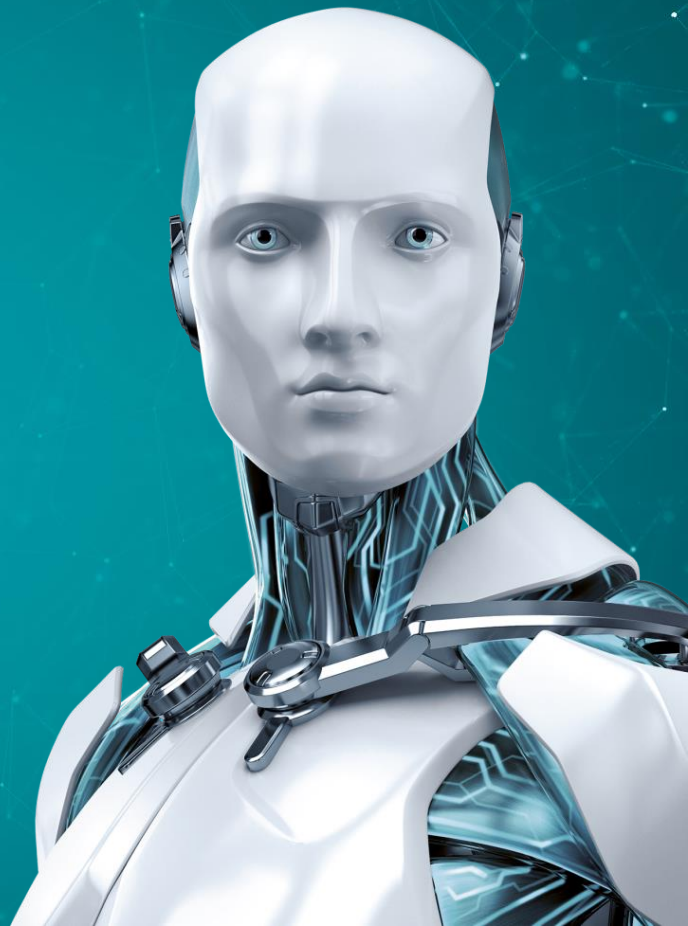
Digital Security
Progress. Protected.

Under the hood of Wslink's multilayered virtual machine

Vladislav Hrčka | Malware Researcher



Vladislav Hrčka

Malware Researcher at ESET

@HrckaVladislav

Agenda

- Intro to VMs in general and symbolic execution
- Internals of the VM used in Wslink
- Our approach to dealing with the obfuscation
- Demonstration of the approach

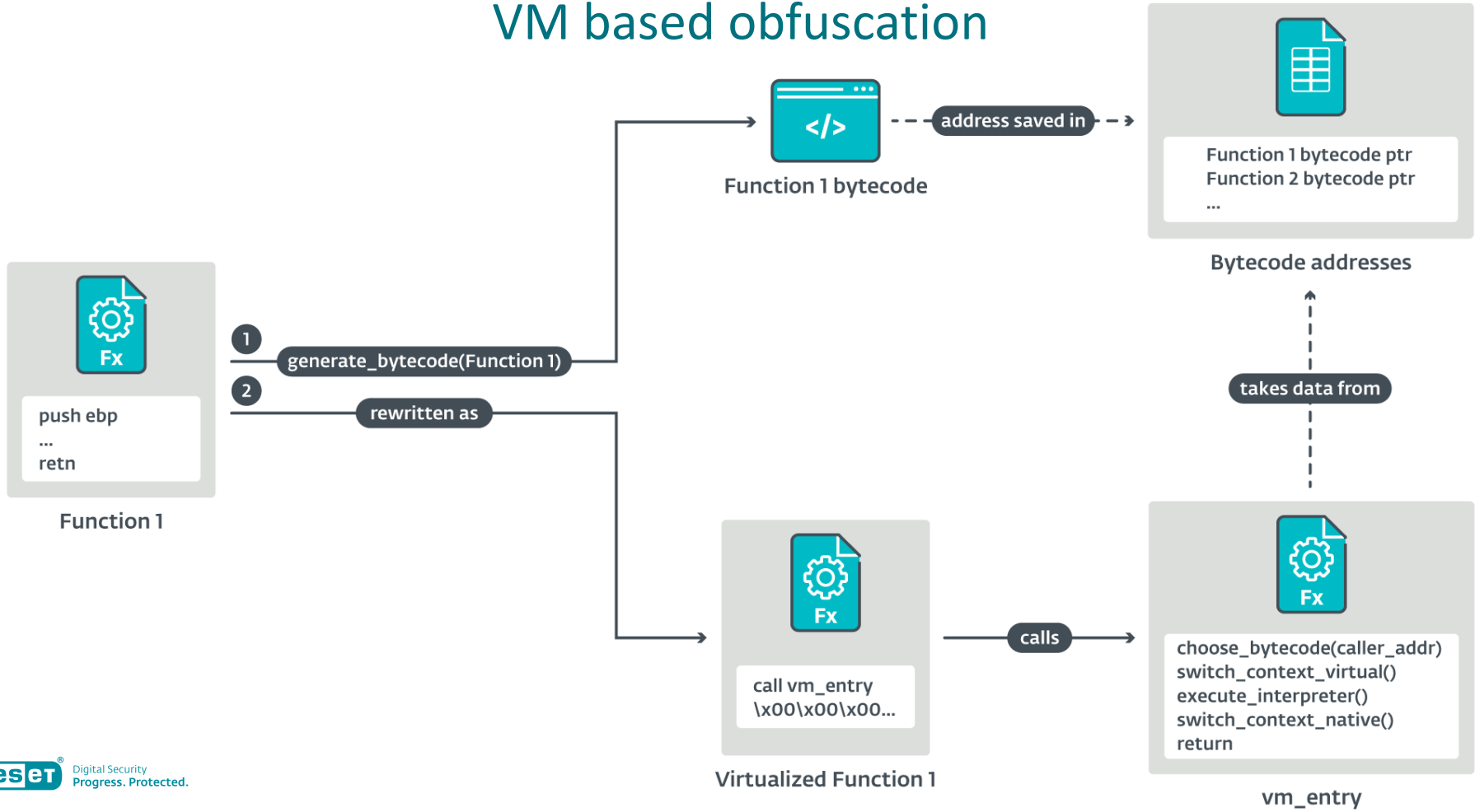
Process VMs basics

```python
def handler1(vm_ctx, bytecode):
    operand1 = bytecode[vm_ctx.PC]
    operand2 = bytecode[vm_ctx.PC+1]
    vm_ctx.PC += 2
    vm_ctx.gp_regs[operand1] = vm_ctx.gp_regs[operand2]

def handler2(vm_ctx, bytecode):
    operand1 = bytecode[vm_ctx.PC]
    vm_ctx.PC += 1
    vm_ctx.gp_regs[vm_ctx.stack.pop()] = operand1

def handler3(vm_ctx, bytecode):
    operand1 = bytecode[vm_ctx.PC]
    operand2 = bytecode[vm_ctx.PC+1]
    vm_ctx.PC += 2
    if vm_ctx.gp_regs[operand1] == operand2:
        exit()

...

handlers = [handler1, handler2, handler3, ...]

def vm_interpreter(vm_ctx, bytecode):
    while True:
        opcode = bytecode[vm_ctx.PC]
        vm_ctx.PC += 1
        handlers[opcode-1](vm_ctx, bytecode)
```

01	08	09	01	07	08	02	85	03	01	01

- Interpreter executes the bytecode
- Bytecode contains instructions with:
  - Opcodes
  - Operands
- Handlers define individual opcodes

VM based obfuscation



Symbolic execution in Miasm

- Expresses the code in mathematical formulas
- Registers and memory are treated as symbolic values
- Summarizes the code's effects on the symbolic values
- We will frequently use it

- Original ASM:

```
MOV EAX, EBX
MOV ECX, DWORD PTR [EDX]
XOR ECX, 0x123
MOV AX, WORD PTR [ESI]
JMP ECX
```

- Performed symbolic execution:

```
EAX = {@16[ESI] 0 16, EBX[16:32] 16 32}
ECX = @32[EDX] ^ 0x123
zf = @32[EDX] == 0x123
nf = (@32[EDX] ^ 0x123)[31:32]
...
EIP = @32[EDX] ^ 0x123
IRDst = @32[EDX] ^ 0x123
```

Symbolic execution in Miasm

- Allows us to simply apply known concrete values
- Concrete values can simplify the expressions
- Performed Symbolic execution:

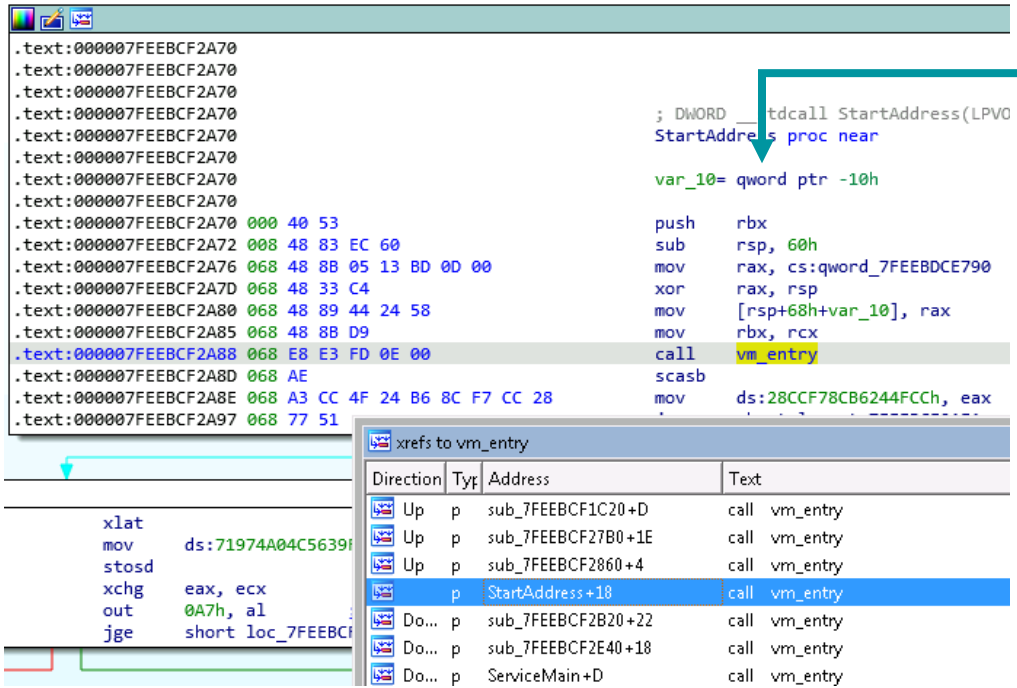
```
EAX = {@16[ESI] 0 16, EBX[16:3…
ECX = @32[EDX] ^ 0x123
zf = @32[EDX] == 0x123
nf = (@32[EDX] ^ 0x123)[31:32]
...
EIP = @32[EDX] ^ 0x123
IRDst = @32[EDX] ^ 0x123
```

- Applied `EDX = 0x96` and `@32[0x96] = 0xFEED`:

```
EAX = {@16[ESI] 0 16, EBX[16:3…
ECX = 0xFFCE
zf = 0x0
nf = 0x0
...
EIP = 0xFFCE
IRDst = 0xFFCE
EDX = 0x96
@32[0x96] = 0xFEED
```

End of introduction

Wslink: First contact



- A virtualized function
- Virtualized functions contain:
  - Prologue
  - Call to VM entry
  - Gibberish code

Junk code

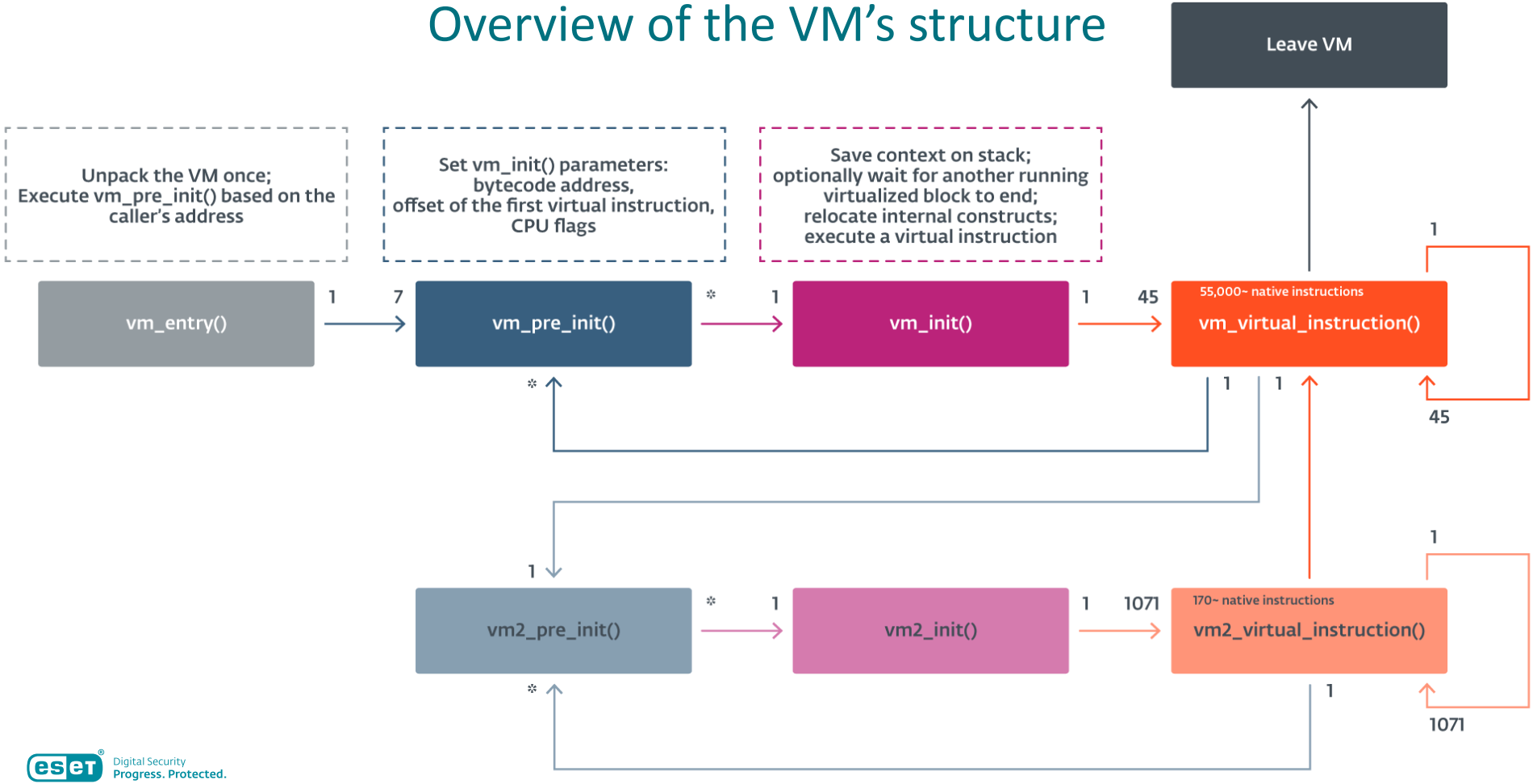
IDA View-A | Symbolic Execution - 0x11dfd8 to 0x11e842

```
RAX = call_func_ret(0x11DFDD, RSP_init, RCX_init, RDX_init, R8_init, R9_init)
RBX = 0x127
RCX = 0x1
RSP = call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFF80
RBP = @64[call_func_stack(0x11DFDD, RSP_init)] + 0xFFFFFFFFFFFDB229
zf = RSI_init == 0x0
nf = (RSI_init)[63:64]
pf = parity(RSI_init & 0xFF)
of = 0x0
cf = 0x0
af = ((call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFF78) ^ (call_func_sta
IRDst = loc_key_3
@64[call_func_stack(0x11DFDD, RSP_init)] = call_func_ret(0x11DFDD, RSP_init, RCX_
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFF58] = RDX_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFF60] = call_func_stack(0
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFF68] = call_func_stack(0
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFF70] = call_func_stack(0
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFF78] = R12_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFF80] = @64[call_func_sta
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFF88] = R8_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFF90] = R9_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFF98] = R10_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFFA0] = R11_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFFA8] = R12_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFFB0] = R13_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFFB8] = R14_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFFC0] = R15_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFFC8] = RDI_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFFD0] = RSI_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFFD8] = RBP_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFFE0] = RBX_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFFE8] = RBX_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFFF0] = RDX_init
@64[call_func_stack(0x11DFDD, RSP_init) + 0xFFFFFFFFFFFFFFF8] = RCX_init
```

Pseudocode-A

```
84  v32 = (__int64 *)(v86[0] ^ v31);
85  v86[0] = v20;
86  v85 = 0x4B6F99F2i64;
87  v84 = v28;
88  v83 = (__int64)v32;
89  v82 = (char *)v30;
90  v81 = (__int64)v32;
91  v33 = _InterlockedExchange64((volatile __int64 *)&v82, (
92  v84 = v20;
93  v34 = v83;
94  v83 = v21;
95  v82 = (char *)v23;
96  v81 = 0x2AF80900i64;
97  v80 = 0x50D361D3i64;
98  v79 = 0x78D1A8C6i64;
99  v78 = v25;
100 v77 = v24;
101 v35 = _InterlockedExchange64(&v78, (__int64)&v78);
102 v78 = v19;
103 v79 = (__int64)v22;
104 v82 = (char *)v22;
105 v81 = a2;
106 v36 = a2 ^ (unsigned __int64)&v81;
107 v81 ^= v36;
108 v37 = v81 ^ v36;
109 v80 = v33;
110 _InterlockedExchange64(&v80, (__int64)&v80);
111 v82 = (char *)v29;
112 v81 = 0x74E27FBEi64;
113 v80 = v37;
114 v38 = _InterlockedExchange64(&v80, (__int64)&v80);
115 v82 = (char *)v34;
116 v77 = v32;
117 v78 = (__int64)v32;
118 v81 = (__int64)v32;
119 v80 = 0x276C41B1i64;
120 v79 = (__int64)v24;
121 v39 = _InterlockedExchange64(&v79, (__int64)&v79);
122 v79 = a4;
123 v40 = _InterlockedExchange64(&v79, (__int64)&v79);
124 v81 = v26;
```

Overview of the VM's structure



VM2: The first executed virtual instruction

- Prepares the next virtual instruction
- Increases VPC (virtual program counter)
- Zeroes out a register

- `RBP_init` is context pointer
- Instruction table at offset `0xA4`
- VPC at offset `0x28`

Symbolic Execution - 0xe8a7a to 0xe8ad4

```
RAX = RBP_init + 0xB5
RCX = @64[@64[RBP_init + 0xA4] + {0x0, 0, 3, @16[@64[RBP_init + 0x28]], 3, 19, 0x0, 19, 64}]
RIP = @64[@64[RBP_init + 0xA4] + {0x0, 0, 3, @16[@64[RBP_init + 0x28]], 3, 19, 0x0, 19, 64}]
RSI = {0x0, 0, 3, @16[@64[RBP_init + 0x28]], 3, 19, 0x0, 19, 64}
R10 = @64[RBP_init + 0xA4] + {0x0, 0, 3, @16[@64[RBP_init + 0x28]], 3, 19, 0x0, 19, 64}
R13 = RBP_init + 0x28
zf = @64[RBP_init + 0x28] == 0xFFFFFFFFFFFFFFFC
nf = (@64[RBP_init + 0x28] + 0x4)[63:64]
pf = parity((@64[RBP_init + 0x28] + 0x4) & 0xFF)
of = ((@64[RBP_init + 0x28] ^ (@64[RBP_init + 0x28] + 0x4)) & (@64[RBP_init + 0x28] ^ 0xFFFFFF
cf = (@64[RBP_init + 0x28] ^ ((@64[RBP_init + 0x28] ^ (@64[RBP_init + 0x28] + 0x4)) & (@64[RBP_
af = (@64[RBP_init + 0x28] ^ (@64[RBP_init + 0x28] + 0x4) ^ 0x4)[4:5]
IRDst = @64[@64[RBP_init + 0xA4] + {0x0, 0, 3, @16[@64[RBP_init + 0x28]], 3, 19, 0x0, 19, 64}]
@64[RBP_init + 0x28] = @64[RBP_init + 0x28] + 0x4
@32[RBP_init + 0xB5] = 0x0
```

VM2: Virtual instructions 2 and 3

- Instruction 2
  - Zeroes out several virtual registers

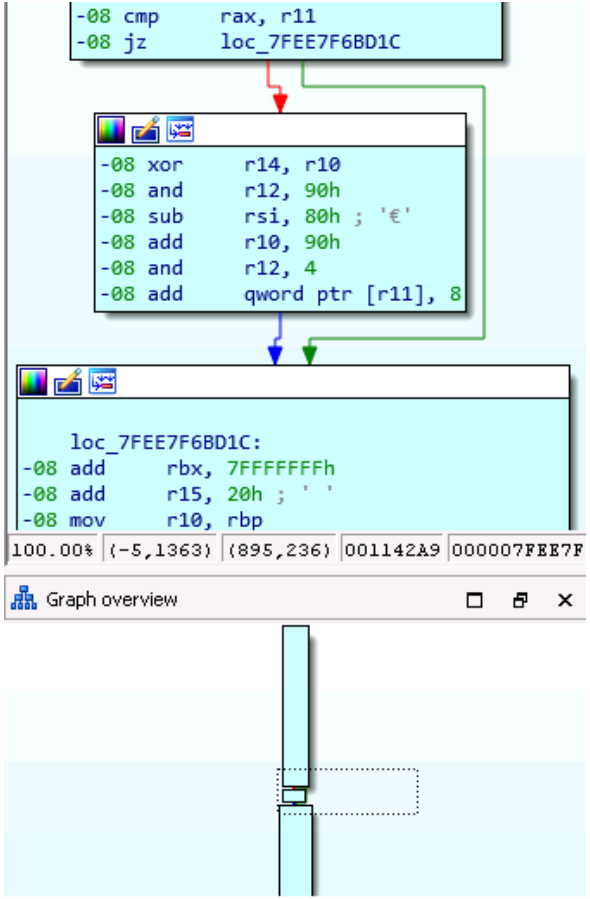
```
IRDst = @64[@64[RBP_init + 0xA4] + {0x0, 0, 3, @16[@64[RBP_init + 0x28]], 3, 19, 0x0, 19, 64}]
@16[RBP_init + 0xB] = 0x0
@64[RBP_init + 0x28] = @64[RBP_init + 0x28] + 0x2
@32[RBP_init + 0x48] = 0x0
@32[RBP_init + 0x70] = 0x0
@32[RBP_init + 0x94] = 0x0
@32[RBP_init + 0xA0] = 0x0
@32[RBP_init + 0xEE] = 0x0
@32[RBP_init + 0xFA] = 0x0
@16[RBP_init + 0x103] = 0x0
@32[RBP_init + 0x133] = 0x0
@16[RBP_init + 0x149] = 0x0
```

- Instruction 3
  - Stores RSP in a virtual register

```
IRDst = @64[@64[RBP_init + 0xA4] + {0x0, 0, 3, @16[@64[RBP_init + 0x28] + 0x2], 3, 19, 0x0, 19, 64}]
@64[RBP_init + {@16[@64[RBP_init + 0x28]], 0, 16, 0x0, 16, 64}] = RSP_init
@64[RBP_init + 0x28] = @64[RBP_init + 0x28] + 0x4
```

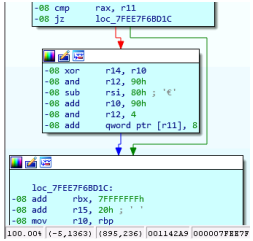
VM2: Virtual instruction 4

- Multiple basic blocks:



VM2: Virtual instruction 4

- Multiple basic blocks:

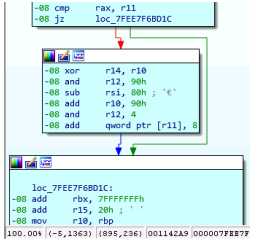


- Summary of the first block – memory assignments and IRDst:

```
IRDst = ({{@16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})[0:16]) ^ 0x3038, 0, 16, 0x0, 16, 64} == {@16[@64[RBP_init + 0x28] + 0x6], 0,
@16[RBP_init + 0xB] = @16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})[0:16]
@32[RBP_init + 0x70] = @32[RBP_init + 0x70] & (@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})
@64[RBP_init +  (@16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})[0:16]) ^ 0x3038, 0, 16, 0x0, 16, 64}] = @64[RSP_init]
```

VM2: Virtual instruction 4

- Multiple basic blocks:

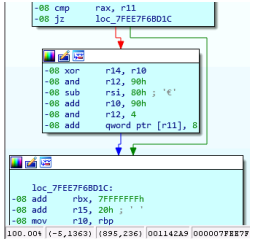


- Summary of the first block – memory assignments and IRDst:

```
IRDst = ({(@16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64[RBP_i
@16[RBP_init + 0xB] = @16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@1
@32[RBP_init + 0x70] = @32[RBP_init + 0x70] & (@32[RBP_init + 0x70] ^ {@
@64[RBP_init + {(@16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64
```

VM2: Virtual instruction 4

- Multiple basic blocks:



- Summary of the first block – memory assignments and IRDst:

```
IRDst = ({{@16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})[0:16]) ^ 0x3038, 0, 16, 0x0, 16, 64} == {@16[@64[RBP_init + 0x28] + 0x6], 0,
@16[RBP_init + 0xB] = @16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})[0:16]
@32[RBP_init + 0x70] = @32[RBP_init + 0x70] & (@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})
@64[RBP_init +  (@16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})[0:16]) ^ 0x3038, 0, 16, 0x0, 16, 64}] = @64[RSP_init]
```

- Rolling decryption – encryption of operands
  - Values of virtual registers at offsets `0x0B` and `0x70` were set earlier

"You're not allowed to use the sprinkler system to keep your audience awake."

VM2: Virtual instruction 4 - deobfuscation

- The first original block

```
IRDst = (({@16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})[0:16]) ^ 0x3038, 0, 16, 0x0, 16, 64} == {@16[@64[RBP_init + 0x28] + 0x6], 0,
@16[RBP_init + 0xB] = @16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})[0:16]
@32[RBP_init + 0x70] = @32[RBP_init + 0x70] & (@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})
@64[RBP_init + {@16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ @16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})[0:16]) ^ 0x3038, 0, 16, 0x0, 16, 64}] = @64[RSP_init]
```

VM2: Virtual instruction 4 - deobfuscation

- The first original block

```
 = ({(@16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64[RBP_init
BP_init + 0xB] = @16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@6
BP_init + 0x70] = @32[RBP_init + 0x70] & (@32[RBP_init + 0x70] ^ {@16[@
BP_init + {[@16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64[RBP
```

VM2: Virtual instruction 4 - deobfuscation

- The first original block

```
IRDst = (({@16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})[0:16]) ^ 0x3038, 0, 16, 0x0, 16, 64} == {@16[@64[RBP_init + 0x28] + 0x6], 0,
@16[RBP_init + 0xB] = @16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})[0:16]
@32[RBP_init + 0x70] = @32[RBP_init + 0x70] & (@32[RBP_init + 0x70] ^ {@16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})
@64[RBP_init + {@16[RBP_init + 0xB] + -(@32[RBP_init + 0x70] ^ @16[@64[RBP_init + 0x28] + 0x4], 0, 16, 0x0, 16, 32})[0:16]) ^ 0x3038, 0, 16, 0x0, 16, 64}] = @64[RSP_init]
```

- Application of known values of the rolling decryption registers

```
IRDst = (-@16[@64[RBP_init + 0x28] + 0x4] ^ 0x3038 == @16[@64[RBP_init +
0x28] + 0x6])?(0x7FEC91ABD1C,0x7FEC91ABCF6)
@64[RBP_init + {-@16[@64[RBP_init + 0x28] + 0x4] ^ 0x3038, 0, 16, 0x0,
16, 64}] = @64[RSP_init]
```

- Application of known bytecode values reveals `POP`

```
IRDst = @64[@64[RBP_init + 0xA4] + 0x5A8]
@64[RBP_init + 0x28] = @64[RBP_init + 0x28] + 0x8
@64[RBP_init + 0x141] = @64[RBP_init + 0x141] + 0x8
@64[RBP_init + 0x12A] = @64[RSP_init]
```

VM2: Deobfuscating bytecode chunks

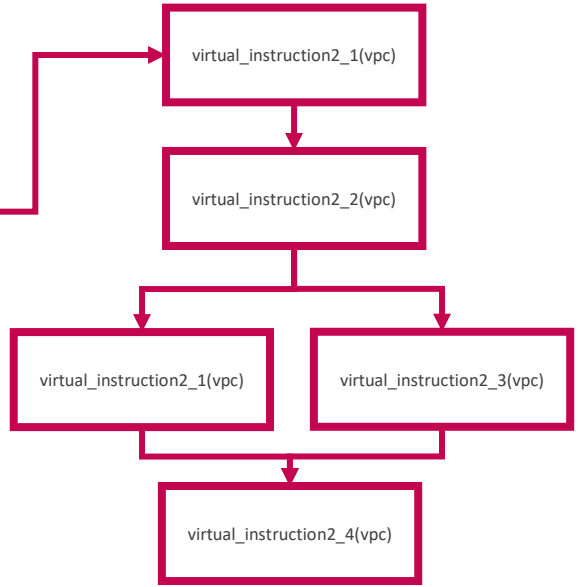
- Virtual instruction 4 summary:

```
IRDst = @64[@64[RBP_init + 0xA4] + 0x5A8]
@64[RBP_init + 0x28] = @64[RBP_init + 0x28] + 0x8
@64[RBP_init + 0x141] = @64[RBP_init + 0x141] + 0x8
@64[RBP_init + 0x12A] = @64[RSP_init]
```

VM2: Deobfuscating bytecode chunks

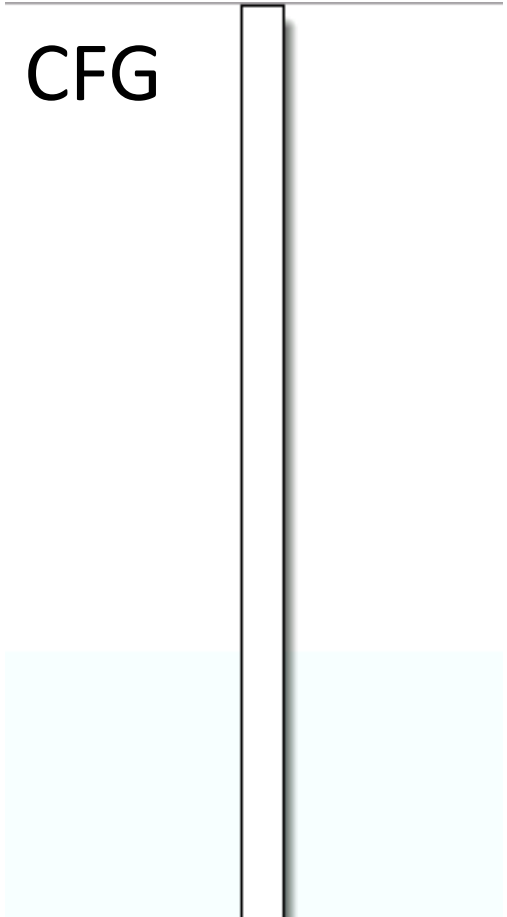
- Virtual instruction 4 summary:

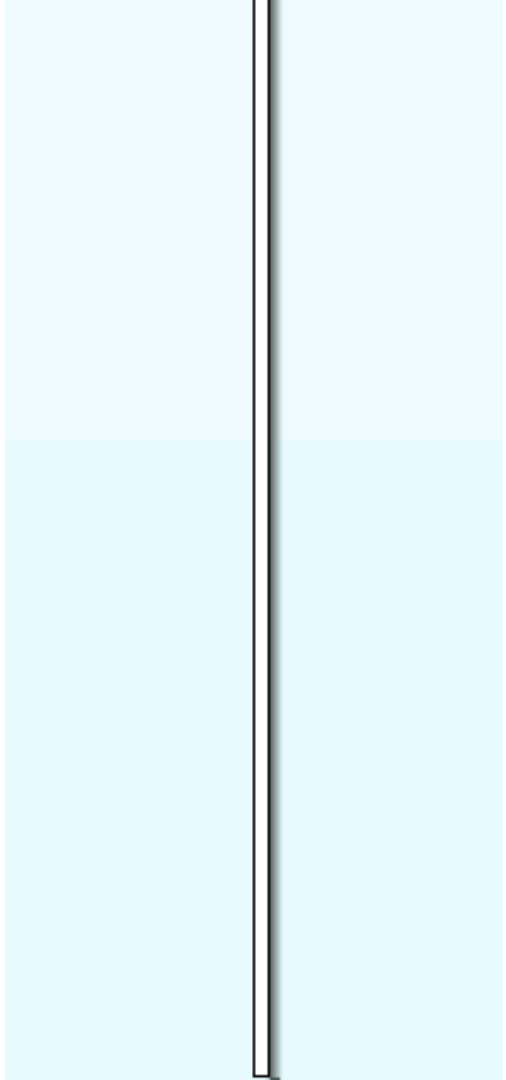
```
IRDst = @64[@64[RBP_init + 0xA4] + 0x5A8]
@64[RBP_init + 0x28] = @64[RBP_init + 0x28] + 0x8
@64[RBP_init + 0x141] = @64[RBP_init + 0x141] + 0x8
@64[RBP_init + 0x12A] = @64[RSP_init]
```



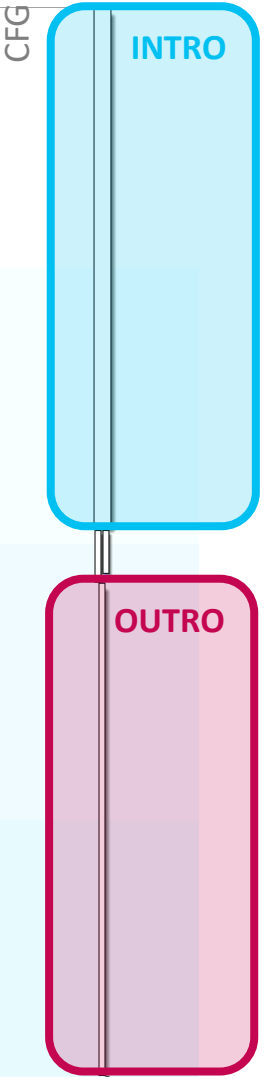
- Build a graph from summaries
- Treat some values as concrete:
  - Rolling decryption registers
  - Memory accesses relative to the bytecode pointer
- Preserve only decryption registers' values between blocks

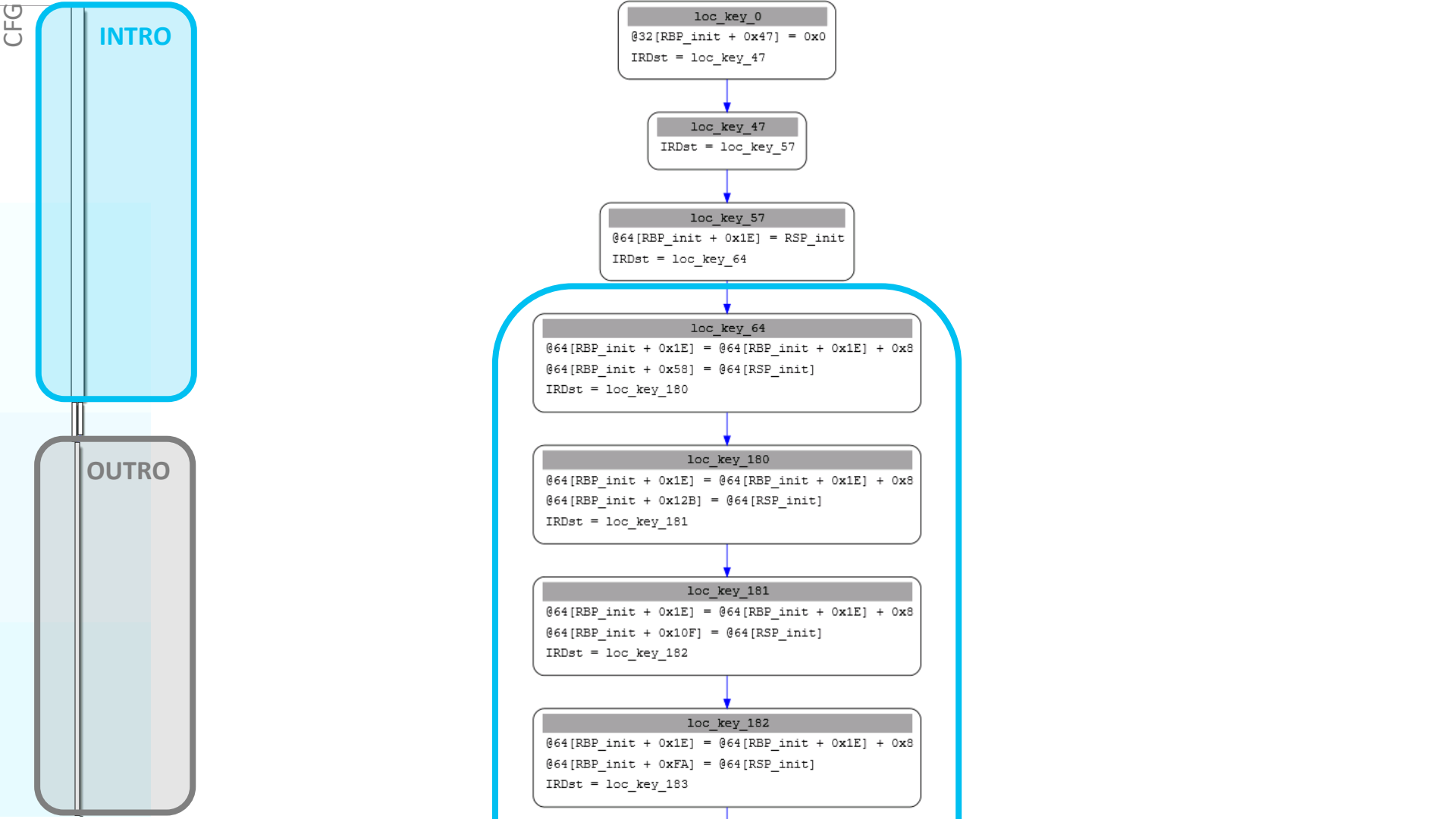
VM1: Deobfuscated virtual instruction structure

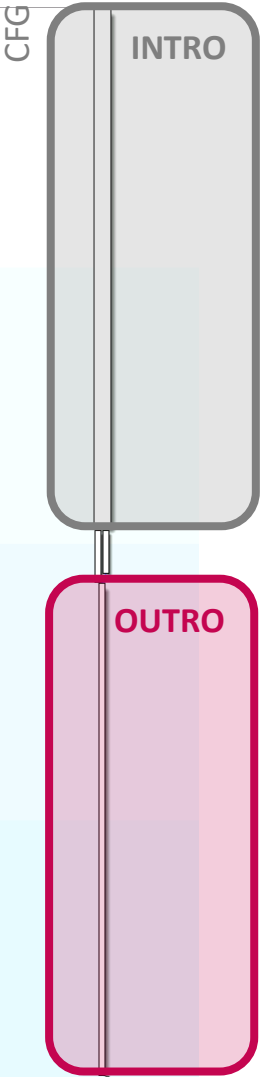




VM1: Deobfuscated virtual instruction structure




CFG
INTRO
OUTRO
loc_key_0
@32[RBP_init + 0x47] = 0x0
IRDst = loc_key_47
loc_key_47
IRDst = loc_key_57
loc_key_57
@64[RBP_init + 0x1E] = RSP_init
IRDst = loc_key_64
loc_key_64
@64[RBP_init + 0x1E] = @64[RBP_init + 0x1E] + 0x8
@64[RBP_init + 0x58] = @64[RSP_init]
IRDst = loc_key_180
loc_key_180
@64[RBP_init + 0x1E] = @64[RBP_init + 0x1E] + 0x8
@64[RBP_init + 0x12B] = @64[RSP_init]
IRDst = loc_key_181
loc_key_181
@64[RBP_init + 0x1E] = @64[RBP_init + 0x1E] + 0x8
@64[RBP_init + 0x10F] = @64[RSP_init]
IRDst = loc_key_182
loc_key_182
@64[RBP_init + 0x1E] = @64[RBP_init + 0x1E] + 0x8
@64[RBP_init + 0xFA] = @64[RSP_init]
IRDst = loc_key_183




```
@64[RSP_init + 0xFFFFFFFFFFFFFFF8] = @64[RBP_init + 0x98]
@64[RBP_init + 0x141] = @64[RBP_init + 0x141] + 0xFFFFFFFFFFFFFFF8

@64[RSP_init + 0xFFFFFFFFFFFFFFF8] = @64[RBP_init + 0x10D]
@64[RBP_init + 0x141] = @64[RBP_init + 0x141] + 0xFFFFFFFFFFFFFFF8
```

```
R10 = @64[RSP_init + 0x10]
R11 = @64[RSP_init + 0x18]
R12 = @64[RSP_init + 0x20]
R13 = @64[RSP_init + 0x28]
R14 = @64[RSP_init + 0x30]
R15 = @64[RSP_init + 0x38]
zf = @32[RSP_init + 0x78][6:7]
```

```
exception_flags = @32[RSP_init + 0x78][8:9]?(0x2,exception_flags_init)
IRDst = @64[RBP_init + 0x74]
@32[RBP_init + 0xFF] = 0x0
```

Legend:

Yellow – Push virtual registers

Red – Pop virtual registers; switch context

Green – Jump to register

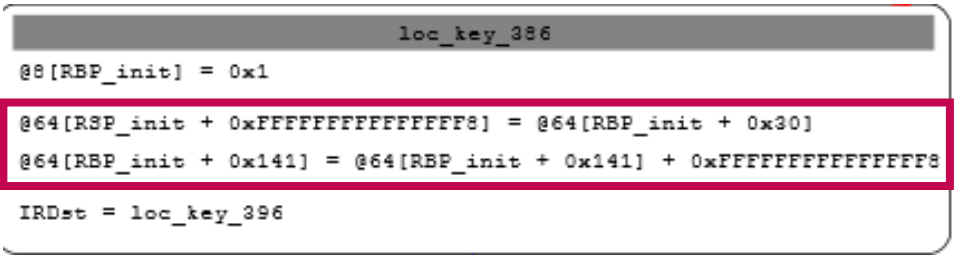
VM1: Initially executed virtual instructions

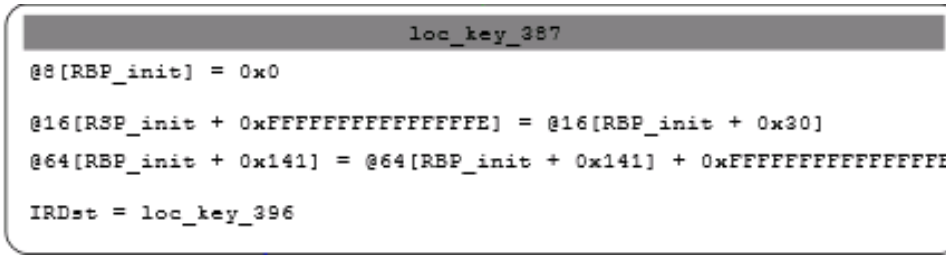
- Virtual instructions 1, 2, 3
  - The same behavior as in VM2

VM1: Virtual instruction 4

VM1: Virtual instruction 4

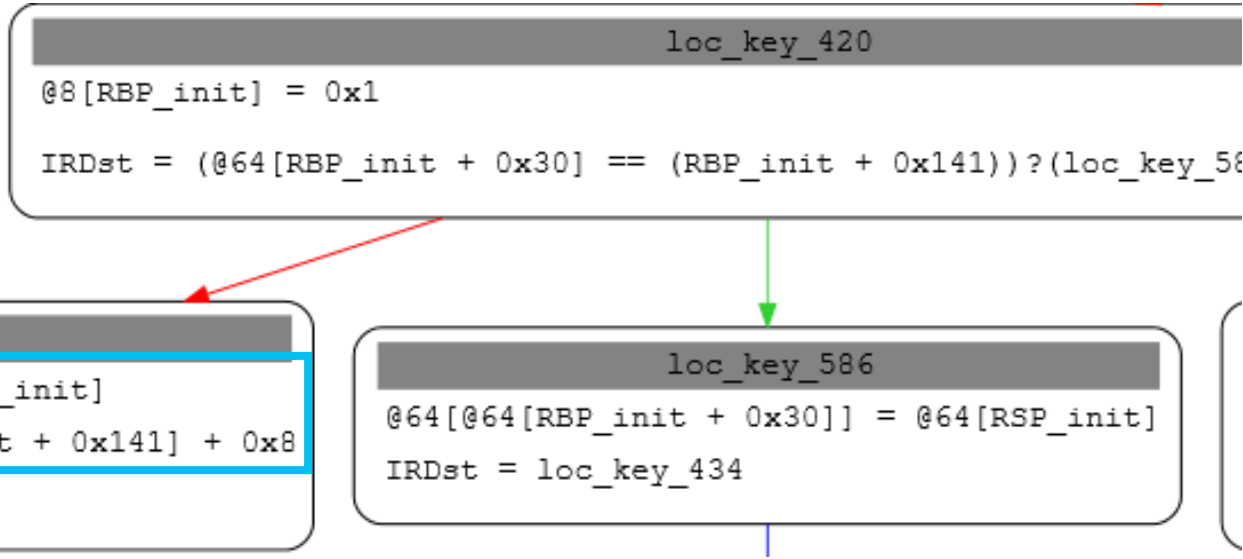
- Instruction merging – contains, e.g., `POP` and `PUSH` operations
- Operands decide which instruction is executed
- `PUSH`:





VM1: Virtual instruction 4

- Instruction merging – contains, e.g., `POP` and `PUSH` operations
- Operands decide which instruction is executed
- `POP`:

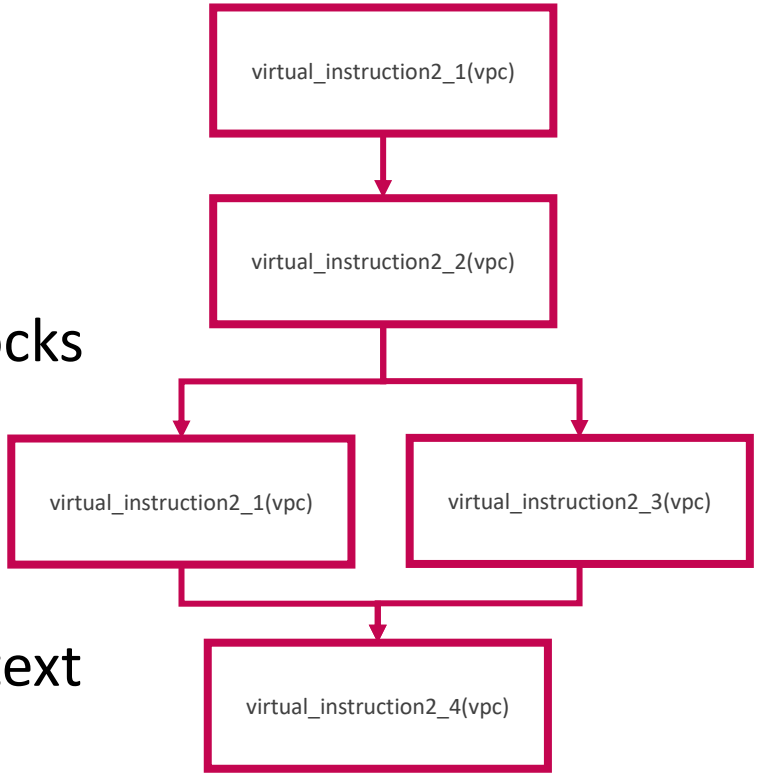


VM1: Virtual instruction 4

- Instruction merging – contains, e.g., `POP` and `PUSH` operations
- Operands decide which instruction is executed
- `PUSH/POP`
- Can be simplified by making the **operands concrete**

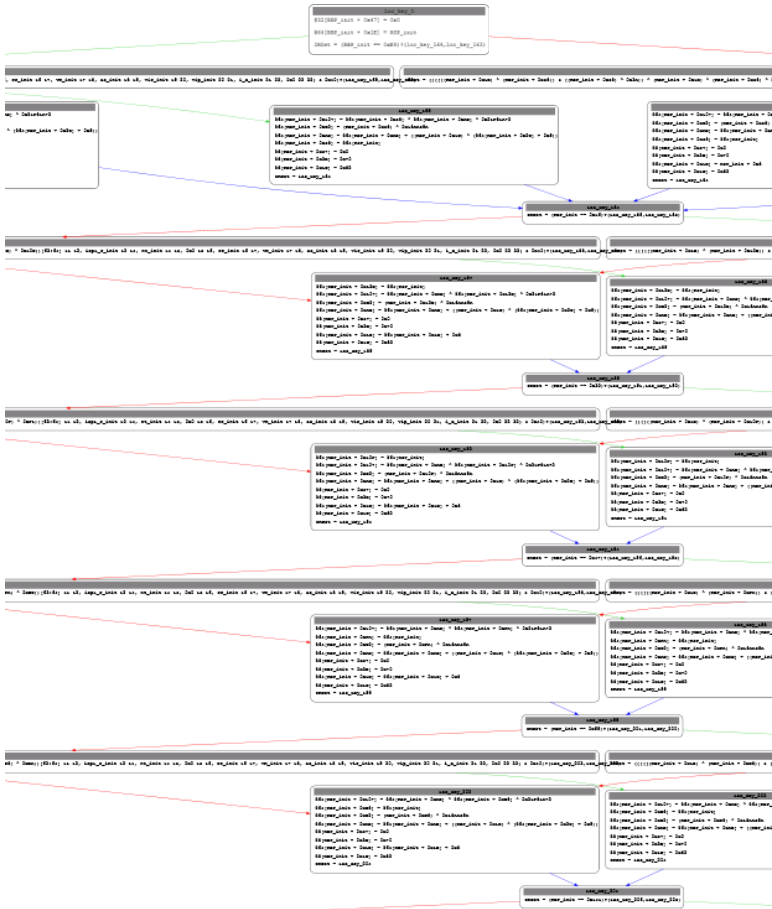
VM1: Deobfuscating bytecode chunks

- Use the same approach as in VM2:
  - Build a graph from summaries
  - Treat certain values as concrete
  - Preserve certain values between blocks
- Process both VMs at once:
  - Additionally make the entire VM2 concrete
  - Ignore assignments to the VM2 context



VM1: Issue with opaque predicates during deobfuscation

Resulting graph contained unexpected branches instead of a series of `POP`s



VM1: Issue with opaque predicates during deobfuscation

```
loc_key_0
@32[RBP_init + 0x47] = 0x0

@64[RBP_init + 0x1E] = RSP_init

IRDst = (RBP_init == 0xE9)?(loc_key_164,loc_key_163)
```

- The branches check a known value
  - We can apply the value and simplify it
  - This is a sort of opaque predicates

Does the approach work?

Analyzing results

Bytecode block	VM1	VM2
Size in bytes	695	1,145
Total number of processed virtual instructions	62	109
Total number of underlying native instructions	3,536,427	17,406
Total number of resulting IR instructions (including IRDsts)	192	307
Execution time in seconds	382	10

Processed ServiceMain bytecode

```
ServiceMain     proc near               ; DATA XREF: .rda
                                        ; .rdata:0000000
arg_8           = qword ptr  10h

                mov     [rsp+arg_8], rbx
                push    rdi
                sub     rsp, 20h
                mov     rbx, rdx
                call    sub_1800F2870
                fdivr   st, st(6)
; ---------------------------------------------------------
                dd 628D92C7h
```

```
public ServiceMain
ServiceMain proc near

arg_0= qword ptr  8
arg_8= qword ptr  10h

mov     [rsp+10h], rbx
push    rdi
sub     rsp, 20h
mov     rbx, [rdx]
mov     eax, 28
lea     rdi, ServiceStatus
nop     dword ptr [rax+00000000h]
```

```
loc_180003060:
dec     rax
mov     byte ptr [rax+rdi], 0
```

OBFUSCATED

NON-OBFUSCATED SAMPLE*

DEOBFUSCATED

```
@64[RBP_init+  0x15] = @64[RSP_init]

@64[RBP_init+  0x11F] = @64[RSP_init]
@64[RBP_init+  0x1E] = RSP_init+  0x78

@64[RBP_init+  0x4F] = @64[RSP_init]
@64[RBP_init+  0x1E] = RSP_init+  0x80

@64[RBP_init+  0xCC] = @64[RSP_init]
@64[RBP_init+  0x1E] = RSP_init+  0x88

@64[RBP_init+  0x1E] = RSP_init+  0x98

@64[RBP_init+  0x13F] = @64[@64[RBP_init+  0x13F]]

@32[RBP_init+  0x53] = 0x0
@32[RBP_init+  0x4F] = 0x1C

@64[RBP_init+  0x133] = 0x3092

@64[RBP_init+  0x133] = @64[RBP_init+  0x80] + 0x3092

@64[RBP_init+  0x133] = @64[RBP_init+  0x80] + 0xE1808

@64[RBP_init+  0x74] = @64[RBP_init+  0x80] + 0xE3808
IRDst=  loc_key_291
```

```
mov    R0, [R0]
mov    R1, 28
```

```
                                public ServiceMain
                                ServiceMain proc near

                                arg_0= qword ptr  8
                                arg_8= qword ptr  10h

5C 24 10                        mov     [rsp+10h], rbx
                                push    rdi
EC 20                           sub     rsp, 20h
1A                              mov     rbx, [rdx]
00 00 00                        mov     eax, 28
3D AF 07 0E 00                  lea     rdi, ServiceStatus
80 00 00 00 00                  nop     dword ptr [rax+00000000h]

                                loc_180003060:
FF C8                           dec     rax
04 38 00                        mov     byte ptr [rax+rdi], 0
F7                              jnz     short loc_180003060

10 FF FF FF                     lea     rdx, HandlerProc ; lpHandlerPr
                                mov     rcx, rbx         ; lpServiceNam
8F 08 00                        call    cs:RegisterServiceCtrlHandlerW
B0 07 0E 00                     mov     cs:hServiceStatus, rax
                                test    rax, rax
                                jz      short loc_1800030FD
```

DEOBFUSCATED

SIMPLIFIED

ORIGINAL SAMPLE

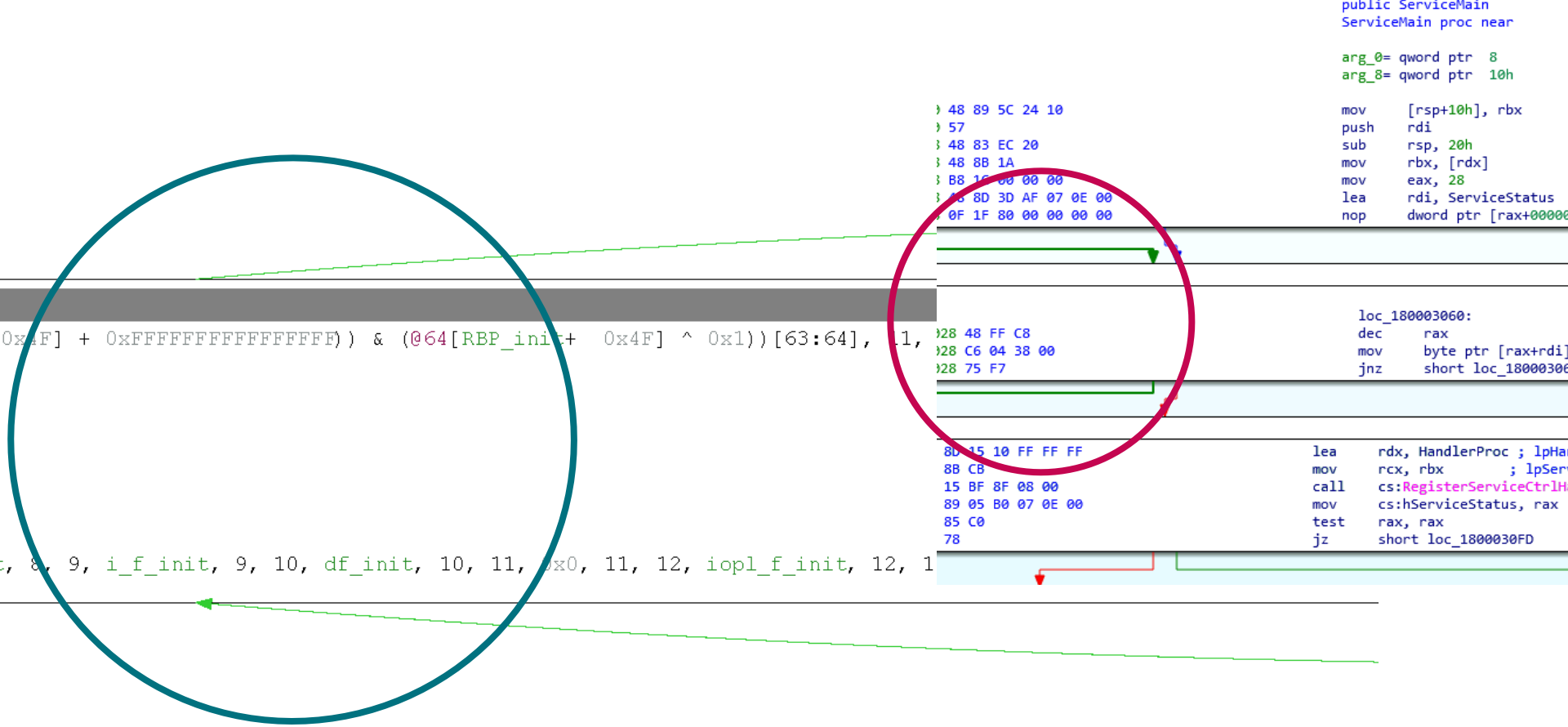
```
@64[RBP_init+  0x15] = @64[RSP_init]

@64[RBP_init+  0x11F] = @64[RSP_init]
@64[RBP_init+  0x1E] = RSP_init+  0x78

@64[RBP_init+  0x4F] = @64[RSP_init]
@64[RBP_init+  0x1E] = RSP_init+  0x80

@64[RBP_init+  0xCC] = @64[RSP_init]
@64[RBP_init+  0x1E] = RSP_init+  0x88

@64[RBP_init+  0x1E] = RSP_init+  0x98

@64[RBP_init+  0x13F] = @64[@64[RBP_init+  0x13F]]

@32[RBP_init+  0x53] = 0x0
@32[RBP_init+  0x4F] = 0x1C

@64[RBP_init+  0x133] = 0x3092

@64[RBP_init+  0x133] = @64[RBP_init+  0x80] + 0x3092

@64[RBP_init+  0x133] = @64[RBP_init+  0x80] + 0xE3808

@64[RBP_init+  0x74] = @64[RBP_init+  0x80] + 0xE3808
IRDst=  loc_key_291
```

```
                                public ServiceMain
                                ServiceMain proc near

                                arg_0= qword ptr  8
                                arg_8= qword ptr  10h

5C 24 10                        mov     [rsp+10h], rbx
                                push    rdi
EC 20                           sub     rsp, 20h
1A                              mov     rbx, [rdx]
00 00 00                        mov     eax, 28
3D AF 07 0E 00                  lea     rdi, ServiceStatus
80 00 00 00 00                  nop     dword ptr [rax+00000000h]

                                loc_180003060:
FF C8                           dec     rax
04 38 00                        mov     byte ptr [rax+rdi], 0
F7                              jnz     short loc_180003060

10 FF FF FF                     lea     rdx, HandlerProc ; lpHandlerPr
                                mov     rcx, rbx         ; lpServiceNam
8F 08 00                        call    cs:RegisterServiceCtrlHandlerW
B0 07 0E 00                     mov     cs:hServiceStatus, rax
                                test    rax, rax
                                jz      short loc_1800030FD
```

lea R2, BASE+0xE3808

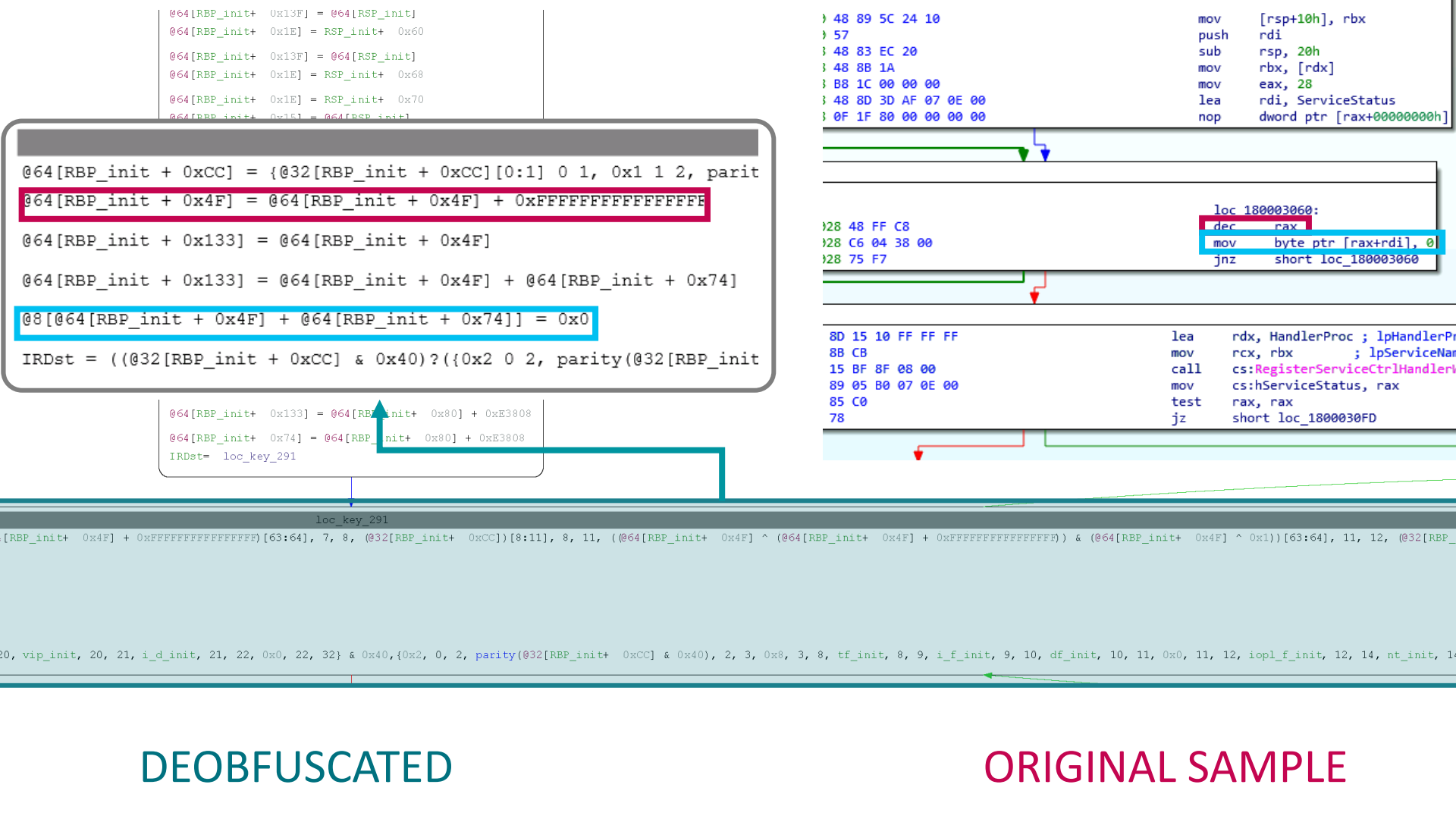
```
.data:000007FEB5FF3808      ; struct _SERVICE_STATUS ServiceStatus
.data:000007FEB5FF3808      ServiceStatus _SERVICE_STATUS <0>
.data:000007FEB5FF3808
```

DEOBFUSCATED SIMPLIFIED ORIGINAL SAMPLE



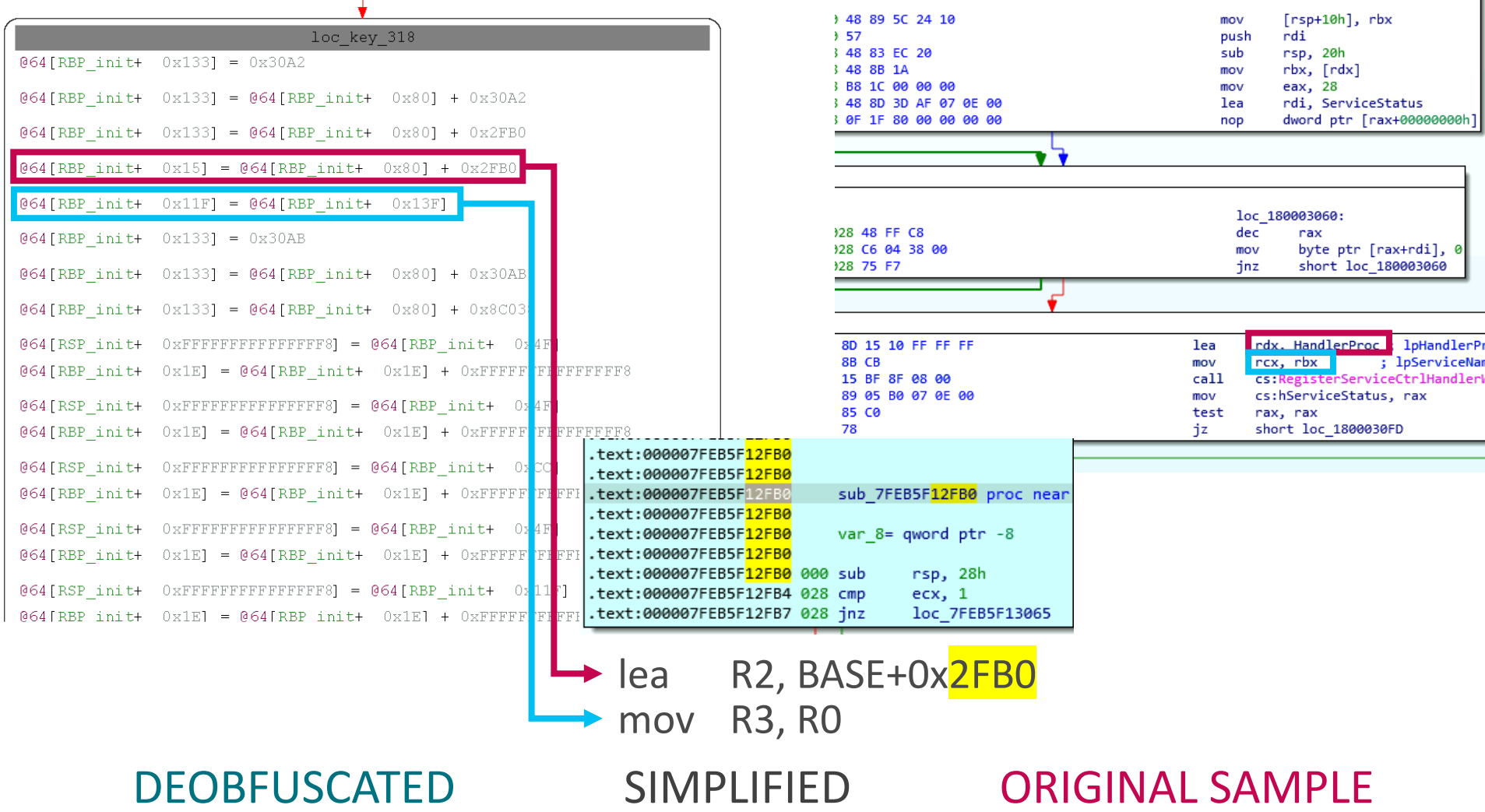
DEOBFUSCATED

ORIGINAL SAMPLE



DEOBFUSCATED

ORIGINAL SAMPLE


loc_key_318
@64[RBP_init+ 0x133] = 0x30A2
@64[RBP_init+ 0x133] = @64[RBP_init+ 0x80] + 0x30A2
@64[RBP_init+ 0x133] = @64[RBP_init+ 0x80] + 0x2FB0
@64[RBP_init+ 0x15] = @64[RBP_init+ 0x80] + 0x2FB0
@64[RBP_init+ 0x11F] = @64[RBP_init+ 0x13F]
@64[RBP_init+ 0x133] = 0x30AB
@64[RBP_init+ 0x133] = @64[RBP_init+ 0x80] + 0x30AB
@64[RBP_init+ 0x133] = @64[RBP_init+ 0x80] + 0x8C03
@64[RSP_init+ 0xFFFFFFFFFFFFFFF8] = @64[RBP_init+ 0x4F]
@64[RBP_init+ 0x1E] = @64[RBP_init+ 0x1E] + 0xFFFFFFFFFFFFFFF8
@64[RSP_init+ 0xFFFFFFFFFFFFFFF8] = @64[RBP_init+ 0x4F]
@64[RBP_init+ 0x1E] = @64[RBP_init+ 0x1E] + 0xFFFFFFFFFFFFFFF8
@64[RSP_init+ 0xFFFFFFFFFFFFFFF8] = @64[RBP_init+ 0xCC]
@64[RBP_init+ 0x1E] = @64[RBP_init+ 0x1E] + 0xFFFFFFFFFFFFFFF8
@64[RSP_init+ 0xFFFFFFFFFFFFFFF8] = @64[RBP_init+ 0x4F]
@64[RBP_init+ 0x1E] = @64[RBP_init+ 0x1E] + 0xFFFFFFFFFFFFFFF8
@64[RSP_init+ 0xFFFFFFFFFFFFFFF8] = @64[RBP_init+ 0x11F]
@64[RBP_init+ 0x1E] = @64[RBP_init+ 0x1E] + 0xFFFFFFFFFFFFFFF8
48 89 5C 24 10 mov [rsp+10h], rbx
57 push rdi
48 83 EC 20 sub rsp, 20h
48 8B 1A mov rbx, [rdx]
B8 1C 00 00 00 mov eax, 28
48 8D 3D AF 07 0E 00 lea rdi, ServiceStatus
0F 1F 80 00 00 00 00 nop dword ptr [rax+00000000h]
loc_180003060:
28 48 FF C8 dec rax
28 C6 04 38 00 mov byte ptr [rax+rdi], 0
28 75 F7 jnz short loc_180003060
8D 15 10 FF FF FF lea rdx, HandlerProc ; lpHandlerPr
8B CB mov rcx, rbx ; lpServiceNam
15 BF 8F 08 00 call cs:RegisterServiceCtrlHandlerW
89 05 B0 07 0E 00 mov cs:hServiceStatus, rax
85 C0 test rax, rax
78 jz short loc_1800030FD
.text:000007FEB5F12FB0
.text:000007FEB5F12FB0
.text:000007FEB5F12FB0 sub_7FEB5F12FB0 proc near
.text:000007FEB5F12FB0
.text:000007FEB5F12FB0 var_8= qword ptr -8
.text:000007FEB5F12FB0
.text:000007FEB5F12FB0 000 sub rsp, 28h
.text:000007FEB5F12FB4 028 cmp ecx, 1
.text:000007FEB5F12FB7 028 jnz loc_7FEB5F13065
lea R2, BASE+0x2FB0
mov R3, R0
DEOBFUSCATED
SIMPLIFIED
ORIGINAL SAMPLE

```
loc_key_318
@64[RBP_init+  0x133] = 0x30A2
@64[RBP_init+  0x133] = @64[RBP_init+  0x80] + 0x30A2
@64[RBP_init+  0x133] = @64[RBP_init+  0x80] + 0x2FB0
@64[RBP_init+  0x15] = @64[RBP_init+  0x80] + 0x2FB0
@64[RBP_init+  0x11F] = @64[RBP_init+  0x13F]
@64[RBP_init+  0x133] = 0x30AB
@64[RBP_init+  0x133] = @64[RBP_init+  0x80] + 0x30AB
@64[RBP_init+  0x133] = @64[RBP_init+  0x80] + 0x8C038
@64[RSP_init+  0xFFFFFFFFFFFFFFF8] = @64[RBP_init+  0x4F]
@64[RBP_init+  0x1E] = @64[RBP_init+  0x1E] + 0xFFFFFFFFFFFFFFF8
@64[RSP_init+  0xFFFFFFFFFFFFFFF8] = @64[RBP_init+  0x4F]
@64[RBP_init+  0x1E] = @64[RBP_init+  0x1E] + 0xFFFFFFFFFFFFFFF8
@64[RSP_init+  0xFFFFFFFFFFFFFFF8] = @64[RBP_init+  0xCC]
@64[RBP_init+  0x1E] = @64[RBP_init+  0x1E] + 0xFFFFFFFFFFFFFFF8
@64[RSP_init+  0xFFFFFFFFFFFFFFF8] = @64[RBP_init+  0x4F]
@64[RBP_init+  0x1E] = @64[RBP_init+  0x1E] + 0xFFFFFFFFFFFFFFF8
@64[RSP_init+  0xFFFFFFFFFFFFFFF8] = @64
@64[RBP_init+  0x1E] = @64[RBP_init+  0x
```

```
48 89 5C 24 10          mov     [rsp+10h], rbx
57                      push    rdi
48 83 EC 20             sub     rsp, 20h
48 8B 1A                mov     rbx, [rdx]
B8 1C 00 00 00          mov     eax, 28
48 8D 3D AF 07 0E 00    lea     rdi, ServiceStatus
0F 1F 80 00 00 00 00    nop     dword ptr [rax+00000000h]
```

```
                                loc_180003060:
28 48 FF C8                     dec     rax
28 C6 04 38 00                  mov     byte ptr [rax+rdi], 0
28 75 F7                        jnz     short loc_180003060
```

```
8D 15 10 FF FF FF       lea     rdx, HandlerProc ; lpHandlerPr
8B CB                   mov     rcx, rbx         ; lpServiceNa
15 BF 8F 08 00          call    cs:RegisterServiceCtrlHandlerW
89 05 B0 07 0E 00       mov     cs:hServiceStatus, rax
85 C0                   test    rax, rax
78                      jz      short loc_1800030FD
```

lea Rdest, BASE+0x8C038

```
.idata:000007FEB5F9C038     ; SERVICE_STATUS_HANDLE __stdcall RegisterServiceCtrlH
.idata:000007FEB5F9C038     RegisterServiceCtrlHandlerW dq offset advapi32_Registe
.idata:000007FEB5F9C040     db 0, 0, 0, 0, 0, 0, 0, 0
```

DEOBFUSCATED

SIMPLIFIED

ORIGINAL SAMPLE

```
@64[RBP_init+  0x133] = @64[RBP_init+  0x80] + 0x30AB
@64[RBP_init+  0x133] = @64[RBP_init+  0x80] + 0x8C038
@64[RSP_init+  0xFFFFFFFFFFFFFFF8] = @64[RBP_init+  0x4F]
@64[RBP_init+  0x1E] = @64[RBP_init+  0x1E] + 0xFFFFFFFFFFFFFFF8
```

lea Rdest, BASE+0x8C038

```
.idata:000007FEB5F9C038     ; SERVICE_STATUS_HANDLE __stdcall RegisterServiceCtrlH
.idata:000007FEB5F9C038     RegisterServiceCtrlHandlerW dq offset advapi32_Registe
.idata:000007FEB5F9C040     db 0, 0, 0, 0, 0, 0, 0, 0
@64[RSP_init+  0xFFFFFFFFFFFFFFF8] = @64[RBP_init+  0x4F]
@64[RBP_init+  0x1E] = @64[RBP_init+  0x1E] + 0xFFFFFFFFFFFFFFF8
...
@64[RBP_init+  0x1E] = @64[RBP_init+  0x1E] + 0xFFFFFFFFFFFFFFF8
@64[RSP_init+  0xFFFFFFFFFFFFFFF8] = @64[RBP_init+  0x58]
@64[RBP_init+  0x1E] = @64[RBP_init+  0x1E] + 0xFFFFFFFFFFFFFFF8
@64[RSP_init+  0x88] = @64[RBP_init+  0x80] + 0x21EA5C
@32[RBP_init+  0x12
RAX.0=  @64[RSP_ini
RSP.0=  RSP_init+
exception flags= ( @32[RSP_init+  0x78])
IRDst=  @64[@64[RBP_init+  0x133]]
```

RETaddr = &vm_pre_initX()

jmp Rdest

DEOBFUSCATED

```
48 89 5C 24 10          mov     [rsp+10h], rbx
57                      push    rdi
48 83 EC 20             sub     rsp, 20h
48 8B 1A                mov     rbx, [rdx]
B8 1C 00 00 00          mov     eax, 28
48 8D 3D AF 07 0E 00    lea     rdi, ServiceStatus
0F 1F 80 00 00 00 00    nop     dword ptr [rax+00000000h]

                        loc_180003060:
28 48 FF C8             dec     rax
28 C6 04 38 00          mov     byte ptr [rax+rdi], 0
28 75 F7                jnz     short loc_180003060

8D 15 10 FF FF FF       lea     rdx, HandlerProc ; lpHandlerPr
8B CB                   mov     rcx, rbx         ; lpServiceMa
15 BF 8F 08 00          call    cs:RegisterServiceCtrlHandlerW
89 05 B0 07 0E 00       mov     cs:hServiceStatus, rax
85 C0                   test    rax, rax
78                      jz      short loc_1800030FD
```

ORIGINAL SAMPLE

Limitations

- Symbolic execution cannot process

unbounded loops

- Instructions using such loops need to be addressed

by other means

Takeaway

Symbolic execution can help **devirtualize** advanced unknown VMs in a **reasonable time** if we treat the right values as concrete

Links

You can read the whitepaper at

WeLiveSecurity.com

Full source code is available in

ESET git repository

Digital Security
Progress. Protected.

Questions?

Vladislav Hrčka

Malware Researcher

vladislav.hrcka@eset.com | @HrckaVladislav

www.eset.com | www.welivesecurity.com | @ESETresearch