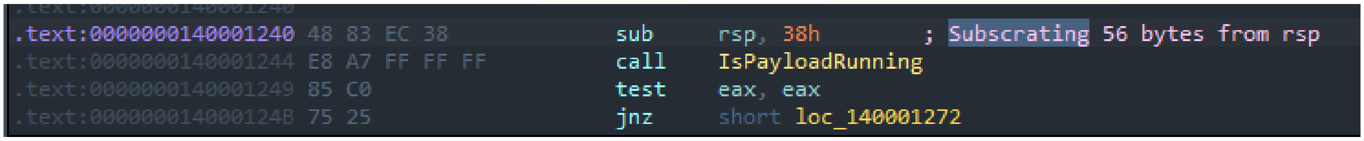
Summary

This time we are going to analyze 3 different methods to control the execution of the payload. There are several types of synchronization objects, including semaphores, mutexes, and events. Each type of synchronization object works in a slightly different manner but ultimately they all serve the same purpose which is to coordinate access of shared resources.

Sample 1 (Semaphore)

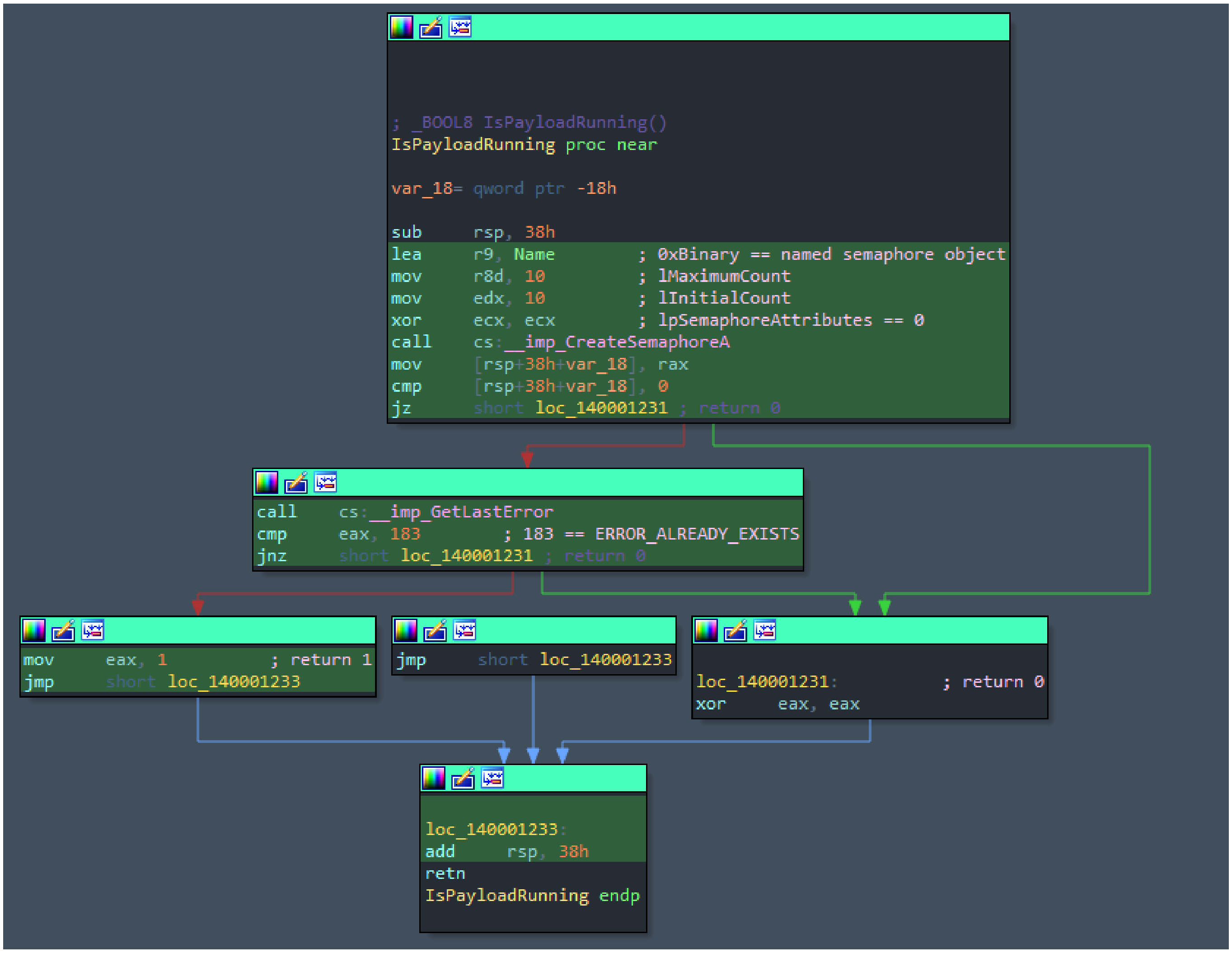
We can see see at at this `main()` subroutine is very small, is only calls 1 other subroutine named `IsPayloadRunning()` which we will dive into a bit. First we can see that the main function is subtracting 0x38 (56 bytes) from rsp register, this is done in order to make space space on the stack.

Next, another subroutine is called at: `140001244` named: `IsPayloadRunning()`



```
.text:0000000140001240
.text:0000000140001240 48 83 EC 38                sub     rsp, 38h        ; Subscrating 56 bytes from rsp
.text:0000000140001244 E8 A7 FF FF FF             call    IsPayloadRunning
.text:0000000140001249 85 C0                      test    eax, eax
.text:000000014000124B 75 25                      jnz     short loc_140001272
```

This subroutine called `CreateSemaphoreA()` WinAPI. This Windows API, is responsible for creating / opening a semaphore object. In this case a named semaphore to prevent executions after the initial binary run. If the named semaphore is already running, `CreateSemaphoreA` will return a handle to the existing object and `GetLastError` will return `ERROR_ALREADY_EXISTS`.



```
; _BOOL8 IsPayloadRunning()
IsPayloadRunning proc near

var_18= qword ptr -18h

sub     rsp, 38h
lea     r9, Name         ; 0xBinary == named semaphore object
mov     r8d, 10          ; lMaximumCount
mov     edx, 10          ; lInitialCount
xor     ecx, ecx         ; lpSemaphoreAttributes == 0
call    cs:__imp_CreateSemaphoreA
mov     [rsp+38h+var_18], rax
cmp     [rsp+38h+var_18], 0
jz      short loc_140001231 ; return 0
```

```
call    cs:__imp_GetLastError
cmp     eax, 183         ; 183 == ERROR_ALREADY_EXISTS
jnz     short loc_140001231 ; return 0
```

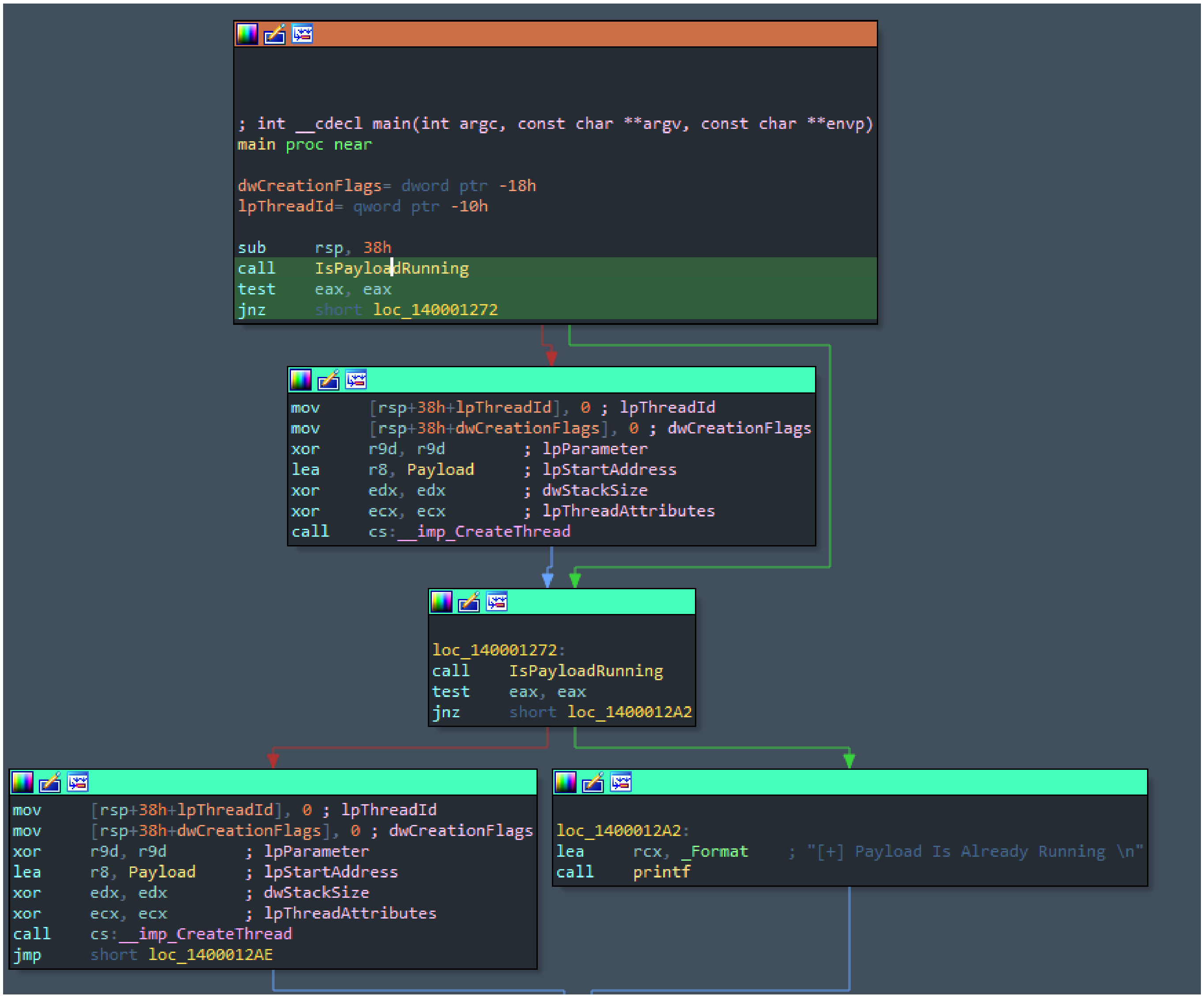
```
mov     eax, 1           ; return 1
jmp     short loc_140001233
```

```
jmp     short loc_140001233
```

```
loc_140001231:           ; return 0
xor     eax, eax
```

```
loc_140001233:
add     rsp, 38h
retn
IsPayloadRunning endp
```

After a new semaphore object has been created, the next stage is to run the actual payload. This is done via `CreateThread()` and assign `lpStartAddress` to our payload. Next after successful execution, the program is going to check whether a semaphore named object has been created named `0xBinary` if not, the program is going to create thread if the named semaphore already exists the program will print `[+] Payload Is Already Running\n`.



```
; int __cdecl main(int argc, const char **argv, const char **envp)
main proc near

dwCreationFlags= dword ptr -18h
lpThreadId= qword ptr -10h

sub     rsp, 38h
call    IsPayloadRunning
test    eax, eax
jnz     short loc_140001272
```

```
mov     [rsp+38h+lpThreadId], 0 ; lpThreadId
mov     [rsp+38h+dwCreationFlags], 0 ; dwCreationFlags
xor     r9d, r9d          ; lpParameter
lea     r8, Payload       ; lpStartAddress
xor     edx, edx          ; dwStackSize
xor     ecx, ecx          ; lpThreadAttributes
call    cs:__imp_CreateThread
```

```
loc_140001272:
call    IsPayloadRunning
test    eax, eax
jnz     short loc_1400012A2
```

```
mov     [rsp+38h+lpThreadId], 0 ; lpThreadId
mov     [rsp+38h+dwCreationFlags], 0 ; dwCreationFlags
xor     r9d, r9d          ; lpParameter
lea     r8, Payload       ; lpStartAddress
xor     edx, edx          ; dwStackSize
xor     ecx, ecx          ; lpThreadAttributes
call    cs:__imp_CreateThread
jmp     short loc_1400012AE
```

```
loc_1400012A2:
lea     rcx, _Format    ; "[+] Payload Is Already Running \n"
call    printf
```

Pseudo-Code C Sempahore

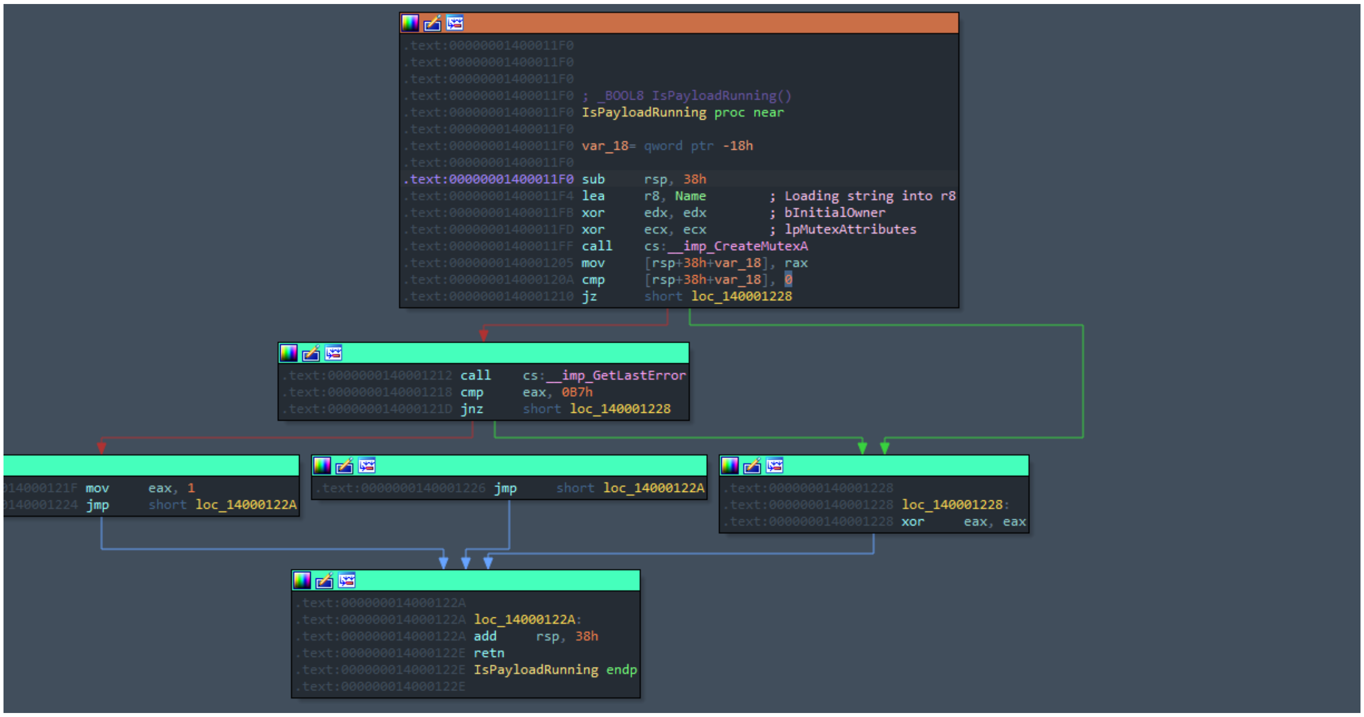
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  if ( !IsPayloadRunning() )
    CreateThread(0i64, 0i64, (LPTHREAD_START_ROUTINE)Payload, 0i64, 0, 0i64);
  if ( IsPayloadRunning() )
    printf("[+] Payload Is Already Running \n");
  else
    CreateThread(0i64, 0i64, (LPTHREAD_START_ROUTINE)Payload, 0i64, 0, 0i64);
  getchar();
  return 0;
}
```

Mutex

Mutex is a specific kind of binary semaphore that is used to provide a locking mechanism. It stands for Mutual Exclusion Object. Mutex is mainly used to provide mutual exclusion to a specific portion of the code so that the process can execute and work with a particular section of the code at a particular time.

The binary looks very the same as functionality as semaphore, therefore we are only gong to dive into the mutex functionality inside this specimen.

We can see the screenshot below, that it's creating mutex named object, therefore it's locking this object in order to execute `CreateThread()` once.



```
.text:00000001400011F0
.text:00000001400011F0
.text:00000001400011F0
.text:00000001400011F0 ; _BOOL8 IsPayloadRunning()
.text:00000001400011F0 IsPayloadRunning proc near
.text:00000001400011F0
.text:00000001400011F0 var_18= qword ptr -18h
.text:00000001400011F0
.text:00000001400011F0 sub     rsp, 38h
.text:00000001400011F4 lea     r8, Name          ; Loading string into r8
.text:00000001400011FB xor     edx, edx          ; bInitialOwner
.text:00000001400011FD xor     ecx, ecx          ; lpMutexAttributes
.text:00000001400011FF call    cs:__imp_CreateMutexA
.text:0000000140001205 mov     [rsp+38h+var_18], rax
.text:000000014000120A cmp     [rsp+38h+var_18], 0
.text:0000000140001210 jz      short loc_140001228
```

```
.text:0000000140001212 call    cs:__imp_GetLastError
.text:0000000140001218 cmp     eax, 0B7h
.text:000000014000121D jnz     short loc_140001228
```

```
14000121F mov     eax, 1
140001224 jmp     short loc_14000122A
```

```
.text:0000000140001226 jmp     short loc_14000122A
```

```
.text:0000000140001228
.text:0000000140001228 loc_140001228:
.text:0000000140001228 xor     eax, eax
```

```
.text:000000014000122A
.text:000000014000122A loc_14000122A:
.text:000000014000122A add     rsp, 38h
.text:000000014000122E retn
.text:000000014000122E IsPayloadRunning endp
.text:000000014000122E
```

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rdx
  __int64 v4; // rcx
  __int64 MutexString; // r8

  if ( !(unsigned int)IsPayloadRunning(argc, argv, envp) )
  {
    printf("[i] Running Payload [1] ... ");
    CreateThread(0i64, 0i64, (LPTHREAD_START_ROUTINE)Payload, 0i64, 0, 0i64);
    printf("[+] DONE \n");
  }
  if ( (unsigned int)IsPayloadRunning(v4, v3, MutexString) )
  {
    printf("[+] Payload Is Already Running \n");
  }
  else
  {
    printf("[i] Running Payload [2] ... ");
    CreateThread(0i64, 0i64, (LPTHREAD_START_ROUTINE)Payload, 0i64, 0, 0i64);
    printf("[+] DONE \n");
  }
  getchar();
  return 0;
}
```