WCTF 2018 binja Editorial

rswc

Author: @Charo_IT

Overview

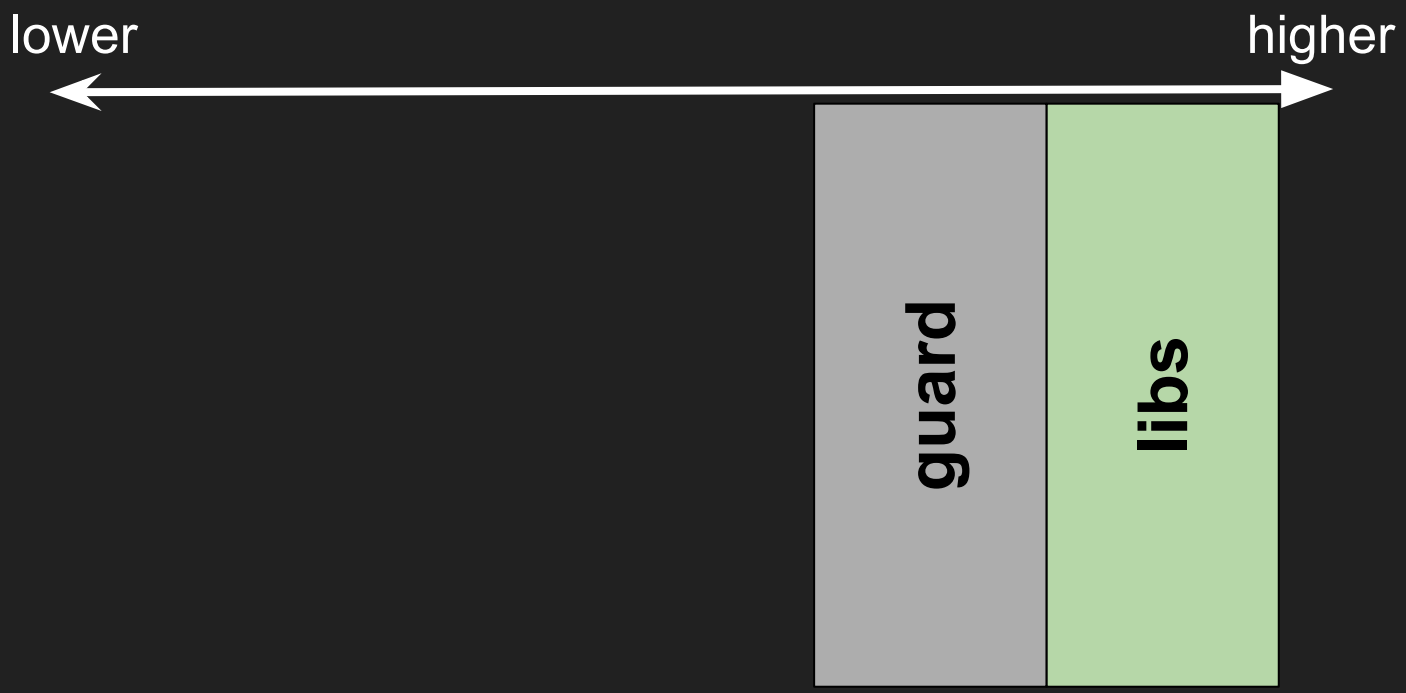
- A simple memo manager which uses original heap allocator
- You are dropped into an unprivileged shell
- Can you exploit the binary and gain your privilege?

About the original allocator

```
struct {
    void *heap_base;
    void *top;
    size_t heap_size;
    unsigned long num_chunks;
    chunkinfo chunks[(0x1000 - 0x20) / sizeof(chunkinfo)];
    // typedef struct {
    //     void *ptr;
    //     size_t size;
    // } chunkinfo;
} *arena;
```

Initializing the heap

```
1.  mmap(NULL, 0x1000, PROT_NONE, ...);
    // map guard page
```

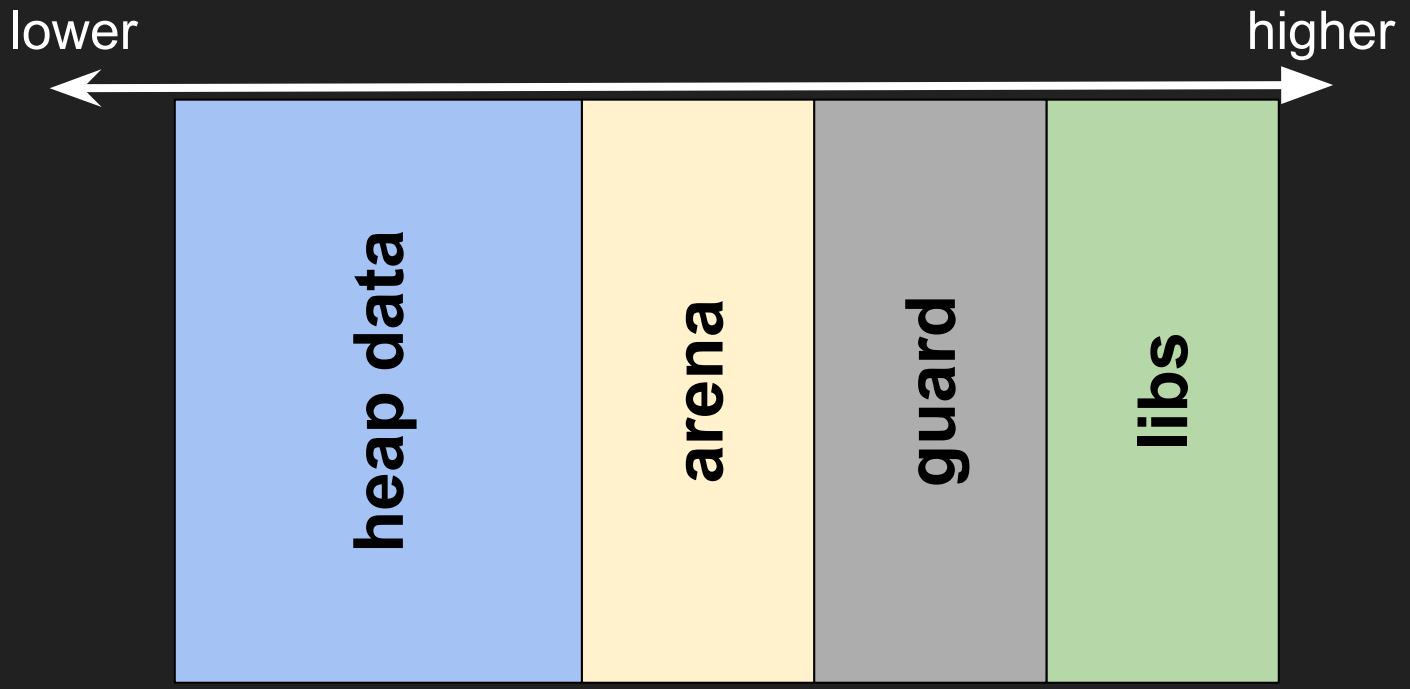


Initializing the heap

2.
```
arena =
    mmap(NULL, 0x1000, PROT_READ | PROT_WRITE, ...);
    // map area for arena
```

Initializing the heap

```
3.  arena->heap_base =
      mmap(NULL, heap_size, PROT_READ | PROT_WRITE,
           ...);  // map data area
```



On malloc

1. Round up requested size to multiple of 0x10
2. Scan `arena->chunks` and find a chunk which satisfies
   ```
   (chunk->size & 1) == 0  // chunk is not in use
   && aligned_requested_size <= arena->chunk[idx].size
   ```
3. If found, set LSB of `arena->chunks[idx].size` and return `arena->chunks[idx].ptr`
4. If not found, create a new chunk at `arena->top`
5. Add new chunk to `arena->chunks`
6. Return the address of new chunk

On free

1. Find target chunk from `arena->chunks`
2. Clear LSB of `arena->chunks[idx].size`

Vulnerability

On malloc:

1. Round up requested size to multiple of 0x10
2. Scan `arena->chunks` and find a chunk which satisfies
   ```
   (chunk->size & 1) == 0  // chunk is not in use
   && aligned_requested_size <= arena->chunk[idx].size
   ```
3. If found, set LSB of `arena->chunks[idx].size` and return `arena->chunks[idx].ptr`
4. If not found, create a new chunk at `arena->top`
5. Add new chunk to `arena->chunks`
6. Return the address of new chunk

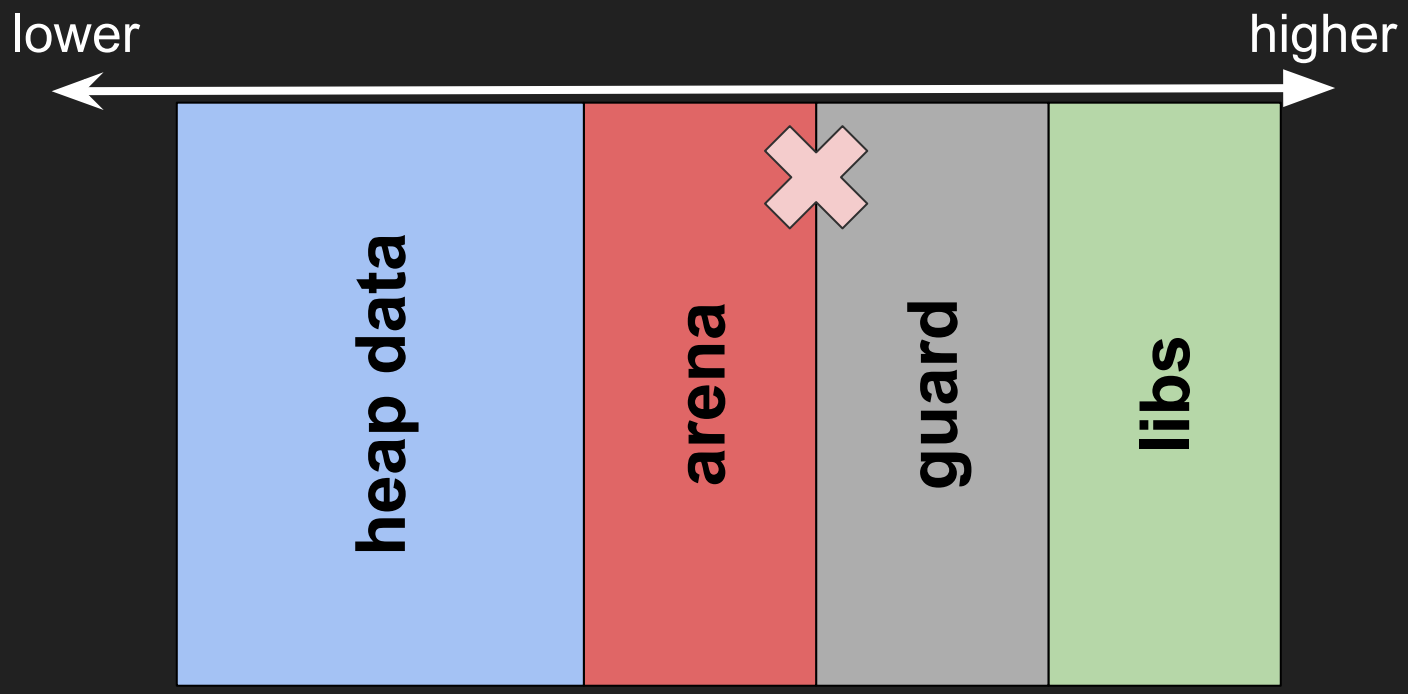
Vulnerability

```
    // no chunks available so use top
    if(arena->heap_base + arena->heap_size < arena->top + size){
        return NULL;
    }
    p = arena->top;
    arena->top += size;
    arena->chunks[arena->num_chunks].ptr = p;
    arena->chunks[arena->num_chunks].size = size | 1;
    arena->num_chunks++;

    return p;
}
```

allocating many chunks will make `arena->chunks` overflow

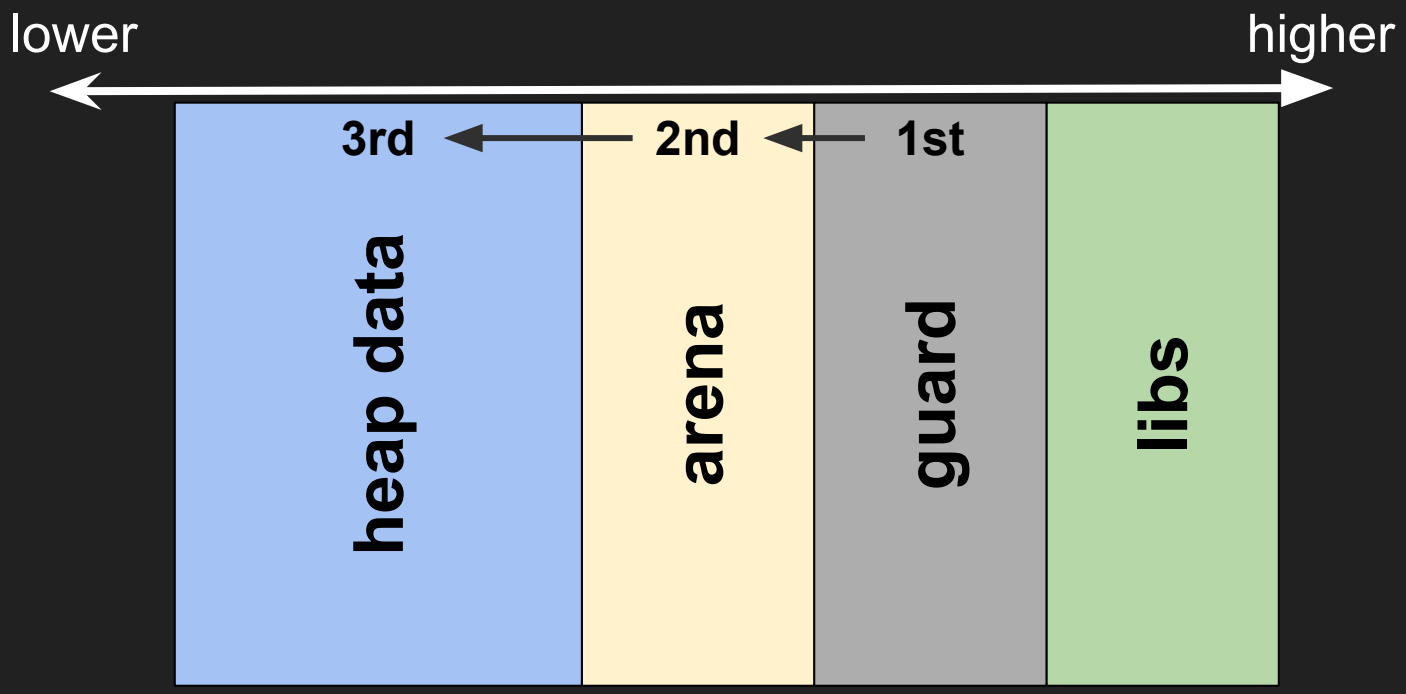
Is this exploitable?

`arena->chunks` will overflow but we can't overwrite anything because of guard page :(



But wait...

The initialization process relies on the assumption that mmap tries to allocate pages from higher to lower address.
Can we break it?



About mmap layout

```
void arch_pick_mmap_layout(struct mm_struct *mm)
{
    unsigned long random_factor = 0UL;

    if (current->flags & PF_RANDOMIZE)
        random_factor = arch_mmap_rnd();

    mm->mmap_legacy_base = TASK_UNMAPPED_BASE + random_factor;

    if (mmap_is_legacy()) {
        mm->mmap_base = mm->mmap_legacy_base;
        mm->get_unmapped_area = arch_get_unmapped_area;
    } else {
        mm->mmap_base = mmap_base(random_factor);
        mm->get_unmapped_area = arch_get_unmapped_area_topdown;
    }
}
```

About mmap layout

```
void arch_pick_mmap_layout(struct mm_struct *mm)
{
    unsigned
    if (cur
        ran
    mm->mmap                                    om_factor;

    if (mmap_is_legacy()) {
        mm->mmap_base = mm->mmap_legacy_base;
        mm->get_unmapped_area = arch_get_unmapped_area;
    } else {
        mm->mmap_base = mmap_base(random_factor);
        mm->get_unmapped_area = arch_get_unmapped_area_topdown;
    }
}
```

If some condition is met, use bottom-up (from lower to higher address) layout

About mmap layout

```
void arch_pick_mmap_layout(struct mm_struct *mm)
{
    unsigned long random_factor = 0UL;

    if (current->flags & PF_RANDOMIZE)
        rand

    mm->mmap                                    om_factor;

    if (mmap
        mm->
        mm->get_unmapped_area = arch_get_unmapped_area;
    } else {
        mm->mmap_base = mmap_base(random_factor);
        mm->get_unmapped_area = arch_get_unmapped_area_topdown;
    }
}
```

If not, use top-down (from higher to lower address) layout

About mmap layout

```
static int mmap_is_legacy(void)
{
    if (current->personality & ADDR_COMPAT_LAYOUT)
        return 1;

    if (rlimit(RLIMIT_STACK) == RLIM_INFINITY)
        return 1;

    return sysctl_legacy_va_layout;
}
```

Let's confirm it

```
$ ./test
[mmap 1] 0x7fc03cd9b000
[mmap 2] 0x7fc03cd9a000
[mmap 3] 0x7fc03cd99000
[mmap 4] 0x7fc03cd98000
[mmap 5] 0x7fc03cd97000
```

higher to lower

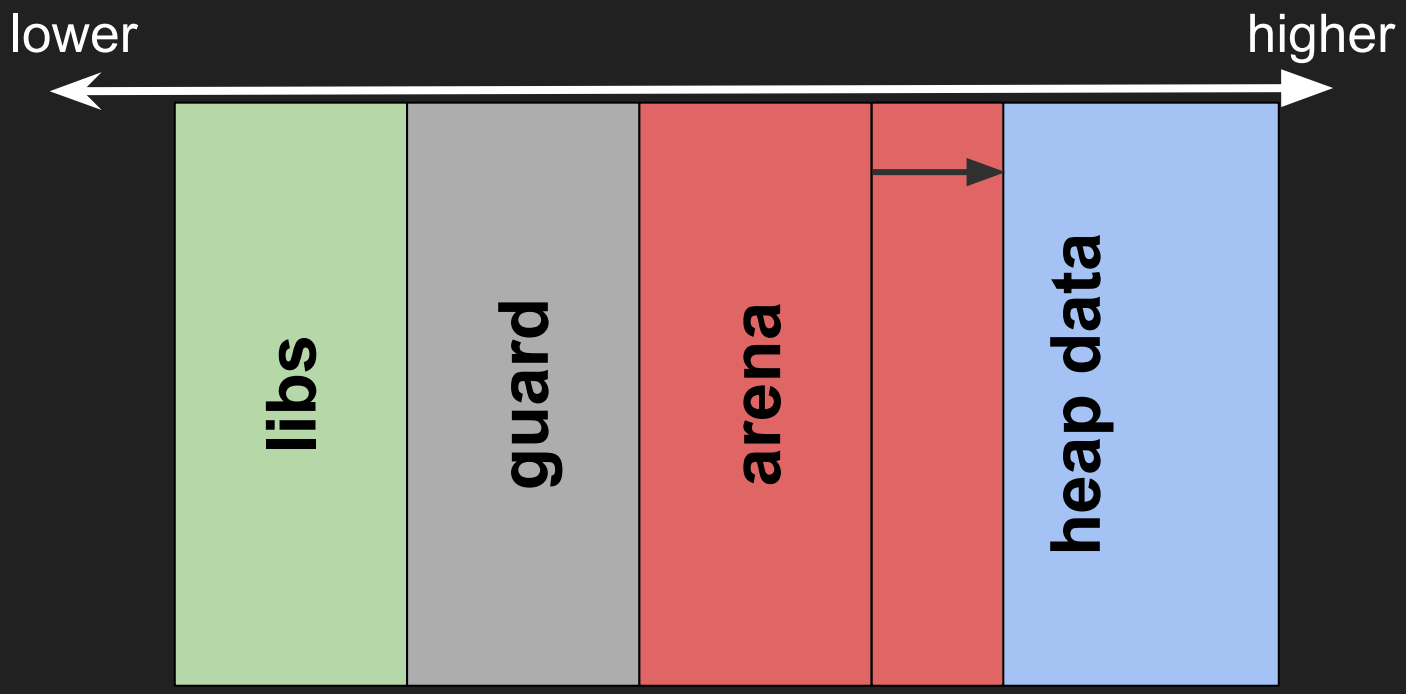
```
$ ulimit -s unlimited; ./test
[mmap 1] 0x2b0b265a1000
[mmap 2] 0x2b0b265a2000
[mmap 3] 0x2b0b265a3000
[mmap 4] 0x2b0b265a4000
[mmap 5] 0x2b0b265a5000
```

lower to higher

Note: This behavior has been removed in Linux 4.13
(The challenge was running on Linux 4.4)

Is this challenge exploitable?

We can change mmap layout to bottom-up style since we can do "`ulimit -s unlimited`".
So yes, it is exploitable!



Solution

1. `ulimit -s unlimited`
2. Launch the binary
3. Allocate many chunks to make `arena->chunks` overflow
4. Break link list of memo
5. GOT leak & GOT overwrite (eg. atoi -> gets)
6. Hijack the control flow and read the flag file

Truth

Author: @Charo_IT

Overview

- .NET reversing challenge which looks very easy at a first glance

Static analysis with dnSpy

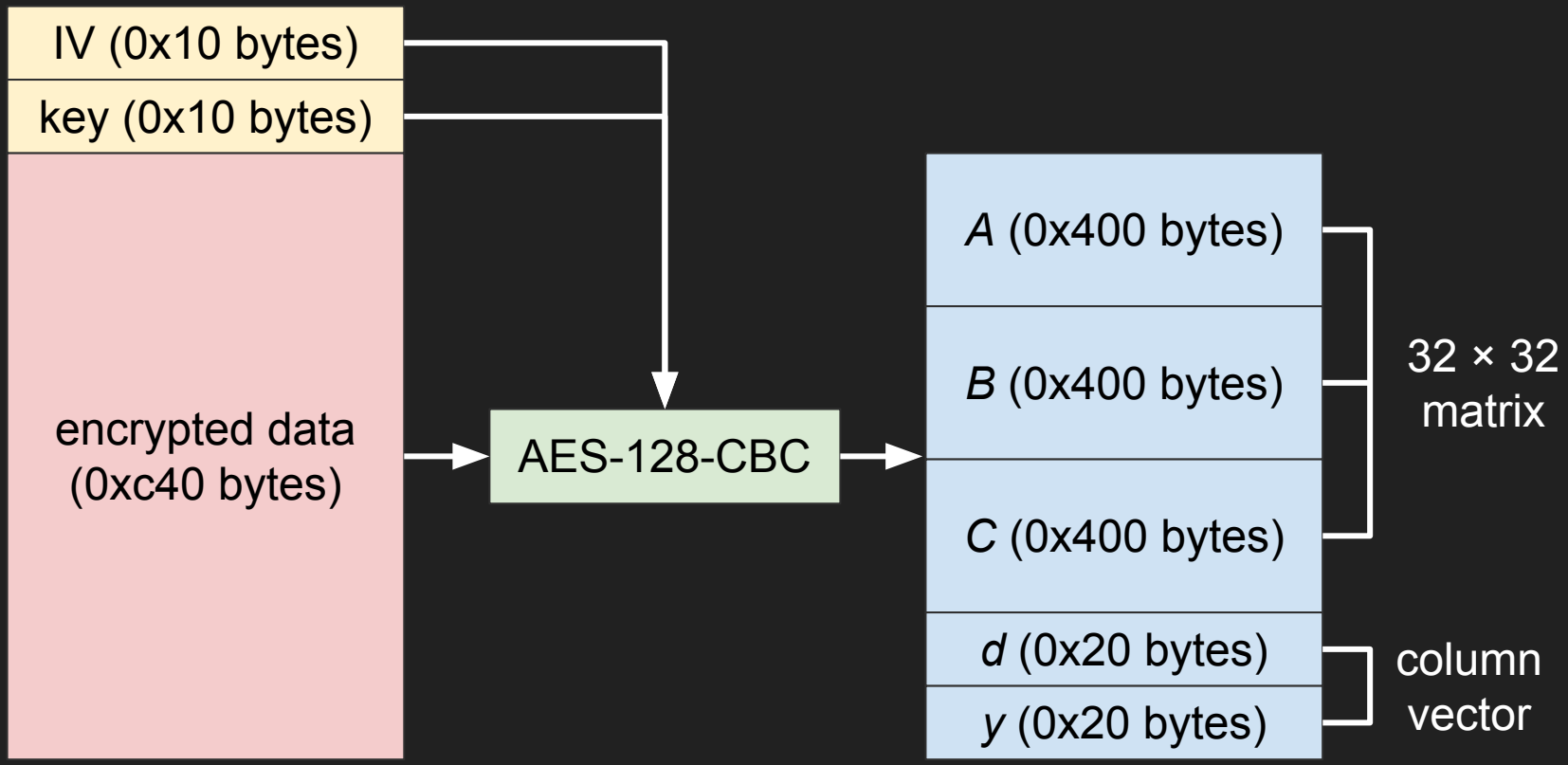
```
private static void Main() {
    Console.Write("Enter your flag: ");
    if (Lib.Verify(Console.ReadLine().Trim())) {
        Console.WriteLine("Great :)");
        return;
    }
    Console.WriteLine("Wrong :(");
}
```

Static analysis with dnSpy

```
public static bool Verify(string s) {
    byte[] bytes = Encoding.ASCII.GetBytes(s);
    if (bytes.Length != 32) {
        return false;
    }
    byte[] array = Lib.Func2();
    Lib.Func3(array, bytes);
    Lib.Func4(array, bytes);
    Lib.Func5(array, bytes);
    for (int i = 0; i < 32; i++) {
        if (bytes[i] != array[3104 + i]) {
            return false;
        }
    }
    return true;
}
```

Read and decrypt resource

Deconstructing resource



Static analysis with dnSpy

```
public static bool Verify(string s) {
    byte[] bytes = Encoding.ASCII.GetBytes(s);
    if (bytes.Length != 32) {
        return false;
    }
    byte[] array = Lib.Func2();
    Lib.Func3(array, bytes);
    Lib.Func4(array, bytes);
    Lib.Func5(array, bytes);
    for (int i = 0; i < 32; i++) {
        if (bytes[i] != array[3104 + i]) {
            return false;
        }
    }
    return true;
}
```

x_0 := input

Func3: $x_1 := Ax_0$

Func4: $x_2 := Bx_1$

Func5: $x_3 := Cx_2 + d$

is $x_3 == y$?

Pretty easy, eh?

According to static analysis, the input *x* should satisfy:

$$CBAx + d = y$$

So, we should be able to get the flag by:

$$flag = A^{-1} B^{-1} C^{-1}(y - d)$$

Pretty easy, eh?

According to static analysis, the input *x* should satisfy:

$$CBAx + d = y$$

So, we should be able to get the flag by:

$$flag = A^{-1} B^{-1} C^{-1}(y - d)$$

```
$ sage -python solver.py
```

Pretty easy, eh?

According to static analysis, the input *x* should satisfy:

$$C\,B\,Ax + d = y$$

So, we should be able to get the flag by:

$$flag = A^{-1}\,B^{-1}\,C^{-1}(y - d)$$

```
$ sage -python solver.py
flag: FAKEFLAGFAKEFLAGFAKEFLAGFAKEFLAG

$ challenge.exe
Enter your flag: FAKEFLAGFAKEFLAGFAKEFLAGFAKEFLAG
Wrong :(
```

The truth

```
unsafe static Resources()
{
    IntPtr intPtr = ldftn(Func) - 16;
    long num = *intPtr;
    IntPtr intPtr2 = ldftn(Func) - 8;
    long num2 = *intPtr2;
    ref long ptr = ldftn(Func) - 16;
    IntPtr intPtr3 = ldftn(Func) + 5;
    long num3 = (long)(*(intPtr3 + 1));
    ptr = *(intPtr3 + (IntPtr)(((int)(*(intPtr3 + 2)) << 3) + 3)) + (num3 << 3);
    ref long ptr2 = ldftn(Func) - 8;
    object obj = *(ldftn(Func) - 16);
    object obj2;
    for (;;)
    {
        obj2 = obj;
        if (*obj2 == 5)
        {
            break;
        }
        obj = obj2 + 16;
    }
    ptr2 = *(obj2 + 8);
    long num4 = *(ldftn(Func) - 8);
    *num4 = 6293447916875450697L;
    long num5 = num4 + 8L;
    *num5 = 996842507592L;
```

About MethodDesc

In .NET, each method has a structure called "MethodDesc" (Method Descriptor).

MethodDesc holds informations like:

- Lower bytes of MethodToken
- Has the method been already JITted?
- Is the method static?
- Method's entry point etc.

There are several types of MethodDesc, but this time we will only focus on the basic one which is used for regular IL methods.

About MethodDesc

```
0:003> !DumpMT -md 00007fff180e5b00
(snip)
MethodDesc Table
           Entry       MethodDesc    JIT Name
00007fff181f0090 00007fff180e5a98    NONE WCTF2018Rev.Lib.Verify(System.String)
00007fff181f0098 00007fff180e5aa8    NONE WCTF2018Rev.Lib.Func(Int32)
00007fff181f00a0 00007fff180e5ab8    NONE WCTF2018Rev.Lib.Func2()
00007fff181f00a8 00007fff180e5ac8    NONE WCTF2018Rev.Lib.Func3(Byte[], Byte[])
00007fff181f00b0 00007fff180e5ad8    NONE WCTF2018Rev.Lib.Func4(Byte[], Byte[])
00007fff181f00b8 00007fff180e5ae8    NONE WCTF2018Rev.Lib.Func5(Byte[], Byte[])

0:003> dq 00007fff180e5a98 l 6
00007fff`180e5a98  00280005`20000003 00007fff`181f0090 <- MethodDesc for Verify
00007fff`180e5aa8  00280006`20020004 00007fff`181f0098 <- MethodDesc for Func
00007fff`180e5ab8  00280007`20040005 00007fff`181f00a0 <- MethodDesc for Func2
```

MethodTokens etc. Entry point

Precode to JITted code

```
PrecodeForFunc1:
    call PrecodeFixupThunk
    pop rsi  ; dummy instruction
    db m_MethodDescChunkIndex_Func1
    db m_PrecodeChunkIndex_Func1
PrecodeForFunc2:
    call PrecodeFixupThunk
    pop rsi  ; dummy instruction
    db m_MethodDescChunkIndex_Func2
    db m_PrecodeChunkIndex_Func2

    dq MethodDescBase  ; pointer to the first element of
                         MethodDesc array
```

Precode to JITted code

```
PrecodeForFunc1:
 => call PrecodeFixupThunk
    pop rsi  ; dummy instruction
    db m_MethodDescChunkIndex_Func1
    db m_PrecodeChunkIndex_Func1
    dq MethodDescBase

PrecodeFixupThunk:
    pop rax
    movzx r10, byte ptr [rax+2]  ; m_PrecodeChunkIndex
    movzx r11, byte ptr [rax+1]  ; m_MethodDescChunkIndex
    mov rax, qword ptr [rax+r10*8+3]  ; MethodDescBase
    lea r10, [rax+r11*8]  ; r10=&MethodDesc[r11]
    jmp ThePreStub
```

Precode to JITted code

```
PrecodeForFunc1:
    call PrecodeFixupThunk
    pop rsi  ; dummy instruction     <= rax
    db m_MethodDescChunkIndex_Func1
    db m_PrecodeChunkIndex_Func1
    dq MethodDescBase

PrecodeFixupThunk:
 => pop rax
    movzx r10, byte ptr [rax+2]  ; m_PrecodeChunkIndex
    movzx r11, byte ptr [rax+1]  ; m_MethodDescChunkIndex
    mov rax, qword ptr [rax+r10*8+3]  ; MethodDescBase
    lea r10, [rax+r11*8]  ; r10=&MethodDesc[r11]
    jmp ThePreStub
```

Precode to JITted code

```
PrecodeForFunc1:
    call PrecodeFixupThunk
    pop rsi  ; dummy instruction     <= rax
    db m_MethodDescChunkIndex_Func1
    db m_PrecodeChunkIndex_Func1
    dq MethodDescBase

PrecodeFixupThunk:
    pop rax
 => movzx r10, byte ptr [rax+2]  ; m_PrecodeChunkIndex
    movzx r11, byte ptr [rax+1]  ; m_MethodDescChunkIndex
    mov rax, qword ptr [rax+r10*8+3]  ; MethodDescBase
    lea r10, [rax+r11*8]  ; r10=&MethodDesc[r11]
    jmp ThePreStub
```

Precode to JITted code

```
PrecodeForFunc1:
    call PrecodeFixupThunk
    pop rsi  ; dummy instruction
    db m_MethodDescChunkIndex_Func1
    db m_PrecodeChunkIndex_Func1
    dq MethodDescBase

PrecodeFixupThunk:
    pop rax
    movzx r10, byte ptr [rax+2]  ; m_PrecodeChunkIndex
    movzx r11, byte ptr [rax+1]  ; m_MethodDescChunkIndex
 => mov rax, qword ptr [rax+r10*8+3]  ; MethodDescBase
    lea r10, [rax+r11*8]  ; r10=&MethodDesc[r11]
    jmp ThePreStub
```

Precode to JITted code

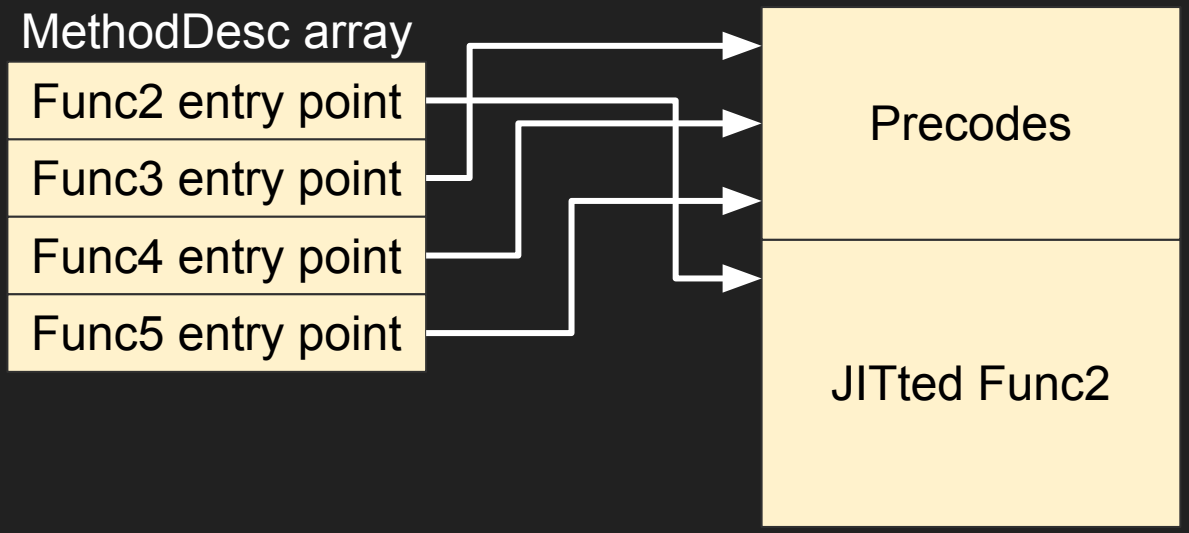
```
PrecodeForFunc1:
    call PrecodeFixupThunk
    pop rsi  ; dummy instruction
    db m_MethodDescChunkIndex_Func1
    db m_PrecodeChunkIndex_Func1
    dq MethodDescBase

PrecodeFixupThunk:
    pop rax
    movzx r10, byte ptr [rax+2]  ; m_PrecodeChunkIndex
    movzx r11, byte ptr [rax+1]  ; m_MethodDescChunkIndex
    mov rax, qword ptr [rax+r10*8+3]  ; MethodDescBase
 => lea r10, [rax+r11*8]  ; r10=&MethodDesc[r11]
    jmp ThePreStub
```

Precode to JITted code

```
ThePreStub:
    ; save registers to stack
    ; (snip)

    ; call PreStubWorker
    lea rcx, [rsp+0x68]
    mov rdx, r10  ; MethodDesc
    call PreStubWorker  ; do JIT compilation and update MethodDesc

    ; restore registers
    ; (snip)

    jmp rax  ; jump to compiled code
```

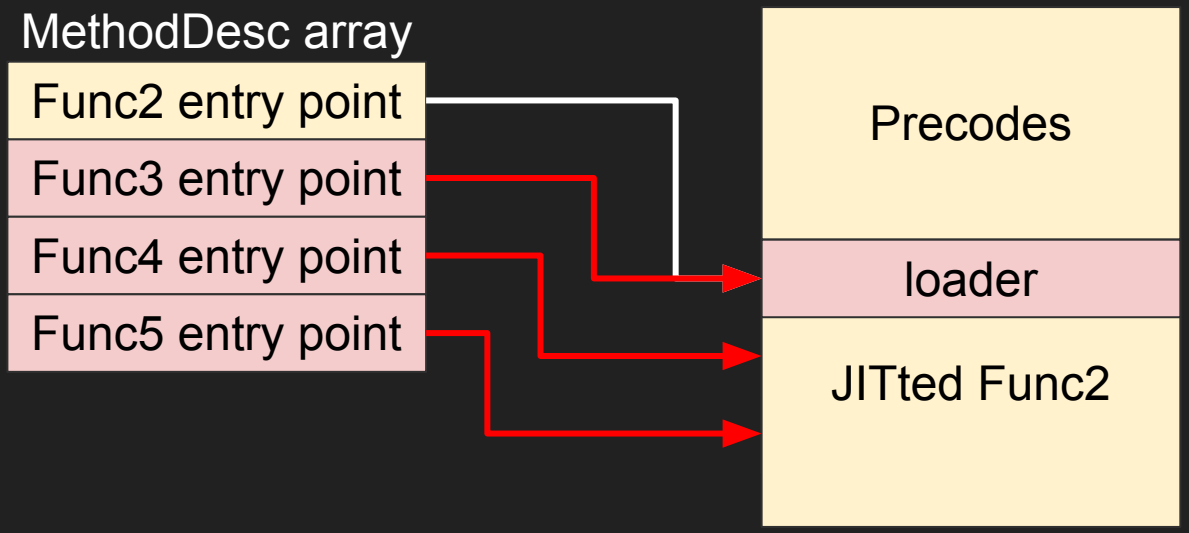
What Resources.cctor() does

1. Get the entry point of Func1 (which is not JITted yet) by using MSIL ldftn instruction
2. Simulate PrecodeFixupThunk and calculate the address of MethodDesc array
3. Overwrite the beginning of JITted Func2
4. Modify entry points for Func3, 4, and 5



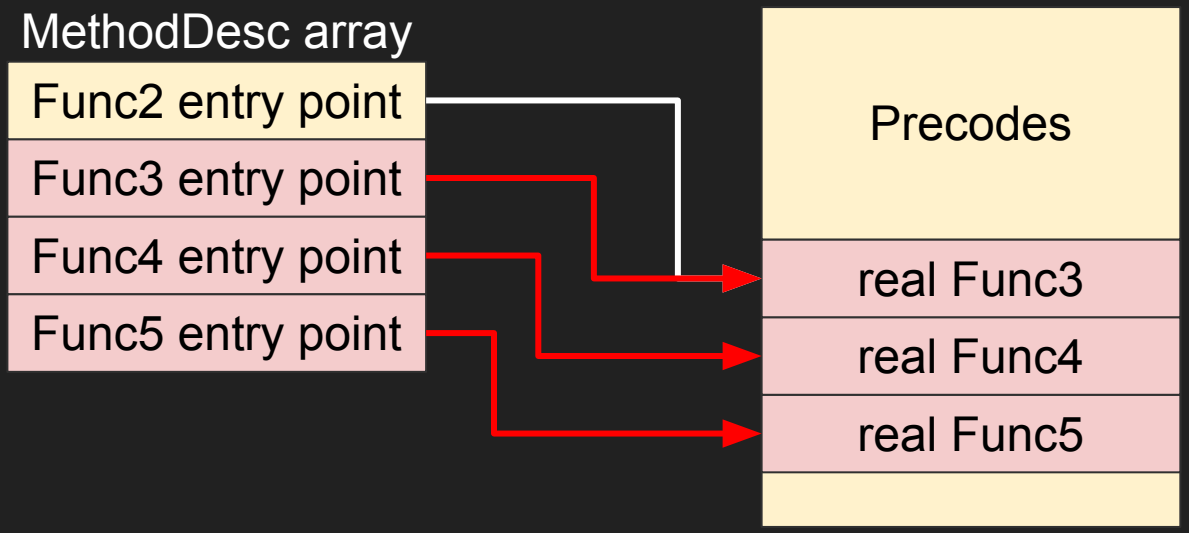
What Resources.cctor() does

1. Get the entry point of Func1 (which is not JITted yet) by using MSIL ldftn instruction
2. Simulate PrecodeFixupThunk and calculate the address of MethodDesc array
3. Overwrite the beginning of JITted Func2
4. Modify entry points for Func3, 4, and 5



The real behavior of Func3, 4, and 5

- Func3
  - Extract the real code of Func4 and Func5 from resource
  - $X_1 = AX_0$
- Func4: $X_2 = B_t^5 X_1$ ^ [0x5a, 0x5a, ..., 0x5a]
- Func5: 10.times{ $X_3 = C (X_2$ ^ [$B_{1,1}$, $B_{1,2}$, ..., $B_{1,32}$]) }



Fin.