Software Vulnerabilities: Exploitation and Mitigation
Lab 6

Artem Kaliahin
artem.kaliahin.001@student.uni.lu

April 2019

Question 1.1 Describe the source code.

main function - creates two arrays *ia* and *ib* with a length of 10, then calls *sort2* function with 3 arguments (arrays *ia*, *ab* and number of arguments in command line *argc*) —>
sort2 function - takes 3 arguments (2 arrays and integer), then calls function *sort* twice. First time with array *ia*, integer *argc* and pointer to function *lt*. Second time - with array *ib*, integer *argc* and pointer to function *gt* —>
sort function - takes 3 arguments (during the first call - *ia*, *argc* and pointer to function *lt*, while second call takes *ib*, *argc* and pointer to *gt*), then it calls function *gt* or *lt* (depending on the argument) with arguments *ia[argc]* and *ia[argc+1]* (or *ib[argc]* and *ib[argc+1]*) —>
lt function - takes two arguments and returns 1 if first argument is less than the second, otherwise - returns 0 (in our case - returns 1 if *ia[argc]* < *ia[argc+1]*, otherwise - returns 0).
gt function - takes two arguments and returns 1 if first argument is bigger than the second, otherwise - returns 0 (in our case - returns 1 if *ib[argc]* > *ib[argc+1]*, otherwise - returns 0).

Question 1.2 Does gcc also support CFI?

Yes. CFI enforcement mechanism Vtable Verification (VTV) has been implemented in GCC version 4.9. [source]

Question 1.3 Describe all five options given to clang above.

-o0 - sets optimization level to 0 (no optimization).
-fsanitize=cfi - enables ALL Clang's available CFI schemes (requires -flto flag and -fvisibility flag).

-flto - enables linking time optimization in 'full' mode.
-fvisibility=hidden - every declaration not explicitly marked with a visibility attribute has a hidden visibility (hides unnecessary symbols).
-o cfi - writes output to file "cfi".

Question 1.4 In the CFI version, there is the instruction ud2 which is not in the non-CFI version. What is this instruction doing?

Instruction *ud2* - Raise invalid opcode exception, so called CFI-trap, which is generated by LLVM to crash the program (if CFI check fails, we execute a *ud2* instruction, which will crash the program).

Question 1.5 What elements (instructions, functions, etc.) are in the CFI version and not in the non-CFI version.

There are 2 functions added by CFI - *lt.cfi* and *gt.cfi*. Instead of executing *lt* and *gt* directly, it jumps to *lt.cfi* and *gt.cfi* respectively (Picture 1) and executes *lt.cfi* and *gt.cfi* (when instructions for *lt* function in non-CFI version are the same as *lt.cfi* in CFI version, applicable to *gt* function as well).

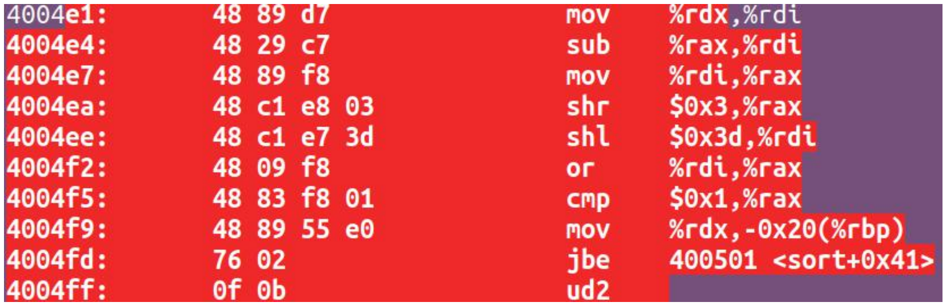
```
00000000004005d0 <lt>:
  4005d0:       e9 ab fe ff ff          jmpq   400480 <lt.cfi>
  4005d5:       cc                      int3
  4005d6:       cc                      int3
  4005d7:       cc                      int3

00000000004005d8 <gt>:
  4005d8:       e9 c3 fe ff ff          jmpq   4004a0 <gt.cfi>
  4005dd:       cc                      int3
  4005de:       cc                      int3
  4005df:       cc                      int3
```

Picture 1

Instruction added to sort function are shown on Picture 2. They check for control flow integrity and crash the program if this check fails (instruction *ud2* crashes the program if *jbe* doesn't jump on an instruction below, this jump requires flags CF = 1 or ZF = 1, which are updated by *cmp* instruction).

```
4004c8:       48 b8 d0 05 40 00 00    movabs $0x4005d0,%rax
4004cf:       00 00 00
```



```
4004e1:    48 89 d7        mov    %rdx,%rdi
4004e4:    48 29 c7        sub    %rax,%rdi
4004e7:    48 89 f8        mov    %rdi,%rax
4004ea:    48 c1 e8 03     shr    $0x3,%rax
4004ee:    48 c1 e7 3d     shl    $0x3d,%rdi
4004f2:    48 09 f8        or     %rdi,%rax
4004f5:    48 83 f8 01     cmp    $0x1,%rax
4004f9:    48 89 55 e0     mov    %rdx,-0x20(%rbp)
4004fd:    76 02           jbe    400501 <sort+0x41>
4004ff:    0f 0b           ud2
```

Picture 2

While functions *sort2* and *main* remain the same in CFI and non-CFI version.

Question 1.6 Explain how the added instructions check for CFI. Describre precisely under which conditions instruction ud2 is executed. Drow the CFG for the C program above.

```
4004c8:    48 b8 d0 05 40 00 00    movabs $0x4005d0,%rax
4004cf:    00 00 00
```

Copies \$0x4005d0 (address of *lt.cfi* function) to \$rax.

```
4004e1:    48 89 d7                mov    %rdx,%rdi
```

Copies \$rdx to \$rdi (there is address of function that was given as an argument to function in \$rdx according to the convention - address of *lt.cfi* or *gt.cfi*, whereas the difference between addresses is 8 bytes).

```
4004e4:    48 29 c7                sub    %rax,%rdi
```

Subtracts value in \$rax from value in \$rdi and puts difference in \$rdi. For now this value according to CFG can be either 0 or 8, because \$rdi can have either the same value as \$rax if *lt.cfi* is called or 8 bytes more if *gt.cfi* is called.

```
4004e7:    48 89 f8                mov    %rdi,%rax
```

Copies the result of subtraction to \$rax.

```
4004ea:    48 c1 e8 03             shr    $0x3,%rax
4004ee:    48 c1 e7 3d             shl    $0x3d,%rdi
```

Here we shift \$rax value 3 bits to the right and \$rdi value 62 bits to the left. In this case the only value that will eventually make zero in both registers after shifts is 0 or 8 (100_2) - this is valid behaviour. But it is also possible if the value is 24 (1100_2), but in our case it is not a threat since it will jump to the program's __libc_csu_init library.

```
4004f2:    48 09 f8                or     %rdi,%rax
```

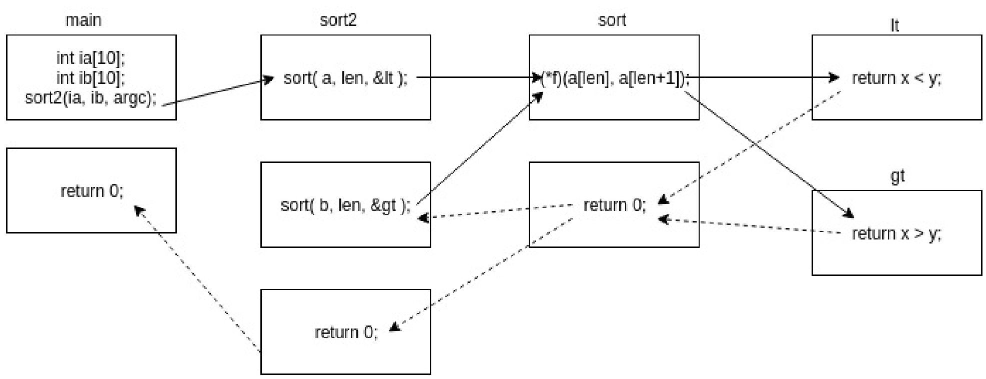
This instruction puts to \$rax the result of OR operation on \$rdi and \$rax. It will be equal to 0 only if both registers have value 0.

```
4004f5:    48 83 f8 01          cmp    $0x1,%rax
4004fd:    76 02                jbe    400501 <sort+0x41>
```

Here we compare \$rax with 0x1 (subtract \$rax from 0x1). If control flow is not disrupted, \$rax will store 0-value as proved above and *cmp* will change flags needed to execute *jbe* (jump if below or equal) instruction. It will jump to address 400501 (if CF or ZF flags are equal to 1) and jump over instruction *ud2* to the next one. Jump will also be executed if \$rax stores value 0x1.

```
4004ff:    0f 0b                ud2
```

Instruction *ud2* crashes program. It will be executed ONLY when jump is not executed (above it is described precisely under which conditions jump is not executed).



Picture 3. Control Flow Graph