Control Flow Transformation Assignment

Date : 28 - 09 - 2025

Raj Shekhar - 240545002004

Topic : Reverse Engineering Obfuscated Control Flow Through State Machine Analysis

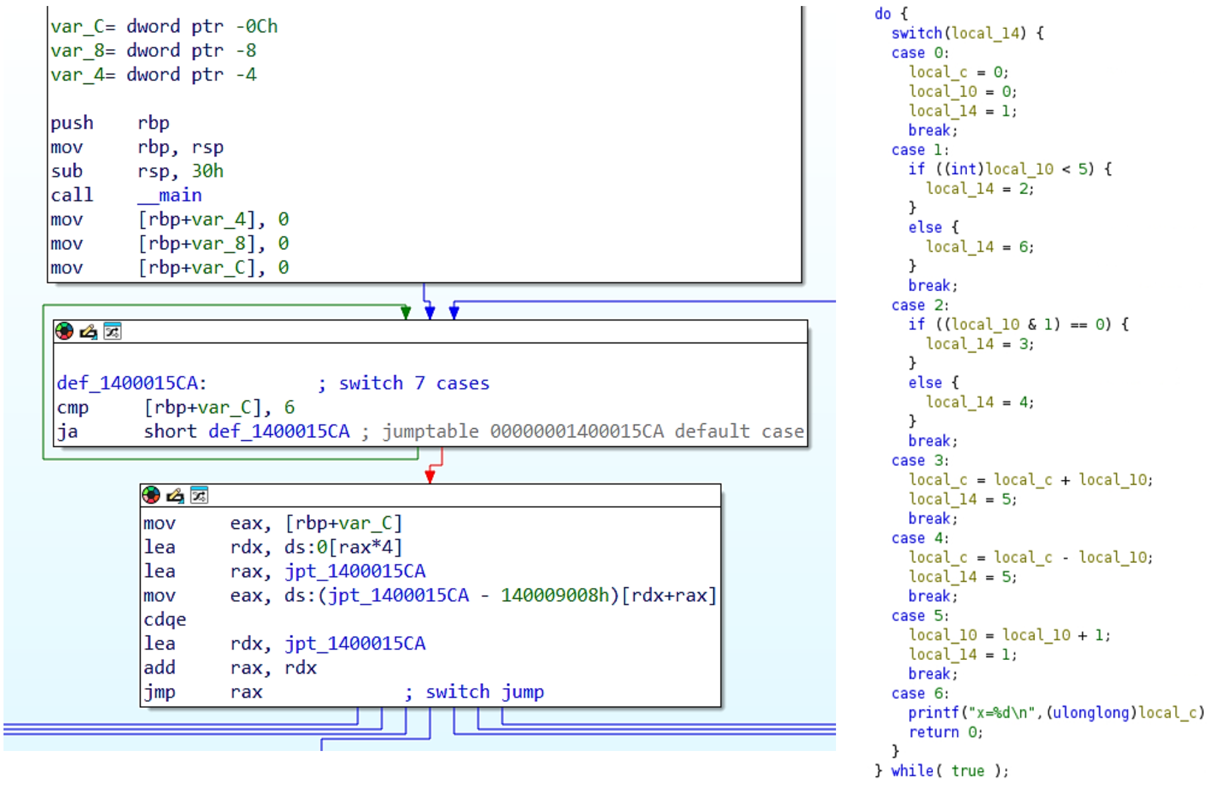
Objective : Analyse a obfuscated C program that uses state machine patterns to hide its actual functionality.

Introduction

Control flow transformation is a sophisticated technique used to protect sensitive code and being reverse engineered. This method is used to manually transforms a structured code into something that preserves its original functionality while reducing its readability as code. In this assignment patterns analysed shows the control flow flattening to hide the program logic.

The challenge is the ability to conceal execution patterns that would otherwise be immediately apparent in traditional structured programming. Where a normal program would utilize clear loop constructs and conditional statements, obfuscated code implements these same logic patterns through state transitions and a central dispatcher. This technique transforms its structured control flow graph into what reverse engineers often describe as "spaghetti code" (online term). Tracing the actual program behavior, and understanding the vulnerabilities inherent in these patterns are very useful to perform any reverse engineering.

Analysis



```
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

push    rbp
mov     rbp, rsp
sub     rsp, 30h
call    __main
mov     [rbp+var_4], 0
mov     [rbp+var_8], 0
mov     [rbp+var_C], 0

def_1400015CA:          ; switch 7 cases
cmp     [rbp+var_C], 6
ja      short def_1400015CA ; jumptable 00000001400015CA default case

mov     eax, [rbp+var_C]
lea     rdx, ds:0[rax*4]
lea     rax, jpt_1400015CA
mov     eax, ds:(jpt_1400015CA - 140009008h)[rdx+rax]
cdqe
lea     rdx, jpt_1400015CA
add     rax, rdx
jmp     rax                 ; switch jump
```

```
do {
  switch(local_14) {
  case 0:
    local_c = 0;
    local_10 = 0;
    local_14 = 1;
    break;
  case 1:
    if ((int)local_10 < 5) {
      local_14 = 2;
    }
    else {
      local_14 = 6;
    }
    break;
  case 2:
    if ((local_10 & 1) == 0) {
      local_14 = 3;
    }
    else {
      local_14 = 4;
    }
    break;
  case 3:
    local_c = local_c + local_10;
    local_14 = 5;
    break;
  case 4:
    local_c = local_c - local_10;
    local_14 = 5;
    break;
  case 5:
    local_10 = local_10 + 1;
    local_14 = 1;
    break;
  case 6:
    printf("x=%d\n",(ulonglong)local_c)
    return 0;
  }
} while( true );
```

Fig1: Spotted Obfuscation code in IDA pro & Ghidra with do-while loop, switch and if-else

The State Diagram (in a Linear Sequence) : It is a directed graph whose nodes are the states 0, 1, 2, 3, 4, 5, 6. Edges indicate state transitions, like :

State 0 → State 1

State 1 → State 2 (if i < 5) # Conditional Branching

State 1 → State 6 (if i >= 5) # Conditional Branching

State 2 → State 3 (if i % 2 == 0) # Conditional Branching

State 2 → State 4 (otherwise)

State 3 → State 5

State 4 → State 5

State 5 → State 1

State 6 → (exit)

States 1→2→3, 4→5→1, shows : Cyclic Behavior Pattern

From full execution, the net effect is:

When even i: x += i

When odd i: x -= i

Overall, the obfuscated code is for to hide variable x such that its result is not visible through any disassembler easily by searching it. But analysing the code in disassembler one can conclude the following results :

Loop occurred for i = 0, 1, 2, 3, 4

Then ends with, case 1:

```
if (i < 5) # i -> 5, in case 5
  else # triggered
    state = 6; # the single exit point
```

And the final result is :

x = (0) – 1 + 2 – 3 + 4 = 2

Hence output will print x = 2.

So, control flow transformation is occurring using a central dispatcher (switch over state) and jumping between "basic block states" instead of structured loops/ifs.

Summary

Features of Control Flow Obfuscation :

- The code uses a state machine dispatcher instead of traditional loops.
- The while(1) loop with internal state transitions creates non-linear execution flow.
- This pattern hides the actual program logic behind state transitions.

State Transition Patterns are :

- Linear sequencing : States progress in numerical order during initialization.
- Conditional branching : State 1 and State 2 introduce conditional paths.
- Cyclic behavior : States 1→2→3 & 4→5→1 form a loop structure.
- Termination condition : State 6 provides the exit mechanism.

How Data Obfuscation performed :

- Variables x and i are modified across multiple states.
- The relationship between operations is distributed across the state machine.
- The final computation emerges from cumulative state transitions.

Structural Limitations :

- Single exit point despite multiple logical paths.
- State variables controlling flow instead of explicit conditionals.
- Mixed concerns within what appears to be a simple loop, modifications are complex to be made.

Some Vulnerability Assessment to perform (Researched from sources like Google)

- Identify potential maintenance risks in the state machine approach
- Analyze error handling and edge case management
- Suggest improvements for robustness