;
  ptr = malloc(0x200uLL);
  if ( !ptr )
  {
    fwrite("allocation error\n", 1uLL, 0x11uLL, stderr);
    exit(1);
  }
  for ( i = strtok(a1, " \t\r\n\a"); i; i = strtok(0LL, " \t\r\n\a") )
  {
    ptr[v3++] = i;
    if ( v3 >= v2 )
    {
      v2 += 64;
      ptr = realloc(ptr, 8LL * v2);
      if ( !ptr )
      {
        fwrite("allocation error\n", 1uLL, 0x11uLL, stderr);
        exit(1);
      }
    }
  }
  ptr[v3] = 0LL;
  return ptr;
}
```

With rest of the read io loop reversed, its easy to assume last call at 45BE handles the execvp execution of sub processes.

```
if ( sockfd[0] >= 0 )
{
  while ( sockfd[0] )
  {
    v8 = bashlike_cli_line();
    bzero(&s, 0x200uLL);
    v8 = read(sockfd[0], &s, 0x200uLL);
    if ( v8 == -1 )
    {
      v3 = __cxa_allocate_exception(4uLL);
      *v3 = 1;
      __cxa_throw(v3, (struct type_info *)&`typeinfo for'int, 0LL);
    }
    parsed_arguments = parseArguments(&s);
    v9 = sub_45BE(*(_QWORD *)sockfd, v5, parsed_arguments);
  }
```

This last function is running a small iteration, with strcmp, and if no matches were found, it returns process call and openssl encryption function as expected.

Loops comparing the first argument (so the process name) to a list at off_206060 and if a match is found, it returns an abstract call to the function pointer retrieved from funcs_466A[i] instead of creating an execvp.

```
5  v4 = a3;
7  if ( HIDWORD(a1) )
8    setuid(HIDWORD(a1));
9  if ( !*v4 )
0    return 1LL;
1  for ( i = 0; i < dword_206018; ++i )
2  {
3    if ( !strcmp(*v4, off_206060[i]) )
4      return ((__int64 (__fastcall *)(__int64, _QWORD, const char **))funcs_466A[i])(a1, a2, v4);
5  }
6  return proc_pipeto_openssl(a1, a2, (char **)v4);
7}
```

Subprocess call from the user input

```
.data:0000000000206060 off_206060      dq offset aCd           ; DATA XREF: sub_45BE+63↑o
.data:0000000000206060                                         ; "cd"
.data:0000000000206068                 dq offset s2            ; "cid"
.data:0000000000206070                 dq offset aSudo         ; "sudo"
.data:0000000000206078                 dq offset aExitBind     ; "exit_BIND"
.data:0000000000206080                 dq offset aAuthenticate ; "authenticate"
.data:0000000000206088                 dq offset aStm          ; "stm"
.data:0000000000206090                 dq offset aReadfileplain ; "readfileplain"
.data:0000000000206098                 dq offset _ZTIi         ; `typeinfo for'int
.data:0000000000206098 _data           ends
.data:0000000000206098
```

off_206060, list being compared to user input to call abstract function pointers.

```
.data:0000000000206020 funcs_466A      dq offset sub_240A      ; DATA XREF: sub_45BE+91↑o
.data:0000000000206020                                         ; sub_45BE+98↑r
.data:0000000000206020                 dq offset sub_251A
.data:0000000000206020                 dq offset sub_26D8
.data:0000000000206020                 dq offset sub_24FF
.data:0000000000206020                 dq offset sub_24C3
.data:0000000000206020                 dq offset sub_247D
.data:0000000000206020                 dq offset sub_25B5
.data:0000000000206058                 align 20h
```

funcs_466A, function pointer array

It looks like there are certain keywords mapped to functions which are builtin, and they are called instead of creating a subprocess.

```
1  __int64 __fastcall sub_236A(__int64 a1, __int64 a2, __int64 a3)
2  {
3    if ( *(_QWORD *)(a3 + 8) )
4    {
5      if ( chdir(*(const char **)(a3 + 8)) != 0 )
6        sub_2A12((__int64)"No such directory\n");
7      else
8        sub_2A12((__int64)"                    \r\n");
9    }
0    else
1    {
2      sub_2A12((__int64)"No arguement to \"cd\"\r\n");
3    }
4    return 1LL;
5  }
```

236A, builtin cd function mapped to “cd” keyword

We can check out these builtin functions at our session

```
ata@DESKTOP-5K61N76:/mnt/c/VM/Share/stmctf$ authenticate asdf
Thats sha512. Its 17 characters [!~]. Please don't waste your time bruteforcing. You can't.
ata@DESKTOP-5K61N76:/mnt/c/VM/Share/stmctf$ cid
connection id:1
Current uid:1000
socket fd:4
ata@DESKTOP-5K61N76:/mnt/c/VM/Share/stmctf$ stm
STMCTF{You need to elevate for flag}
ata@DESKTOP-5K61N76:/mnt/c/VM/Share/stmctf$ exit
ata@DESKTOP-5K61N76:/mnt/c/VM/Share/stmctf$ sudo ls
Not Allowed
ata@DESKTOP-5K61N76:/mnt/c/VM/Share/stmctf$
```

However, it looks like there are few attack surfaces which can yield setuid() and followed by a function call at sub_21FE

```
__int64 __fastcall sub_21FE(__int64 a1, unsigned int a2, __int64 a3)
{
  bool v3; // al
  __int64 v5; // [rsp+8h] [rbp-A8h]
  char v6; // [rsp+2Fh] [rbp-81h]
  char v7; // [rsp+30h] [rbp-80h]
  char v8; // [rsp+50h] [rbp-60h]
  char v9; // [rsp+70h] [rbp-40h]
  unsigned __int64 v10; // [rsp+98h] [rbp-18h]

  v5 = a3;
  v10 = __readfsqword(0x28u);
  std::allocator<char>::allocator(&v6);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v7, v5, &v6);
  std::allocator<char>::~allocator(&v6);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v9, &v7);
  sub_4061(&v8, &v9);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v9);
  v3 = !HIDWORD(a1)
    || !(unsigned int)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::compare(
                        &v8,
                        "8b586bf1b19e2d37191f780286c5ff74f7134ab650aebf7b6226d0d03b5d23acc0f443445315fb9f89ce2322fe1499d1"
                        "124ca22f0be158ad7a8bb9ea549bb1ad");
  if ( v3 )
  {
    echo_openssl((__int64)"Authenticated\n");
    setuid(0);
    sub_20FA((unsigned int)a1);
  }
  else
  {
    echo_openssl((__int64)"Thats sha512. Its 17 characters [!-~]. Please don't waste your time bruteforcing. You can't.\n");
  }
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v8);
  return std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v7);
}
```

This confirms we need remote to get the flag, and we need to elevate privileges for it. Since we cant patch remote server, we'll have to implement rsa decryption to execute commands on the server.

Since openssl rsautl is being used for encryption, we can use it to decrypt too.

```
p =
Popen(["openssl","rsautl","-decrypt","-inkey","cert/private.pem"
,"-in","test.txt"], stdout=PIPE, stdin=PIPE)
```

Then we need to do io read loops on the socket with 512 bytes for bash-like interactive line, and 512 bytes for the response.

Complete python client is also in the folder.

```python
#!/usr/bin/env python3

import socket
from subprocess import Popen, PIPE, STDOUT
import sys
HOST = sys.argv[1]
PORT = int(sys.argv[2])

def readnbytes(sock, n):
    buff = b''
    while n > 0:
        b = sock.recv(n)
        #print(b)
        buff += b
        if len(b) == 0:
            raise EOFError          # peer socket has received a SH_WR shutdown
        n -= len(b)
    return buff

def decrypt(data):
    f = open("test.txt","wb")
    f.write(data)
    f.flush()
    p = Popen(["openssl","rsautl","-decrypt","-inkey","cert/private.pem","-in","test.txt"]
    f.close()
    return ((p.communicate(input=data)[0]).decode())

print(f"Connecting to {HOST}:{PORT}")
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:


    s.connect((HOST, PORT))
    s.settimeout(5)
    while True:
        m = decrypt(readnbytes(s,512))
        print(m, end='')
        i = input("") + "\n"
        s.send((i.encode()))
        m = decrypt(readnbytes(s,512))
        print(m, end='')
```

Using this client, we can connect to the remote server.

We can confirm we need root permissions to get the flag, and that rookie is running as root with euid of our user.

```
Connecting to 192.168.56.137:1338
ignis/:root:/@vm:/home/kali/rookie$ ls
bin
build.sh
cert
flag.txt
include
Makefile
rootkit
src
ignis/:root:/@vm:/home/kali/rookie$ cat flag.txt
ignis/:root:/@vm:/home/kali/rookie$ ls -al flag.txt
-rw------- 1 root root 58 Jul 19 21:30 flag.txt
ignis/:root:/@vm:/home/kali/rookie$ id
uid=0(root) gid=1000(kali) euid=1000(ignis) groups=1000(kali)
ignis/:root:/@vm:/home/kali/rookie$
```

We can view the server source files directly to get a better understanding.

```
ignis/:root:/@vm:/home/kali/rookie$ ls -l src
total 32
drwxr-x--- 2 ignis kali 4096 Jul 19 21:30 auth
drwxr-x--- 2 ignis kali 4096 Jul 19 21:30 builtins
drwxr-x--- 2 ignis kali 4096 Jul 19 21:30 cipher
drwxr-x--- 2 ignis kali 4096 Jul 19 21:30 client
-rwxr-x--- 1 ignis kali 5763 Jul 19 23:08 server.cpp
drwxr-x--- 2 ignis kali 4096 Jul 19 21:30 sha512
drwxr-x--- 2 ignis kali 4096 Jul 19 21:30 shell
```

This gives a better understanding on how builtin abstract function calls work.

```
ignis/:root:/@vm:/home/kali/rookie$ cat src/builtins/builtins.h


#include <iostream>
#include <unistd.h>
#include <sys/socket.h>

#include "../auth/auth.h"

int builtin_cd(session s, char ** args);
int builtin_exit(session s, char ** args);
int builtin_flag(session s, char ** args);
int builtin_cid(session s, char ** args);
int builtin_sudo(session s, char ** args);

extern int (*builtin_func[])(session s, char **);

extern int numBuiltins;

extern const char *builtin_str[];ignis/:root:/@vm:/home/kali/rookie$
```

Rsautil limits us to 512 bytes per response, so we can use builtin readfileplain function to send us to contents of source files.

Reading through the source code, there is a backdoor setting uid to 0 in the sudo function

```
int builtin_sudo(session s, char ** args) {
    char* command = args[1];
    if(args[0]==0) return 1;


    if(s.uid==0 && !strcmp(args[1],"cid")){
        builtin_cid(s, &args[1]);
    }
    else if(s.uid==0 && !strcmp(args[1],"stm")){
        builtin_flag(s, &args[1]);
    }
    else if(s.uid==0 && !strcmp(args[1],"cd")){
        builtin_flag(s, &args[1]);
    }
    else if(s.uid=0 && !strcmp(args[1],"exit")){
        outputEncryptedString("Not Allowed\n");
    }
    else if(s.uid==0 && !strcmp(args[1],"sudo")){
        builtin_sudo(s, &args[1]);
    }else{
        outputEncryptedString("Not Allowed\n");
    }
}
```

This will return false and never execute the exit block, but we can still redirect execution to sudo (again).

Sudo sudo will set our s.uid to 0, giving us root permissions. We can read flag with `sudo sudo stm`

```
ignis/:root:/@vm:/home/kali/rookie$ cat flag.txt
ignis/:root:/@vm:/home/kali/rookie$ ls -l flag.txt
-rw------- 1 root root 58 Jul 19 21:30 flag.txt
ignis/:root:/@vm:/home/kali/rookie$ id
uid=0(root) gid=1000(kali) euid=1000(ignis) groups=1000(kali)
ignis/:root:/@vm:/home/kali/rookie$ stm
STMCTF{You need to elevate for flag}
ignis/:root:/@vm:/home/kali/rookie$ sudo stm
Not Allowed
ignis/:root:/@vm:/home/kali/rookie$ sudo sudo stm
STMCTF{dOub1E_sUdo_t0_b4ckd00r_0r_ju5t_a_r0okIe_M1st4ke?}
ignis/:root:/@vm:/home/kali/rookie$
```