Compact Control Flow Integerity in Linux

Introduction

The system supported stack plays an important role in running any application. If someone has the control of the stack then he/she has the control over the flow of the program. This is possible when some malicious input is given to the program and the attacker then redirects the control flow to a harmfull code segment. We intend to prevent this by modifying the binary therefore fortifying it.

Attack

The stack becomes vulnerable whenever a function is called either directly or indirectly.

(i) *Indirect Call :-* The control flow is vulnerable during both call and return.
(ii) *Direct Call:-* The control flow is vulnerable only during return.

Everytime when a program returns from a function, the return address is read from the stack. Whenever a function is called indirectly, the function address is read from the registers which is inturn read from the stack (happens in case of function pointers). The attacker can give some malicious inputs that overwrites the target adresses in the stack thus gaining power over the control flow.

How do we fortify?

This problem can be solved by clubing all the targets into a single place (which the attacker does not have access to) and redirecting the calls here. We call this address space **the springboard**.

We add a new section called the springboard and all the function calls are made from here and the original call instructions are replaced by the jmp instruction that branch to the corresponding springboard target.

Major steps in fortification

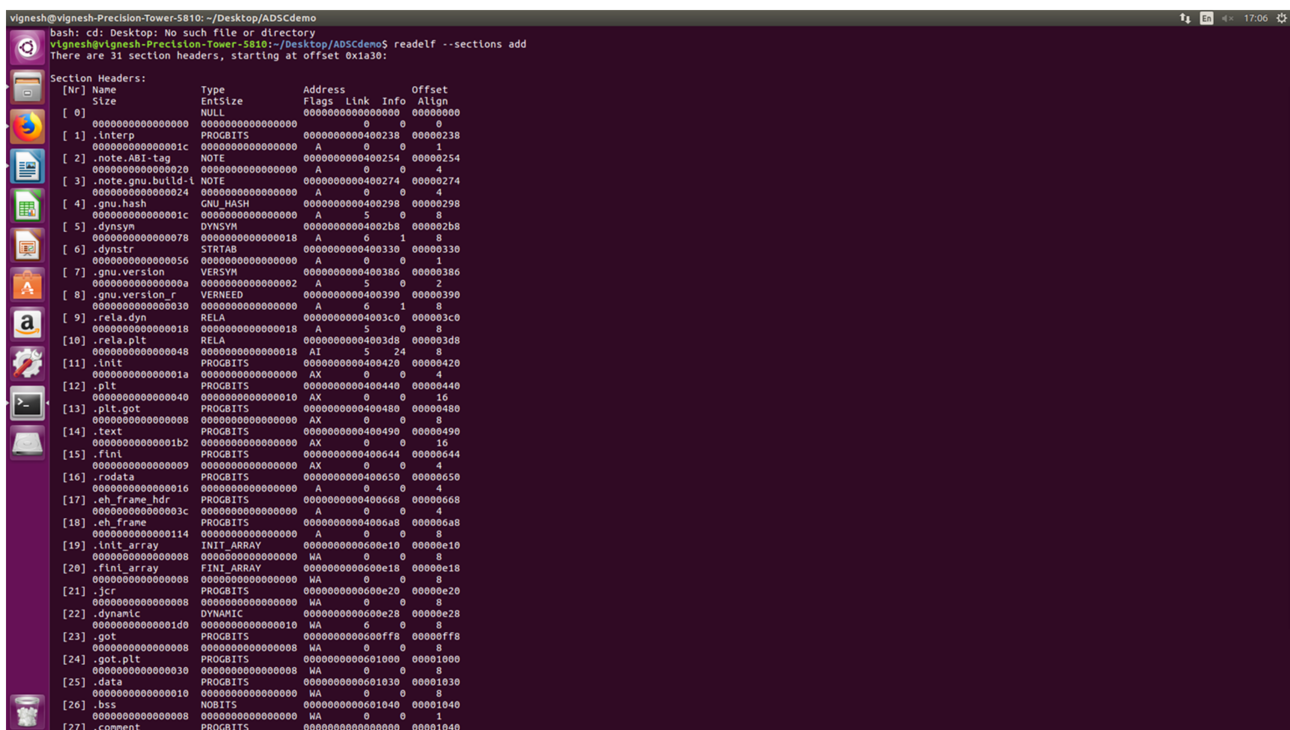
(i) Find all the calls and replace it with jumps that branches to the springboard along with adding call instuctions followed by jmp instructions that jumps back to the original calling address.

(ii) Insert checkpoints before every return statement that checks if the return address lies within the springboard section.

Technical details

Adding a Section to a binary:

We can use the *readelf --sections* command to print the sections of a binary.



To add a section use *objcopy --add-section .mysection=mydata inptfile outptfile* to add a new section with a name .mysection and has a data found in the file mydata. Objcopy, by default, adds the section after the comment section.

Below is the output of readelf –sections add1...where add1 is the output file.

```
vignesh@vignesh-Precision-Tower-5810: ~/Desktop/ADSCdemo
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS         0000000000400238  00000238
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.ABI-tag     NOTE             0000000000400254  00000254
       0000000000000020  0000000000000000   A       0     0     4
  [ 3] .note.gnu.build-i NOTE             0000000000400274  00000274
       0000000000000024  0000000000000000   A       0     0     4
  [ 4] .gnu.hash         GNU_HASH         0000000000400298  00000298
       000000000000001c  0000000000000000   A       5     0     8
  [ 5] .dynsym           DYNSYM           00000000004002b8  000002b8
       0000000000000078  0000000000000018   A       6     1     8
  [ 6] .dynstr           STRTAB           0000000000400330  00000330
       0000000000000056  0000000000000000   A       0     0     1
  [ 7] .gnu.version      VERSYM           0000000000400386  00000386
       000000000000000a  0000000000000002   A       5     0     2
  [ 8] .gnu.version_r    VERNEED          0000000000400390  00000390
       0000000000000030  0000000000000000   A       6     1     8
  [ 9] .rela.dyn         RELA             00000000004003c0  000003c0
       0000000000000018  0000000000000018   A       5     0     8
  [10] .rela.plt         RELA             00000000004003d8  000003d8
       0000000000000048  0000000000000018  AI       5    24     8
  [11] .init             PROGBITS         0000000000400420  00000420
       000000000000001a  0000000000000000  AX       0     0     4
  [12] .plt              PROGBITS         0000000000400440  00000440
       0000000000000040  0000000000000010  AX       0     0     16
  [13] .plt.got          PROGBITS         0000000000400480  00000480
       0000000000000008  0000000000000000  AX       0     0     8
  [14] .text             PROGBITS         0000000000400490  00000490
       00000000000001b2  0000000000000000  AX       0     0     16
  [15] .fini             PROGBITS         0000000000400644  00000644
       0000000000000009  0000000000000000  AX       0     0     4
  [16] .rodata           PROGBITS         0000000000400650  00000650
       0000000000000016  0000000000000000   A       0     0     4
  [17] .eh_frame_hdr     PROGBITS         0000000000400668  00000668
       000000000000003c  0000000000000000   A       0     0     4
  [18] .eh_frame         PROGBITS         00000000004006a8  000006a8
       0000000000000114  0000000000000000   A       0     0     8
  [19] .init_array       INIT_ARRAY       0000000000600e10  00000e10
       0000000000000008  0000000000000000  WA       0     0     8
  [20] .fini_array       FINI_ARRAY       0000000000600e18  00000e18
       0000000000000008  0000000000000000  WA       0     0     8
  [21] .jcr              PROGBITS         0000000000600e20  00000e20
       0000000000000008  0000000000000000  WA       0     0     8
  [22] .dynamic          DYNAMIC          0000000000600e28  00000e28
       00000000000001d0  0000000000000010  WA       6     0     8
  [23] .got              PROGBITS         0000000000600ff8  00000ff8
       0000000000000008  0000000000000008  WA       0     0     8
  [24] .got.plt          PROGBITS         0000000000601000  00001000
       0000000000000030  0000000000000008  WA       0     0     8
  [25] .data             PROGBITS         0000000000601030  00001030
       0000000000000010  0000000000000000  WA       0     0     8
  [26] .bss              NOBITS           0000000000601040  00001040
       0000000000000008  0000000000000000  WA       0     0     1
  [27] .comment          PROGBITS         0000000000000000  00001040
       0000000000000035  0000000000000001  MS       0     0     1
  [28] .mysection        PROGBITS         0000000000000000  00001075
       0000000000000111  0000000000000000           0     0     1
  [29] .shstrtab         STRTAB           0000000000000000  00001a48
       0000000000000117  0000000000000000           0     0     1
  [30] .symtab           SYMTAB           0000000000000000  00001188
       0000000000000690  0000000000000018          31    48     8
```

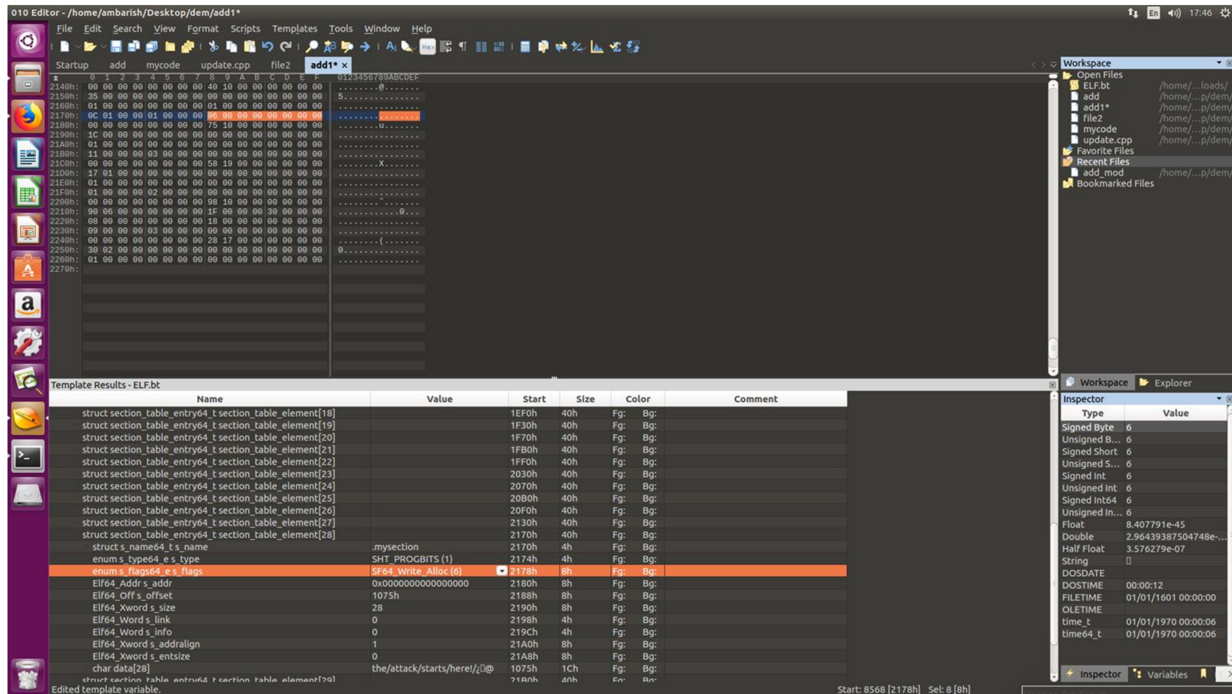
Though we have added a new section, it is not yet executable. If we try to redirect the control flow there we will recieve a segmentation fault. This is because the new section is not yet in the loadable segment. So our section will not be loaded during the runtime.
We can check the loadable segments in the program header with *readelf -l* command.

```
vignesh@vignesh-Precision-Tower-5810: ~/Desktop/ADSCdemo
       0000000000000008  0000000000000008  WA       0     0     8
  [24] .got.plt          PROGBITS         0000000000601000  00001000
       0000000000000030  0000000000000008  WA       0     0     8
  [25] .data             PROGBITS         0000000000601030  00001030
       0000000000000010  0000000000000000  WA       0     0     8
  [26] .bss              NOBITS           0000000000601040  00001040
       0000000000000008  0000000000000000  WA       0     0     1
  [27] .comment          PROGBITS         0000000000000000  00001040
       0000000000000035  0000000000000001  MS       0     0     1
  [28] .mysection        PROGBITS         0000000000000000  00001075
       0000000000000111  0000000000000000           0     0     1
  [29] .shstrtab         STRTAB           0000000000000000  00001a48
       0000000000000117  0000000000000000           0     0     1
  [30] .symtab           SYMTAB           0000000000000000  00001188
       0000000000000690  0000000000000018          31    48     8
  [31] .strtab           STRTAB           0000000000000000  00001818
       0000000000000230  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), l (large)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
vignesh@vignesh-Precision-Tower-5810:~/Desktop/ADSCdemo$ readelf -l add1

Elf file type is EXEC (Executable file)
Entry point 0x400490
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x00000000000007bc 0x00000000000007bc  R E    200000
  LOAD           0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x0000000000000230 0x0000000000000238  RW     200000
  DYNAMIC        0x0000000000000e28 0x0000000000600e28 0x0000000000600e28
                 0x00000000000001d0 0x00000000000001d0  RW     8
  NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x0000000000000668 0x0000000000400668 0x0000000000400668
                 0x000000000000003c 0x000000000000003c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x00000000000001f0 0x00000000000001f0  R      1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .plt.got .text .fini .rodata .eh_frame_hdr .eh_frame
   03     .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .jcr .dynamic .got
vignesh@vignesh-Precision-Tower-5810:~/Desktop/ADSCdemo$
```

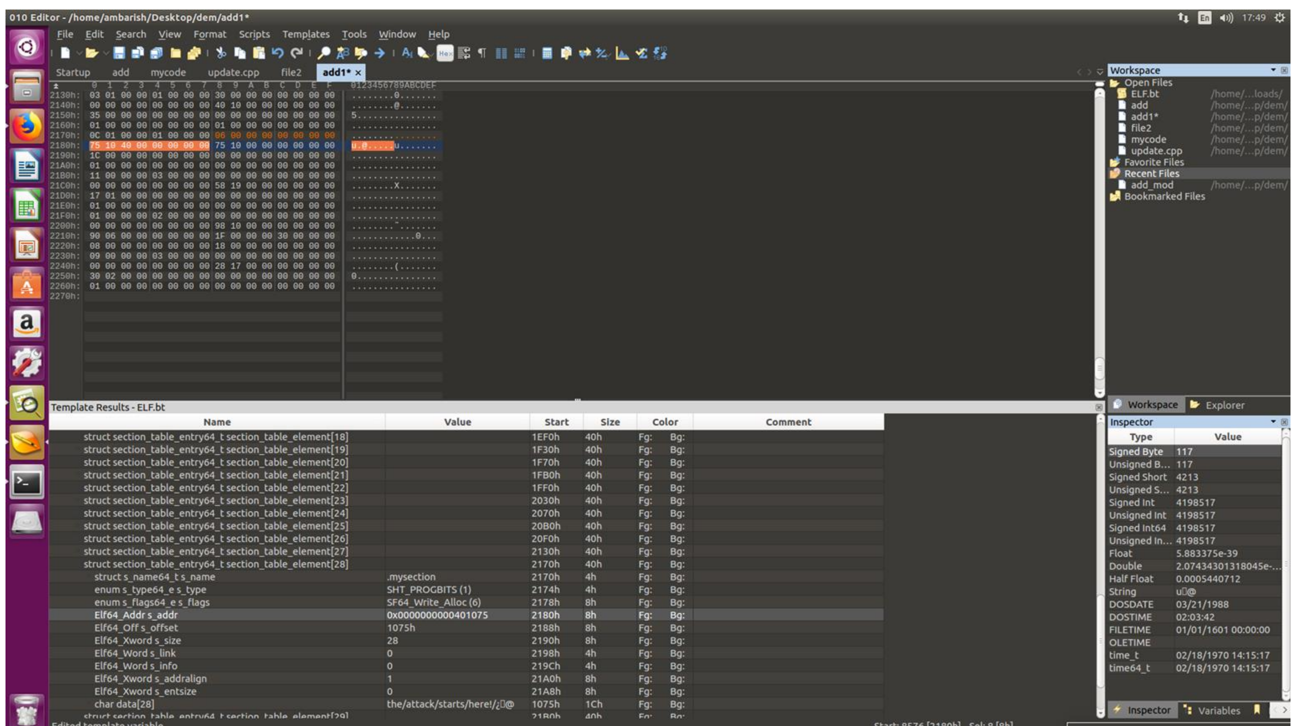
We can see that our section is not there in the loadable segment(no. 3).

To make our section executable, we have to make some mannual changes in the binary. One can use a hex editor like *010Editor* to do these changes.\

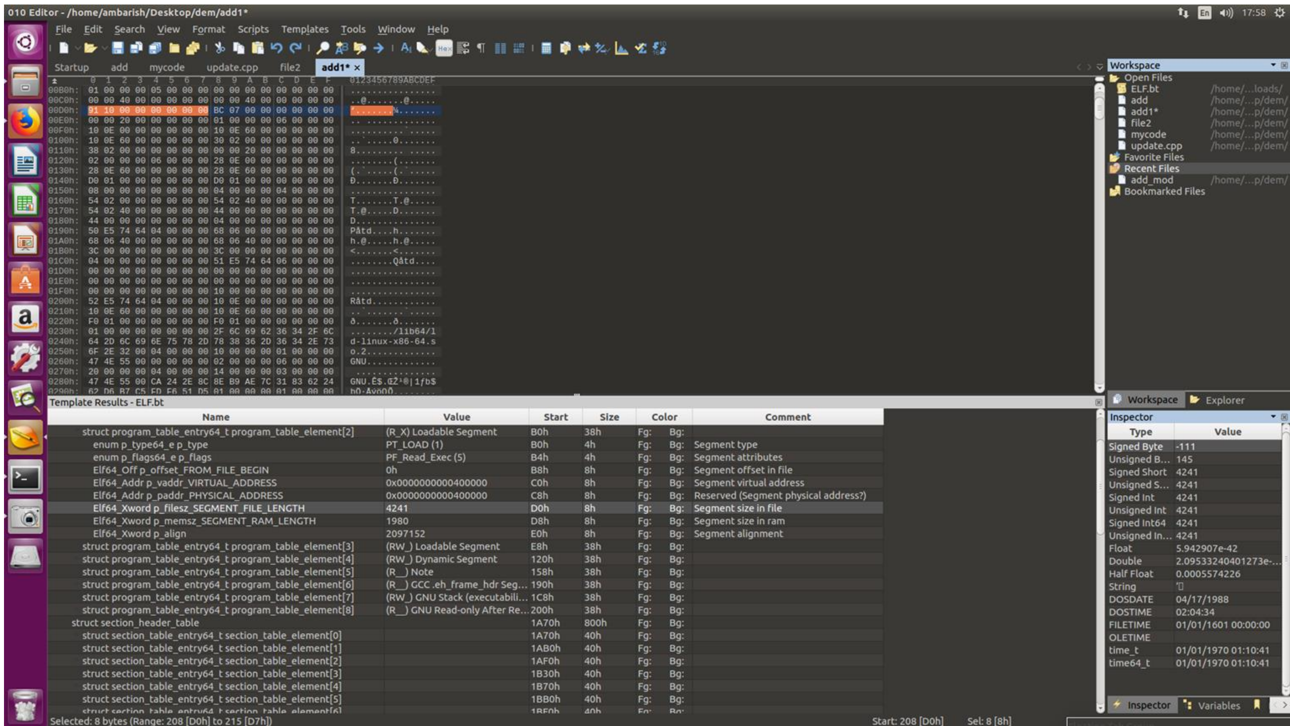
❿ Change the flag of the section in the section header table to **6**.



❿ Make sure the address of the section = base address + offset, the base is same as that of other loadable sections. In this case base = 0x400000, offset = 0x1075. The base address can be obtained by looking at the other loadable sections and using the above formula.



➓ Increase the size of the loadable segment (3rd segment, 02) in the program header table to include our new section. Change both the file length and ram length. The size has to be one greater than the address of the last byte of our section. To get the value for Elf64_xword p_filesz_segment_file_length and Elf64_xword p_memsz_segment_ram_length (they have to be set the same): Approach 1, to look at the size and offset of our added section (.mysection), add them together (and plus 1) to get 0x1091. Approach 2 is to look at the last byte (plus 1) of chardata field of the section.



And now our section is successfully in the loadable segment. After this the section is executable.

```
ambarish@ambarish-Precision-Tower-5810: ~/Desktop/dem
                 0x00000000000001d0 0x00000000000001d0  RW     8
  NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x0000000000000668 0x0000000000400668 0x0000000000400668
                 0x000000000000003c 0x000000000000003c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x00000000000001f0 0x00000000000001f0  R      1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .plt.got .text .fini .rodata .eh_frame_hdr .eh_frame
   03     .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .jcr .dynamic .got
ambarish@ambarish-Precision-Tower-5810:~/Desktop/dem$ readelf -l add1

Elf file type is EXEC (Executable file)
Entry point 0x400490
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x0000000000001091 0x0000000000001091  R E    200000
  LOAD           0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x0000000000000230 0x0000000000000238  RW     200000
  DYNAMIC        0x0000000000000e28 0x0000000000600e28 0x0000000000600e28
                 0x00000000000001d0 0x00000000000001d0  RW     8
  NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x0000000000000668 0x0000000000400668 0x0000000000400668
                 0x000000000000003c 0x000000000000003c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x00000000000001f0 0x00000000000001f0  R      1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .plt.got .text .fini .rodata .eh_frame_hdr .eh_frame .mysection
   03     .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .jcr .dynamic .got
ambarish@ambarish-Precision-Tower-5810:~/Desktop/dem$
```

Now we can add our executable instructions and direct the control flow here.

Direct Call

- 1st add a section called *.springboard.*
- As mentioned earlier redirect the calls to the spring board.
- To add the checkpoints, we need to insert a piece of code before the return statement but that would change the offsets of many other instructions which is not desirable.
- So we add a new section called *.extention* in which we add the extra piece of code with a return at the end.

```
vignesh@vignesh-Precision-Tower-5810: ~/Desktop/ADSCdemo
  400604:       74 20                   je     400626 <__libc_csu_init+0x56>
  400606:       31 db                   xor    %ebx,%ebx
  400608:       0f 1f 84 00 00 00 00    nopl   0x0(%rax,%rax,1)
  40060f:       00
  400610:       4c 89 ea                mov    %r13,%rdx
  400613:       4c 89 f6                mov    %r14,%rsi
  400616:       44 89 ff                mov    %r15d,%edi
  400619:       41 ff 14 dc             callq  *(%r12,%rbx,8)
  40061d:       48 83 c3 01             add    $0x1,%rbx
  400621:       48 39 eb                cmp    %rbp,%rbx
  400624:       75 ea                   jne    400610 <__libc_csu_init+0x40>
  400626:       48 83 c4 08             add    $0x8,%rsp
  40062a:       5b                      pop    %rbx
  40062b:       5d                      pop    %rbp
  40062c:       41 5c                   pop    %r12
  40062e:       41 5d                   pop    %r13
  400630:       41 5e                   pop    %r14
  400632:       41 5f                   pop    %r15
  400634:       c3                      retq
  400635:       90                      nop
  400636:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
  40063d:       00 00 00

0000000000400640 <__libc_csu_fini>:
  400640:       f3 c3                   repz retq

Disassembly of section .fini:

0000000000400644 <_fini>:
  400644:       48 83 ec 08             sub    $0x8,%rsp
  400648:       48 83 c4 08             add    $0x8,%rsp
  40064c:       c3                      retq

Disassembly of section .extension:

0000000000401075 <.extension>:
  401075:       e8 f6 f3 ff ff          callq  400470 <__isoc99_scanf@plt>
  40107a:       90                      nop
  40107b:       c9                      leaveq
  40107c:       50                      push   %rax
  40107d:       8d 05 12 00 00 00       lea    0x12(%rip),%eax        # 401095 <__FRAME_END__+0x8dd>
  401083:       2b 44 24 08             sub    0x8(%rsp),%eax
  401087:       83 f8 00                cmp    $0x0,%eax
  40108a:       7e 05                   jle    401091 <__FRAME_END__+0x8d9>
  40108c:       e9 1f f7 ff ff          jmpq   4007b0 <__GNU_EH_FRAME_HDR+0x148>
  401091:       58                      pop    %rax
  401092:       c3                      retq
  401093:       90                      nop
  401094:       90                      nop

Disassembly of section .springboard:

0000000000401095 <.springboard>:
  401095:       e8 ec f4 ff ff          callq  400586 <scn>
  40109a:       e9 16 f5 ff ff          jmpq   4005b5 <main+0xe>
  40109f:       90                      nop
  4010a0:       90                      nop
  4010a1:       90                      nop
  4010a2:       90                      nop
  4010a3:       90                      nop
  4010a4:       90                      nop
vignesh@vignesh-Precision-Tower-5810:~/Desktop/ADSCdemo$
```

⓾ We also have to redirect the end of the function to this section.

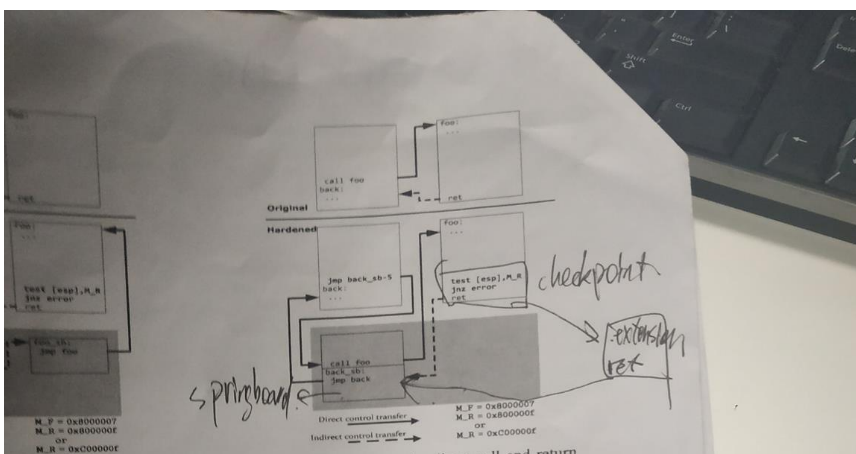
```
ambarish@ambarish-Precision-Tower-5810: ~/Desktop/dem
  40053a:       66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)

0000000000400540 <__do_global_dtors_aux>:
  400540:       80 3d f9 0a 20 00 00    cmpb   $0x0,0x200af9(%rip)        # 601040 <__TMC_END__>
  400547:       75 11                   jne    40055a <__do_global_dtors_aux+0x1a>
  400549:       55                      push   %rbp
  40054a:       48 89 e5                mov    %rsp,%rbp
  40054d:       e8 6e ff ff ff          callq  4004c0 <deregister_tm_clones>
  400552:       5d                      pop    %rbp
  400553:       c6 05 e6 0a 20 00 01    movb   $0x1,0x200ae6(%rip)        # 601040 <__TMC_END__>
  40055a:       f3 c3                   repz retq
  40055c:       0f 1f 40 00             nopl   0x0(%rax)

0000000000400560 <frame_dummy>:
  400560:       bf 20 0e 60 00          mov    $0x600e20,%edi
  400565:       48 83 3f 00             cmpq   $0x0,(%rdi)
  400569:       75 05                   jne    400570 <frame_dummy+0x10>
  40056b:       eb 93                   jmp    400500 <register_tm_clones>
  40056d:       0f 1f 00                nopl   (%rax)
  400570:       b8 00 00 00 00          mov    $0x0,%eax
  400575:       48 85 c0                test   %rax,%rax
  400578:       74 f1                   je     40056b <frame_dummy+0xb>
  40057a:       55                      push   %rbp
  40057b:       48 89 e5                mov    %rsp,%rbp
  40057e:       ff d0                   callq  *%rax
  400580:       5d                      pop    %rbp
  400581:       e9 7a ff ff ff          jmpq   400500 <register_tm_clones>

0000000000400586 <scn>:
  400586:       55                      push   %rbp
  400587:       48 89 e5                mov    %rsp,%rbp
  40058a:       48 83 ec 10             sub    $0x10,%rsp
  40058e:       48 8d 45 f0             lea    -0x10(%rbp),%rax
  400592:       48 89 c6                mov    %rax,%rsi
  400595:       bf 54 06 40 00          mov    $0x400654,%edi
  40059a:       b8 00 00 00 00          mov    $0x0,%eax
  40059f:       e8 cc fe ff ff          callq  400470 <__isoc99_scanf@plt>
  4005a4:       90                      nop
  4005a5:       c9                      leaveq
  4005a6:       c3                      retq

00000000004005a7 <main>:
  4005a7:       55                      push   %rbp
  4005a8:       48 89 e5                mov    %rsp,%rbp
  4005ab:       b8 00 00 00 00          mov    $0x0,%eax
  4005b0:       e8 d1 ff ff ff          callq  400586 <scn>
  4005b5:       bf 57 06 40 00          mov    $0x400657,%edi
  4005ba:       e8 91 fe ff ff          callq  400450 <puts@plt>
  4005bf:       bf 61 06 40 00          mov    $0x400661,%edi
  4005c4:       e8 87 fe ff ff          callq  400450 <puts@plt>
  4005c9:       b8 00 00 00 00          mov    $0x0,%eax
  4005ce:       5d                      pop    %rbp
  4005cf:       c3                      retq

00000000004005d0 <__libc_csu_init>:
  4005d0:       41 57                   push   %r15
  4005d2:       41 56                   push   %r14
  4005d4:       41 89 ff                mov    %edi,%r15d
  4005d7:       41 55                   push   %r13
  4005d9:       41 54                   push   %r12
  4005db:       4c 8d 25 2e 08 20 00    lea    0x20082e(%rip),%r12        # 600e10 <__frame_dummy_init_array_entry>
  4005e2:       55                      push   %rbp
```

-> this is before the modification

```
vignesh@vignesh-Precision-Tower-5810: ~/Desktop/ADSCdemo
  400565:       48 83 3f 00             cmpq   $0x0,(%rdi)
  400569:       75 05                   jne    400570 <frame_dummy+0x10>
  40056b:       eb 93                   jmp    400500 <register_tm_clones>
  40056d:       0f 1f 00                nopl   (%rax)
  400570:       b8 00 00 00 00          mov    $0x0,%eax
  400575:       48 85 c0                test   %rax,%rax
  400578:       74 f1                   je     40056b <frame_dummy+0xb>
  40057a:       55                      push   %rbp
  40057b:       48 89 e5                mov    %rsp,%rbp
  40057e:       ff d0                   callq  *%rax
  400580:       5d                      pop    %rbp
  400581:       e9 7a ff ff ff          jmpq   400500 <register_tm_clones>

0000000000400586 <scn>:
  400586:       55                      push   %rbp
  400587:       48 89 e5                mov    %rsp,%rbp
  40058a:       48 83 ec 10             sub    $0x10,%rsp
  40058e:       48 8d 45 f0             lea    -0x10(%rbp),%rax
  400592:       48 89 c6                mov    %rax,%rsi
  400595:       bf 54 06 40 00          mov    $0x400654,%edi
  40059a:       b8 00 00 00 00          mov    $0x0,%eax
  40059f:       e9 d1 0a 00 00          jmpq   401075 <__FRAME_END__+0x8bd>
  4005a4:       90                      nop
  4005a5:       90                      nop
  4005a6:       90                      nop

00000000004005a7 <main>:
  4005a7:       55                      push   %rbp
  4005a8:       48 89 e5                mov    %rsp,%rbp
  4005ab:       b8 00 00 00 00          mov    $0x0,%eax
  4005b0:       e9 e0 0a 00 00          jmpq   401095 <__FRAME_END__+0x8dd>
  4005b5:       bf 57 06 40 00          mov    $0x400657,%edi
  4005ba:       e8 91 fe ff ff          callq  400450 <puts@plt>
  4005bf:       bf 61 06 40 00          mov    $0x400661,%edi
  4005c4:       e8 87 fe ff ff          callq  400450 <puts@plt>
  4005c9:       b8 00 00 00 00          mov    $0x0,%eax
  4005ce:       5d                      pop    %rbp
  4005cf:       c3                      retq

00000000004005d0 <__libc_csu_init>:
  4005d0:       41 57                   push   %r15
  4005d2:       41 56                   push   %r14
  4005d4:       41 89 ff                mov    %edi,%r15d
  4005d7:       41 55                   push   %r13
  4005d9:       41 54                   push   %r12
  4005db:       4c 8d 25 2e 08 20 00    lea    0x20082e(%rip),%r12        # 600e10 <__frame_dummy_init_array_entry>
  4005e2:       55                      push   %rbp
  4005e3:       48 8d 2d 2e 08 20 00    lea    0x20082e(%rip),%rbp        # 600e18 <__init_array_end>
  4005ea:       53                      push   %rbx
  4005eb:       49 89 f6                mov    %rsi,%r14
  4005ee:       49 89 d5                mov    %rdx,%r13
  4005f1:       4c 29 e5                sub    %r12,%rbp
  4005f4:       48 83 ec 08             sub    $0x8,%rsp
  4005f8:       48 c1 fd 03             sar    $0x3,%rbp
  4005fc:       e8 1f fe ff ff          callq  400420 <_init>
  400601:       48 85 ed                test   %rbp,%rbp
  400604:       74 20                   je     400626 <__libc_csu_init+0x56>
  400606:       31 db                   xor    %ebx,%ebx
  400608:       0f 1f 84 00 00 00 00    nopl   0x0(%rax,%rax,1)
  40060f:       00
  400610:       4c 89 ea                mov    %r13,%rdx
  400613:       4c 89 f6                mov    %r14,%rsi
```

-> this is after modification

we delete the required number of instructions to add a jmp instruction and replace the remaining bytes with nop(0x90). So we copy the deleted code in the .extension before we add the checkpoint. The program control flow for the modified program is shown as the figure below.



indirect call and return

Figure 7: Rewriting of a direct call and return

saved by a call to *setjmp()*, and so is 16-byte aligned. Thus the check for this special *jmp* instruction matches the check for a return instruction: *test ecx, 0xc00000f*.

Optimizations. Indirect jump instructions which are used for switch statements, such as *jmp jtable[eax*4]*, do not need dynamic checks. For any switch statement, regardless of what its control expression is, the control flow in the binary generated by modern compilers (e.g., GCC and VC) is forced to one entry in its jump table. For example, GCC first makes a bound check against *eax* (corresponding to the case value in switch statements). If it exceeds the bound, then *eax* is assigned with a default value (corresponding to the default case). And then, the control flow transfers through *jmp jtable[eax*4]*. In this way, the control flow is always forced to the jump table entries and thus cannot be hijacked by attackers. Thus BitRewrite skips validating these indirect jump instructions, to improve performance.

D. Compatibility Issues

A protected module only allows indirect control transfers whose targets are valid Springboard stubs. But the stubs are not restricted to be within the current module's Springboard section. Stubs within other modules' Springboard sections are also permitted, since their addresses are compatible; they are validated the same way. And thus if every module in a program (i.e. the main program and all DLLs) is rewritten, according to the scheme described in the previous section, [illegible] modules will be compatible with each other in [illegible] combination and the control-flow integrity is enforced.

However, rewriting all modules is not always possible in practice (e.g. system DLLs on Windows 7 cannot be

```
0000000000400586 <scn>:
  400586:       55                      push   %rbp
  400587:       48 89 e5                mov    %rsp,%rbp
  40058a:       48 83 ec 10             sub    $0x10,%rsp
  40058e:       48 8d 45 f0             lea    -0x10(%rbp),%rax
  400592:       48 89 c6                mov    %rax,%rsi
  400595:       bf 54 06 40 00          mov    $0x400654,%edi
  40059a:       b8 00 00 00 00          mov    $0x0,%eax
  40059f:       e8 cc fe ff ff          callq  400470 <__isoc99_scanf@plt>
  4005a4:       90                      nop
  4005a5:       c9                      leaveq
  4005a6:       c3                      retq

00000000004005a7 <main>:
  4005a7:       55                      push   %rbp
  4005a8:       48 89 e5                mov    %rsp,%rbp
  4005ab:       b8 00 00 00 00          mov    $0x0,%eax
  4005b0:       e8 d1 ff ff ff          callq  400586 <scn>
  4005b5:       bf 57 06 40 00          mov    $0x400657,%edi
  4005ba:       e8 91 fe ff ff          callq  400450 <puts@plt>
  4005bf:       bf 61 06 40 00          mov    $0x400661,%edi
  4005c4:       e8 87 fe ff ff          callq  400450 <puts@plt>
  4005c9:       b8 00 00 00 00          mov    $0x0,%eax
  4005ce:       5d                      pop    %rbp
  4005cf:       c3                      retq
```

-> the original code

```
0000000000400586 <scn>:
  400586:       55                      push   %rbp
  400587:       48 89 e5                mov    %rsp,%rbp
  40058a:       48 83 ec 10             sub    $0x10,%rsp
  40058e:       48 8d 45 f0             lea    -0x10(%rbp),%rax
  400592:       48 89 c6                mov    %rax,%rsi
  400595:       bf 54 06 40 00          mov    $0x400654,%edi
  40059a:       b8 00 00 00 00          mov    $0x0,%eax
  40059f:       e9 d1 0a 00 00          jmpq   401075 <__FRAME_END__+0x8bd>
  4005a4:       90                      nop
  4005a5:       90                      nop
  4005a6:       90                      nop

00000000004005a7 <main>:
  4005a7:       55                      push   %rbp
  4005a8:       48 89 e5                mov    %rsp,%rbp
  4005ab:       b8 00 00 00 00          mov    $0x0,%eax
  4005b0:       e9 e0 0a 00 00          jmpq   401095 <__FRAME_END__+0x8dd>
  4005b5:       bf 57 06 40 00          mov    $0x400657,%edi
  4005ba:       e8 91 fe ff ff          callq  400450 <puts@plt>
  4005bf:       bf 61 06 40 00          mov    $0x400661,%edi
  4005c4:       e8 87 fe ff ff          callq  400450 <puts@plt>
  4005c9:       b8 00 00 00 00          mov    $0x0,%eax
  4005ce:       5d                      pop    %rbp
  4005cf:       c3                      retq
```

-> the call is modified

```
Disassembly of section .extension:

0000000000401075 <.extension>:
  401075:       e8 f6 f3 ff ff          callq  400470 <__isoc99_scanf@plt>
  40107a:       90                      nop
  40107b:       c9                      leaveq
  40107c:       50                      push   %rax
  40107d:       8d 05 12 00 00 00       lea    0x12(%rip),%eax        # 401095 <__FRAME_END__+0x8dd>
  401083:       2b 44 24 08             sub    0x8(%rsp),%eax
  401087:       83 f8 00                cmp    $0x0,%eax
  40108a:       7e 05                   jle    401091 <__FRAME_END__+0x8d9>
  40108c:       e9 1f f7 ff ff          jmpq   4007b0 <__GNU_EH_FRAME_HDR+0x148>
  401091:       58                      pop    %rax
  401092:       c3                      retq
  401093:       90                      nop
  401094:       90                      nop

Disassembly of section .springboard:

0000000000401095 <.springboard>:
  401095:       e8 ec f4 ff ff          callq  400586 <scn>
  40109a:       e9 16 f5 ff ff          jmpq   4005b5 <main+0xe>
  40109f:       90                      nop
  4010a0:       90                      nop
  4010a1:       90                      nop
  4010a2:       90                      nop
  4010a3:       90                      nop
  4010a4:       90                      nop
ambarish@ambarish-Precision-Tower-5810:~/Desktop/dem$
```

-> this is the code in our added sections.

The checkpoint :

push %rax #to save the current value of rax
lea 0x12(%rip),%eax #loading the springboard address into rax
sub 0x8(%rsp),%eax #subtracting the return address from the stack and rax (rsp pointing to the 8 bytes below the return address)
cmp 0x0,%eax #checking if the return address is greater than the springboard address (we need to make sure the return address is inside springboard section)
jle 401091 #jumping to return if everything is fine
jmpq 4007b0 #error handling, just a random address in the #binary to cause seg fault.
pop %rax #restore the value of rax
retq #return

Indirect Call

❿ The return statements are handled the same way as the direct call.

❿ As for the call, it also is dealt like the direct call but we dont know the actuall address of the function being called as it is read from the register.

❿ Right now, we dont have an automatic technique for this but one can manually see the address loaded in the register by tracing back few assembly instructions.

Benchmark

We tried our approach a to benchmark binary : **kill**. The results are as below.

- ➓ Number of direct calls = 245(220 library calls, 25 normal calls)
- ➓ Number of indirect calls = 2
- ➓ Number of returns = 14 (2 sensitive returns and 12 normal ones)

we remove our checkpoint for the 2 sensitive returns as they are called from the system and return to the system.

The funcionality has been tested by trying a few commands like:

- ➓ ./kill
- ➓ ./kill --help
- ➓ ./kill --version
- ➓ ./kill -9 $PID

Performance has been tested by creating and killing the process 10,000 times(I wrote a c++ script).

Original

0.859882
0.854217
0.854568
0.854012

= 0.85570 +/- 0.003 , deviation = 0.35%

Fortified

0.861137
0.861522
0.850526
0.852335

= 0.85640 +/- 0.006 , deviation = 0.70%

overhead = 0.08%

The above times are in seconds. The overhead is so small(even smaller than the deviation) as kill is a very small binary.