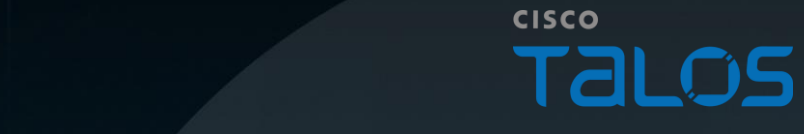


Architecture Analysis of VMProtect 3.8

Holger Unterbrink

Who am I ?

Holger Unterbrink

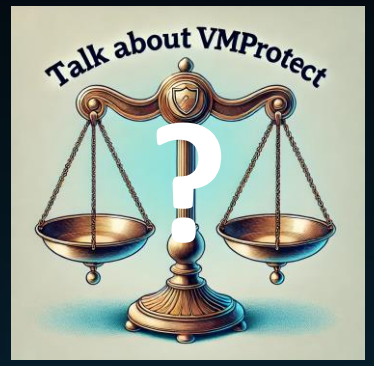
@hunterbr72

Technical leader - Security Researcher at Cisco Talos

Malware Analysis, Threat hunting, Tool development and lately Big Data/Machine Learning
Won the IDA plugin contest 2020 with the Dynamic Data Resolver Instrumentation Plugin

Germany

Disclaimer



Question: What is the goal of the talk ?

Answer: Demystify the Virtual Machine of VMP. Get a starting point for own research.

Question: Can I crack VMProtect protected real world samples after the talk ?

Answer: Most likely, NO. Depends on the time you have and the VMP settings used.

Question: Will I be able to understand how the Virtual Machine work after the talk ?

Answer: Yes, I hope so.

Question: Why should I spend time on VMProtect ?

Answer: VMProtect 3.5 Source code leaked and VMProtect 3.8 binary, too. Latest version on 27th June is 3.8.8

CISCO TALOS

Other research on VMProtect

Inside VMProtect (Samuel Chevet)
https://webtv.univ-lille.fr/video/7566/inside-vmprotect (**2015)**

VMProtect 2 - Detailed Analysis of the Virtual Machine Architecture
https://blog.back.engineering/17/05/2021/ (2021)
https://blog.back.engineering/21/06/2021/

VMProtect 3.5: Virtualization-Based Software Obfuscation
https://www.mitchellzakocs.com/blog/vmprotect3 (2021)

VMProtect 3.x Devirtualization
https://github.com/JonathanSalwan/VMProtect-devirtualization (~2022)

Other research on VMProtect

How To Unpack VMProtect Malware
(Paid - VMProtect 3 – Targets only misconfigured VMP protected malware)
https://www.patreon.com/posts/how-to-unpack-1-61634765

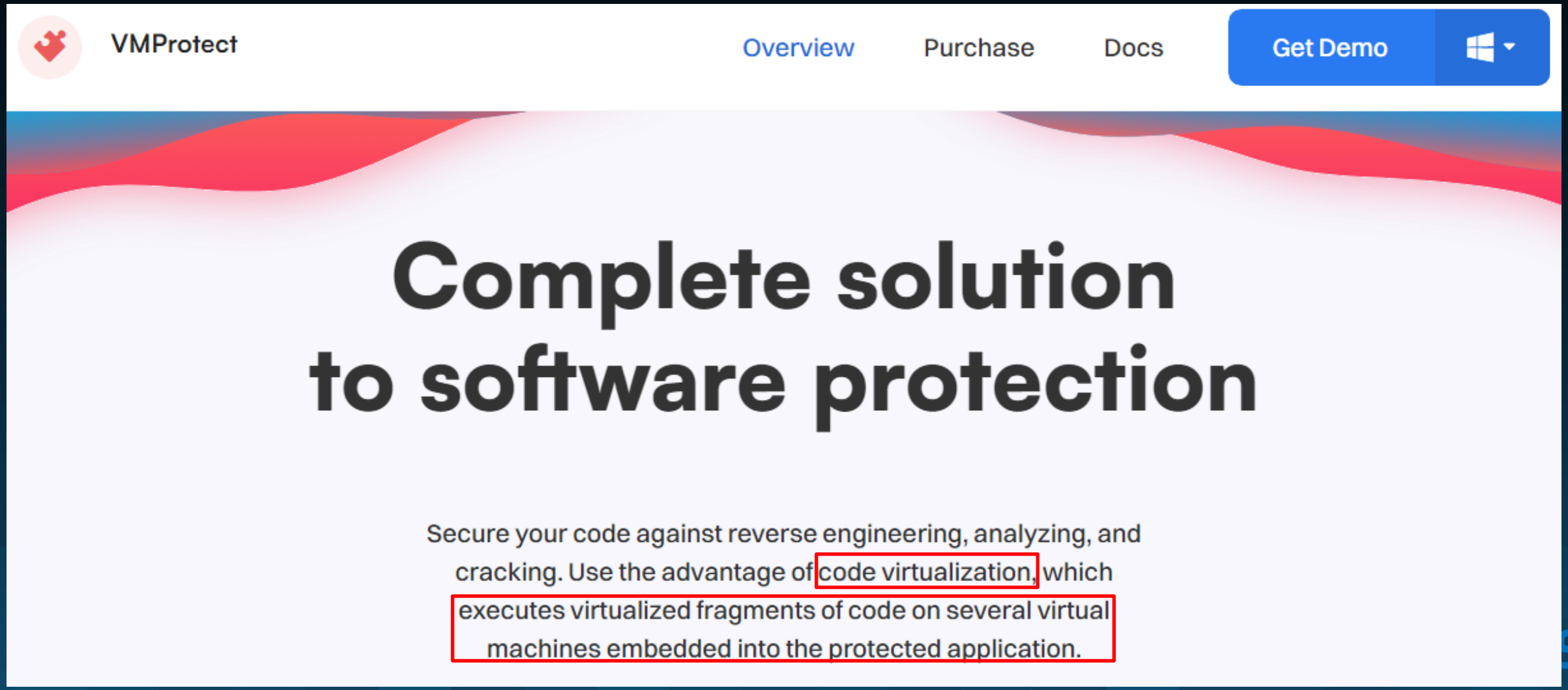
AllThingsIDA's Poor man's guide to de-obfuscating VMProtect's 32bit import obfuscation (2023)
https://www.youtube.com/watch?v=ZhQUbjFbsTw
https://www.youtube.com/watch?v=uxOVbG-azIA
https://www.youtube.com/watch?v=GvWSa6HTlNY

Analyzing Mutation-Coded - VM Protect and Alcatraz (2024)
https://keowu.re/posts/Analyzing-Mutation-Coded-VM-Protect-and-Alcatraz-English/

Intro

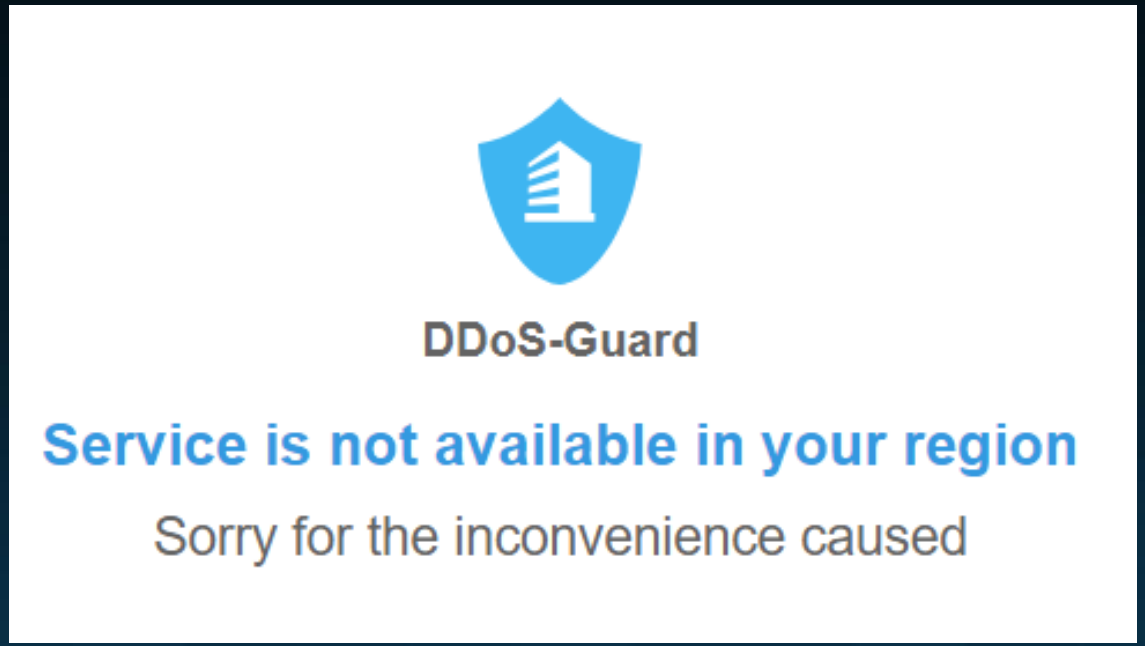
VMProtect

VPN to Singapore



VMProtect

Looks like they are also effected by the recent geo political issues – No Love for the West anymore



Recent IP	IP Country/Region	Prefix	ASN	Network Owner Description
185.178.208.164	RU	185.178.208.0/24	AS57724	DDOS-GUARD, RU 86400

https://vmpsoft.com/

Pricelist

For Individuals | For Companies

Lite

+1 year of free updates

$ 239.00

Obfuscation features:

- Code virtualization

Professional

+1 year of free updates

$ 349.00

All the features in Lite, and more:

Ultimate

+1 year of free updates

$ 699.00

All the features in Professional, and more:

Updates Renewal

1 additional year of updates

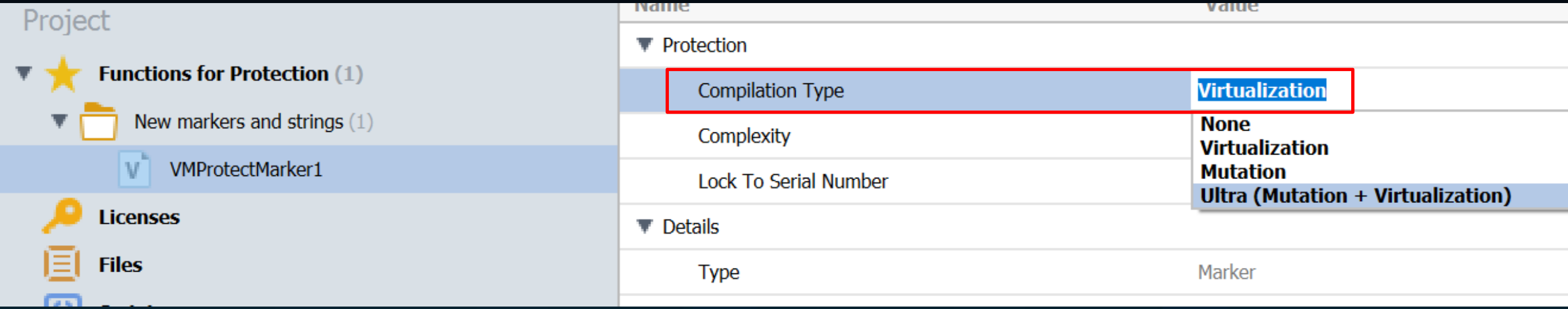
$ 139.00

Renewal of updates for existing license for one

How does it work ?

VMProtect 3.8 GUI – Compilation Type

Translation of CISC code to a RISC Virtual Machine



Original Intel code Complex Instruction Set Computing (CISC) instruction will be translated to multiple virtualized instructions

VMProtect VM uses Reduced Instruction Set Computing (RISC)

e.g. inc eax → v_add(v_reg, 1)

e.g. dec eax → v_add(v_reg, -1)

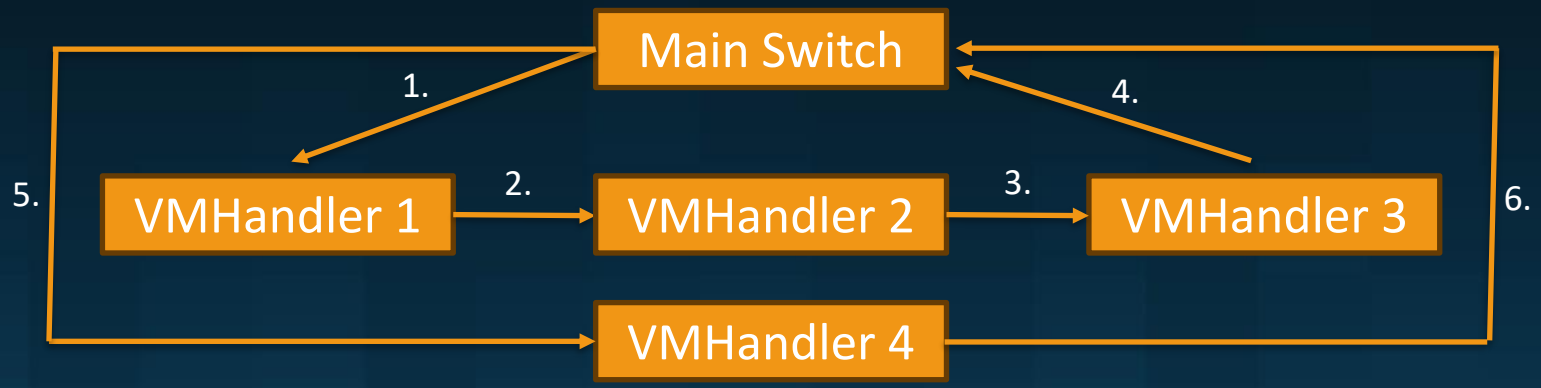
e.g. lea ecx, [esp + ebx * 8 + 23] → … multiple RISC instructions

```
bool IntelFunction::Compile(const CompileContext &ctx)
{
    switch (compilation_type()) {
    case ctMutation: //bian yi
        CompileToNative(ctx);
        break;
    case ctVirtualization: //xu ni hua
        CompileToVM(ctx);
        break;
    case ctUltra://xu ni hua + bian yi
        Mutate(ctx, true);
        CompileToVM(ctx);
        break;
    default:
        return false;
    }
```

Order

Very very simplified architecture

More or less a chain of VMHandlers

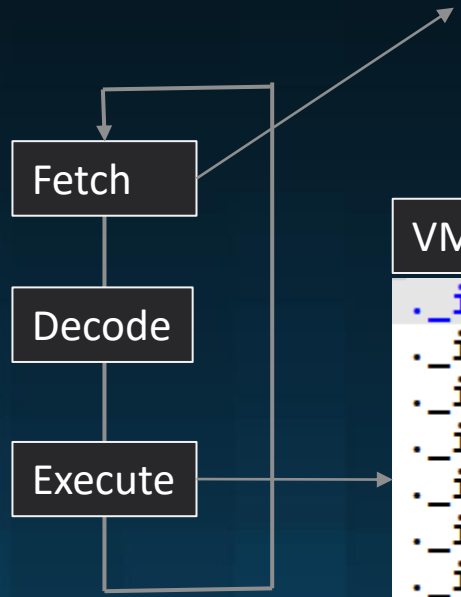


Very simplified VM Architecture – Stack Machine

But a bit more technical…

VMHandlerTable

```
:0000000140010774                dd 41AEA8DFh            ; 1. VMHandler enc. offset
:0000000140010778                dd 93133Ch              ; 0. VMHandler enc. offset
:000000014001077C ; ---------------------------------------------------------------------------
```



VMHandler e.g. v_add()

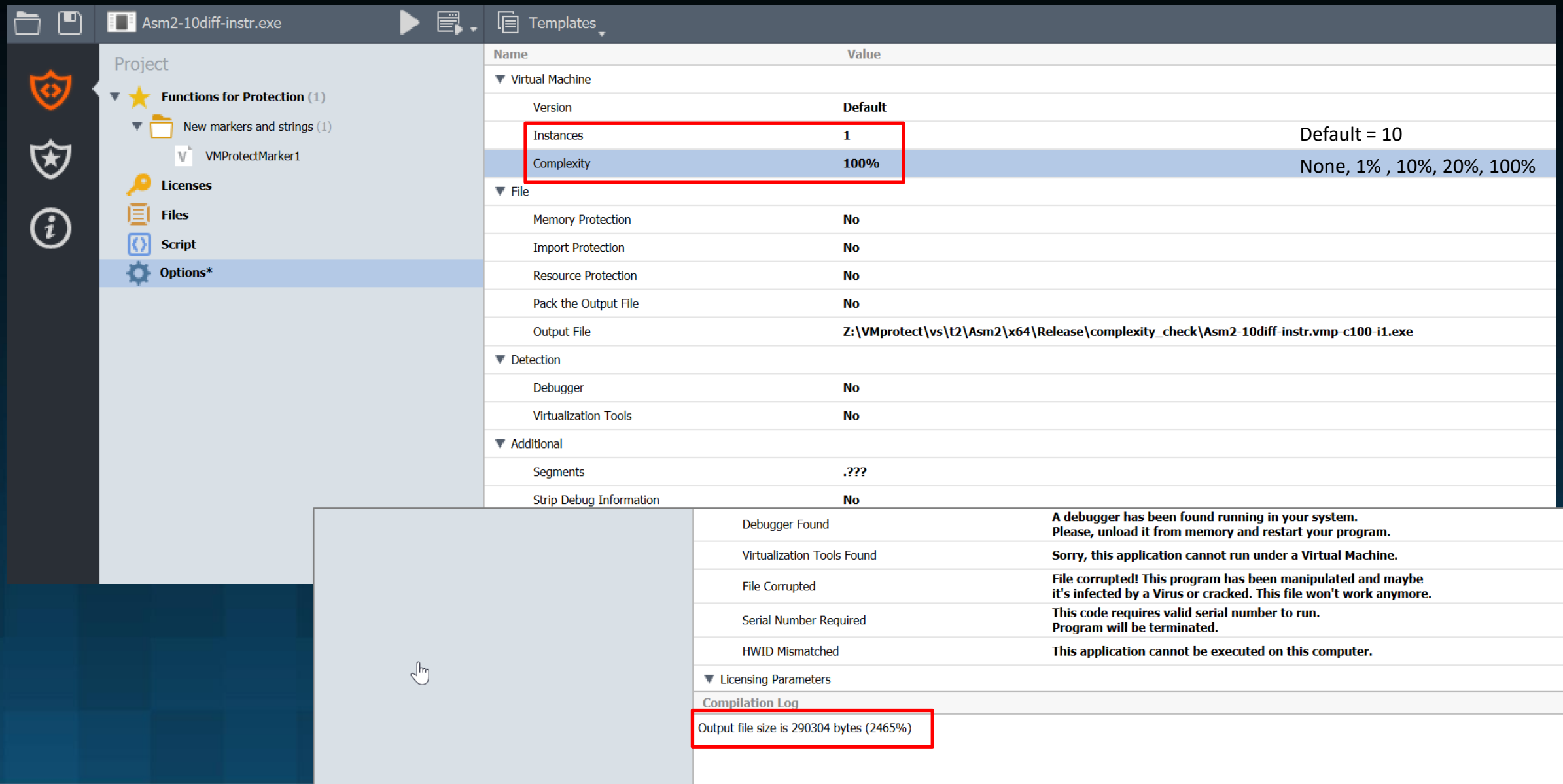
```
._i_:00000001400311D3    lea     r8, loc_1400311D3 ; VMHandler 0
._i_:00000001400311DA    sub     rbx, 4
._i_:00000001400311E1    mov     ecx, 0DF2051BBh
._i_:00000001400311E6    movsx   eax, cl
._i_:00000001400311E9    mov     ecx, [rbx]        ; [rbx]=[140010778]
._i_:00000001400311EC    mov     edx, eax
._i_:00000001400311EE    lea     rsi, ds:0FFFFFFFF8A044781h[rax*2]
._i_:00000001400311F6    xor     ecx, ebp
._i_:00000001400311F8    imul    rax
```

CISCO TALOS

Virtual Machine Protection

- **Instances** - this options allows to specify the number of the virtual machine copies (the default value is 10). Each virtual machine will have unique set of properties (different registers positions, different bytecode direction, different handlers of commands, etc.) that makes harder the analysis and hacking of virtualized code.
- **Complexity** - this options allows to specify the probability of creating complex handlers (consisting of several simple handlers) inside the virtual machine. This option also greatly complicates the analysis and hacking of virtualized code. As the complexity increases, the size of the protected file also increases.

VMProtect 3.8 GUI – Complexity and Instances



Licensed vs. Unlicensed Versions

```
    if
#ifdef DEMO
        (true)
#else
        (ctx.options.flags & cpUnregisteredVersion)
#endif
    {
        crypt_registr_ = (ctx.options.flags & cpEncryptBytecode) ? regEBX : 0;
        pcode_registr_ = regESI;
        stack_registr_ = regEBP;
        if (type_ == vtAdvanced)
            jmp_registr_ = regEDI;
        else if (cpu_address_size == osQWord)
            jmp_registr_ = regR11;
        else
            jmp_registr_ = 0;
    }
    else {
        IntelRegistrList work_registr_list;
        work_registr_list.push_back(regEBX);
        work_registr_list.push_back(regEBP);
        work_registr_list.push_back(regESI);
        work_registr_list.push_back(regEDI);
        if (cpu_address_size == osQWord) {
            for (i = 8; i < 16; i++) {
                work_registr_list.push_back((uint8_t)i);
            }
        }
        work_registr_list.remove(wrong_registr_list);

        crypt_registr_ = 0;
        if (ctx.options.flags & cpEncryptBytecode) {
            if (cpu_address_size == osDWord) {
                crypt_registr_ = regEBX;
                work_registr_list.remove(crypt_registr_);
            }
            else
                crypt_registr_ = work_registr_list.GetRandom();
        }
        pcode_registr_ = work_registr_list.GetRandom();
        stack_registr_ = work_registr_list.GetRandom();
        jmp_registr_ = (type_ == vtAdvanced || cpu_address_size == osQWord) ? work_registr_list.GetRandom() : 0;
```

Demo or unregistered version

Licensed version with random register assignment

Main-Function after protection

```cpp
#include <iostream>
#include "VMProtectSDK.h"

int main()
{
    VMProtectBegin("Test marker");
    printf("Hello Protection\n");
    VMProtectEnd();
    printf("Hello\n");
    exit(0);
}
```

```
.text:0000000140001000 main            proc near               ; CODE XREF: __scrt_common_main_
.text:0000000140001000
.text:0000000140001000 arg_8           = qword ptr  10h
.text:0000000140001000
.text:0000000140001000                 call    _CRT_INIT
.text:0000000140001005                 push    rbp
.text:0000000140001006                 mov     rbp, rsp
.text:0000000140001009                 sub     rsp, 20h
.text:000000014000100D                 push    rax
.text:000000014000100E                 push    rcx
.text:000000014000100F                 call    loc_140001044
.text:0000000140001014
.text:0000000140001014 loc_140001014:                          ; DATA XREF: sub_140046423+1E↓r
.text:0000000140001014                 xchg    ah, [rbx+0]
.text:0000000140001017                 add     [rax-4Fh], ch
.text:000000014000101A                 in      al, dx
.text:000000014000101B                 sub     eax, 0F7489C7Fh
.text:0000000140001020                 and     al, 8
.text:0000000140001020 ; ---------------------------------------------------------------------------
.text:0000000140001023                 db 2Fh, 0D5h, 34h, 0F8h, 0Fh
.text:0000000140001028 ; ---------------------------------------------------------------------------
.text:0000000140001028                 xchg    eax, edx
.text:0000000140001029                 and     al, 8
.text:000000014000102C                 mov     [rsp+arg_8], 376D405Ch
.text:0000000140001035                 push    qword ptr [rsp+0]
.text:0000000140001039                 popfq
.text:000000014000103A                 lea     rsp, [rsp+10h]
.text:000000014000103F                 call    loc_14001E9F0
.text:0000000140001044
.text:0000000140001044 loc_140001044:                          ; CODE XREF: main+F↑p
.text:0000000140001044                 push    rbx
.text:0000000140001045                 call    loc_140012426
.text:000000014000104A                 stosb
.text:000000014000104B
.text:000000014000104B loc_14000104B:                          ; DATA XREF: .vmH:00000001400466
.text:000000014000104B                 cmp     [rcx], al
.text:000000014000104B ; ---------------------------------------------------------------------------
.text:000000014000104D                 db 3 dup(0)
.text:0000000140001050                 dq 0A50D8D485859B7C1h, 10E800002Fh
.text:0000000140001060 ; ---------------------------------------------------------------------------
.text:0000000140001060                 xor     ecx, ecx
.text:0000000140001062                 call    $+5
.text:0000000140001062 main            endp ; sp-analysis failed
.text:0000000140001062
```

Main-Function after protection

Anti-Disasm not fixed

```
.text:0000000140001000 main            proc near               ; CODE XREF: __scrt_common_main_seh+107
.text:0000000140001000
.text:0000000140001000 arg_8           = qword ptr  10h
.text:0000000140001000
.text:0000000140001000                 call    _CRT_INIT
.text:0000000140001005                 push    rbp
.text:0000000140001006                 mov     rbp, rsp
.text:0000000140001009                 sub     rsp, 20h
.text:000000014000100D                 push    rax
.text:000000014000100E                 push    rcx
.text:000000014000100F                 call    loc_140001044
.text:0000000140001014
.text:0000000140001014 loc_140001014:                          ; DATA XREF: sub_140046423+1E↓r
.text:0000000140001014                 xchg    ah, [rbx+0]
.text:0000000140001017                 add     [rax-4Fh], ch
.text:000000014000101A                 in      al, dx
.text:000000014000101B                 sub     eax, 0F7489C7Fh
.text:0000000140001020                 and     al, 8
.text:0000000140001020 ; ---------------------------------------------------------------------------
.text:0000000140001023                 db 2Fh, 0D5h, 34h, 0F8h, 0Fh
.text:0000000140001028 ; ---------------------------------------------------------------------------
.text:0000000140001028                 xchg    eax, edx
.text:0000000140001029                 and     al, 8
.text:000000014000102C                 mov     [rsp+arg_8], 376D405Ch
.text:0000000140001035                 push    qword ptr [rsp+0]
.text:0000000140001039                 popfq
.text:000000014000103A                 lea     rsp, [rsp+10h]
.text:000000014000103F                 call    loc_14001E9F0
.text:0000000140001044
.text:0000000140001044 loc_140001044:                          ; CODE XREF: main+F↑p
.text:0000000140001044                 push    rbx
.text:0000000140001045                 call    loc_140012426
.text:000000014000104A                 stosb
.text:000000014000104B
.text:000000014000104B loc_14000104B:                          ; DATA XREF: .vmH:00000001400466B0↓o
.text:000000014000104B                 cmp     [rcx], al
.text:000000014000104B ; ---------------------------------------------------------------------------
.text:000000014000104D                 db 3 dup(0)
.text:0000000140001050                 dq 0A50D8D485859B7C1h, 10E800002Fh
.text:0000000140001060 ; ---------------------------------------------------------------------------
.text:0000000140001060                 xor     ecx, ecx
.text:0000000140001062                 call    $+5
.text:0000000140001062 main            endp ; sp-analysis failed
.text:0000000140001062
```

Anti-Disasm fixed

```
.text:0000000140001000 main            proc near               ; CODE XREF: __scrt_common_main_seh+107↓p
.text:0000000140001000
.text:0000000140001000 arg_8           = qword ptr  10h
.text:0000000140001000
.text:0000000140001000                 call    _CRT_INIT
.text:0000000140001005                 push    rbp
.text:0000000140001006                 mov     rbp, rsp
.text:0000000140001009                 sub     rsp, 20h
.text:000000014000100D                 push    rax
.text:000000014000100E                 push    rcx
.text:000000014000100F                 call    loc_140001044
.text:000000014000100F ; ---------------------------------------------------------------------------
.text:0000000140001014 unk_140001014   db  86h                 ; DATA XREF: sub_140046423+1E↓r
.text:0000000140001015                 db  63h ; c
.text:0000000140001016                 db    0
.text:0000000140001017                 db    0
.text:0000000140001018 ; ---------------------------------------------------------------------------
.text:0000000140001018                 push    7F2DECB1h
.text:000000014000101D                 pushfq
.text:000000014000101E                 test    qword ptr [rsp+8], 0FFFFFFFFF834D52Fh
.text:0000000140001027                 setb    byte ptr [rsp+8]
.text:000000014000102C                 mov     qword ptr [rsp+10h], 376D405Ch
.text:0000000140001035                 push    qword ptr [rsp+0]
.text:0000000140001039                 popfq
.text:000000014000103A                 lea     rsp, [rsp+10h]
.text:000000014000103F                 call    loc_14001E9F0
.text:0000000140001044
.text:0000000140001044 loc_140001044:                          ; CODE XREF: main+F↑p
.text:0000000140001044                 push    rbx
.text:0000000140001045                 call    loc_140012426
.text:0000000140001045 ; ---------------------------------------------------------------------------
.text:000000014000104A                 db 0AAh
.text:000000014000104B                 db  38h ; 8
.text:000000014000104C                 db    1                ; DATA XREF: .vmH:00000001400466B0↓o
.text:000000014000104D                 db    0
.text:000000014000104E                 db    0
.text:000000014000104F                 db    0
.text:0000000140001050                 db 0C1h
.text:0000000140001051                 db 0B7h
.text:0000000140001052 ; ---------------------------------------------------------------------------
.text:0000000140001052                 pop     rcx
.text:0000000140001053                 pop     rax
.text:0000000140001054                 lea     rcx, fmtStr      ; "Hello\n"
.text:000000014000105B                 call    printf
.text:0000000140001060                 xor     ecx, ecx
.text:0000000140001062                 call    $+5
.text:0000000140001062 main            endp ; sp-analysis failed
```

call/jmp to VMP Section

Code trace

Asm2_single_instr-c1-i1.vmp.e

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00000F20	00001000	00001000	00000400	00000000	00000000	0000	0000	60000020
.rdata	00001056	00002000	00001200	00001400	00000000	00000000	0000	0000	40000040
.data	00000140	00004000	00000200	00002600	00000000	00000000	0000	0000	C0000040
.pdata	000001D4	00005000	00000200	00002800	00000000	00000000	0000	0000	40000040
.4O!	00037CEC	00006000	00037E00	00002A00	00000000	00000000	0000	0000	68000020
.reloc	00000048	0003E000	00000200	0003A800	00000000	00000000	0000	0000	40000040
.rsrc	000001D5	0003F000	00000200	0003AA00	00000000	00000000	0000	0000	40000040

".vmp0"

PE64

Operation system: Windows(Vista)[AMD64, 64-bit, Console]

Linker: Microsoft Linker(14.36.33134)

Compiler: Microsoft Visual C/C++(19.36.33030)[C++]

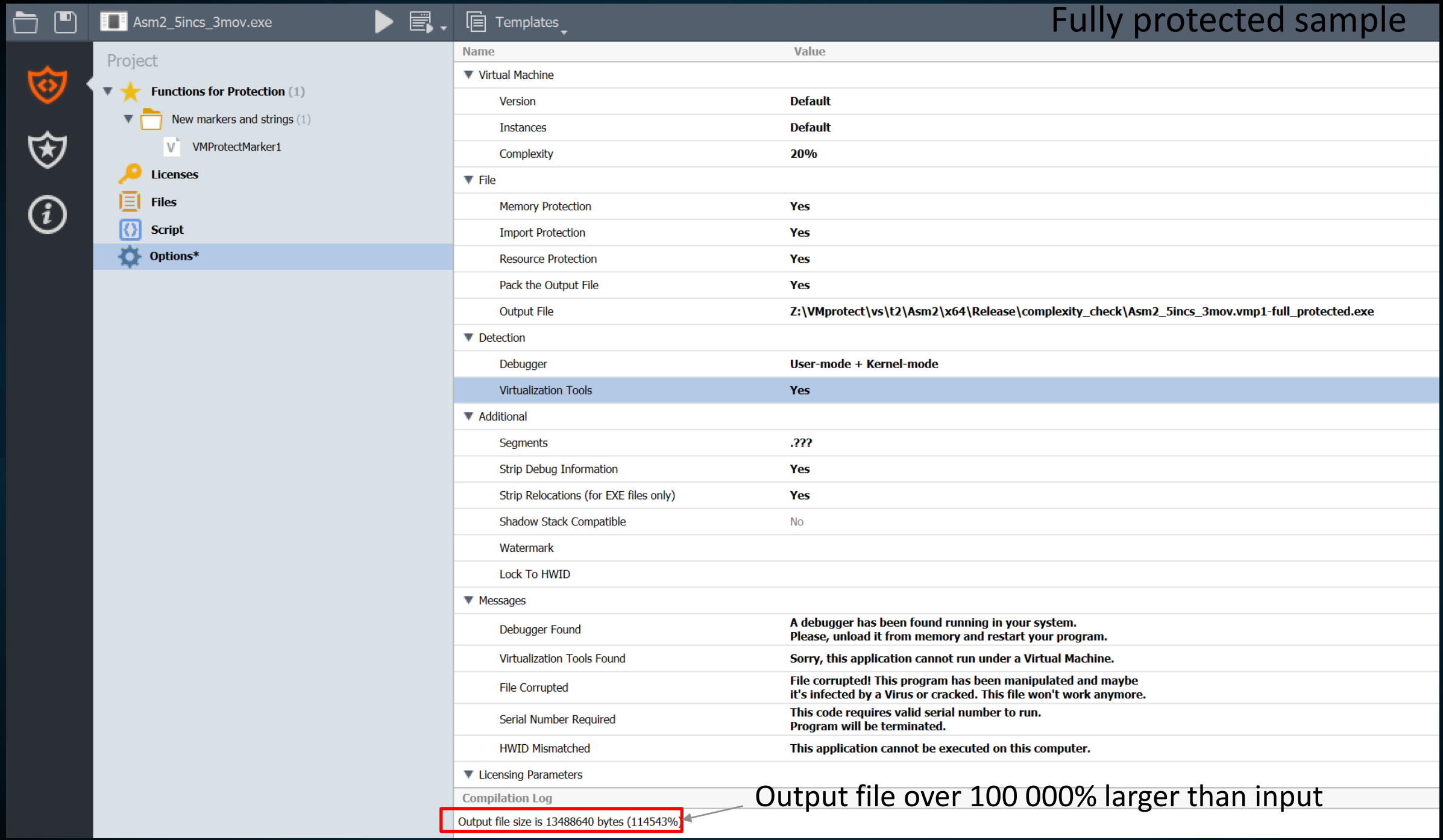
Language: C/C++

Tool: Visual Studio(2022 version 17.6)

Packer: Packer detected(Heuristic)[Section 3 (".pdata") compressed]

Detect it easy !

... but things can be different, depending on the VMP settings!


Asm2_5incs_3mov.exe
Templates
Fully protected sample
Project
Functions for Protection (1)
New markers and strings (1)
VMProtectMarker1
Licenses
Files
Script
Options*
Name
Value
Virtual Machine
Version
Default
Instances
Default
Complexity
20%
File
Memory Protection
Yes
Import Protection
Yes
Resource Protection
Yes
Pack the Output File
Yes
Output File
Z:\VMprotect\vs\t2\Asm2\x64\Release\complexity_check\Asm2_5incs_3mov.vmp1-full_protected.exe
Detection
Debugger
User-mode + Kernel-mode
Virtualization Tools
Yes
Additional
Segments
.???
Strip Debug Information
Yes
Strip Relocations (for EXE files only)
Yes
Shadow Stack Compatible
No
Watermark
Lock To HWID
Messages
Debugger Found
A debugger has been found running in your system.
Please, unload it from memory and restart your program.
Virtualization Tools Found
Sorry, this application cannot run under a Virtual Machine.
File Corrupted
File corrupted! This program has been manipulated and maybe
it's infected by a Virus or cracked. This file won't work anymore.
Serial Number Required
This code requires valid serial number to run.
Program will be terminated.
HWID Mismatched
This application cannot be executed on this computer.
Licensing Parameters
Compilation Log
Output file size is 13488640 bytes (114543%)
Output file over 100 000% larger than input

Multiple random sections

Asm2_5incs_3mov.vmp1-full_pro

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00000F60	00001000	00000000	00000000	00000000	00000000	0000	0000	60000020
.rdata	00001056	00002000	00000000	00000000	00000000	00000000	0000	0000	40000040
.data	00000140	00004000	00000000	00000000	00000000	00000000	0000	0000	C0000040
.pdata	000001D4	00005000	00000000	00000000	00000000	00000000	0000	0000	40000040
.oS"	0079EB12	00006000	00000000	00000000	00000000	00000000	0000	0000	60000020
.fTJ	00000870	007A5000	00000A00	00000400	00000000	00000000	0000	0000	C0000040
.!PG	00CDC0DC	007A6000	00CDC200	00000E00	00000000	00000000	0000	0000	68000060
.rsrc	000001D5	01483000	00000200	00CDD000	00000000	00000000	0000	0000	40000040

Fully protected and packed sample

```
._PG:0000000140FE0DFE                   db  48h ; H
._PG:0000000140FE0DFF ; ---------------------------------------------------------------------------
._PG:0000000140FE0DFF
._PG:0000000140FE0DFF                   public start
._PG:0000000140FE0DFF start:
._PG:0000000140FE0DFF                   call    loc_140F295F0
._PG:0000000140FE0E04                   pop     rbp
._PG:0000000140FE0E05                   pop     rsp
._PG:0000000140FE0E06                   sub     [r15-6Ch], r12
._PG:0000000140FE0E06 ; ---------------------------------------------------------------------------
._PG:0000000140FE0E0A                   db  82h
```

- PE64
  - Operation system: Windows(Vista)[AMD64, 64-bit, Console]
  - Protector: VMProtect(new 36 jmp 12)[DS]
  - Protection: Generic(Heuristic)[Strange sections]
  - Packer: Packer detected(Heuristic)[High entropy + Section 6 (".!PG") compressed]

Latest
DIE version

```
.7A_:0000000140774736 ; ---------------------------------------------------------------
.7A_:0000000140774736
.7A_:0000000140774736                 public start
.7A_:0000000140774736 start:
.7A_:0000000140774736                 push    0FFFFFFFFE1BF151Fh
.7A_:000000014077473B                 pushfq
.7A_:000000014077473C                 xor     byte ptr [rsp+8], 32h
.7A_:0000000140774741                 push    3CBF67B9h
.7A_:0000000140774746                 sub     word ptr [rsp+0], 1F80h
.7A_:000000014077474D                 shl     qword ptr [rsp+10h], 7Eh
.7A_:0000000140774753                 mov     qword ptr [rsp+10h], 0FFFFFFFFF40D5F02h
.7A_:000000014077475C                 push    qword ptr [rsp+8]
.7A_:0000000140774760                 popfq
.7A_:0000000140774761                 lea     rsp, [rsp+10h]
.7A_:0000000140774766                 call    near ptr unk_140B77781
.7A_:000000014077476B                 cwde
```

Fully protected
But not packed
sample

▼ PE64

Operation system: Windows(Vista)[AMD64, 64-bit, Console]
Protector: VMProtect(new 25 jmp 9)[DS]
Protection: Generic(Heuristic)[Strange sections]
Packer: Packer detected(Heuristic)[High entropy + Section 3 (".pdata") compressed]

▼ PE64

Operation system: Windows(Vista)[AMD64, 64-bit, Console]
Packer: Packer detected(Heuristic)[High entropy + Section 3 (".pdata") compressed]

Not packed
No Anti-Dbg

How can we simplify the complexity ?

```
main proc
    call _CRT_INIT
    push rbp
    mov rbp, rsp
    sub rsp, 32

    push rax

    call VMProtectBegin
    mov rax, 0deadcafeh
    call VMProtectEnd

    pop rax
    lea rcx, fmtStr
    call printf

    xor ecx, ecx
    call exit
main endp

end
```

```
main proc
    call _CRT_INIT
    push rbp
    mov rbp, rsp
    sub rsp, 32

    push rax
    push rcx
    call VMProtectBegin

    mov  rax, 0deadbeefh
    mov  rax, 0dead1337h
    add rax, 0aaaaaaaah
    add eax, 0bbbbh
    add ax, 0cch
    rol rax, 1
    ror rax, 2
    nop
    nop
    nop
    not rax
    neg rax
    xor rax, 0ffffh
    inc rax
    dec rcx

    call VMProtectEnd
    pop rcx
    pop rax

    lea rcx, fmtStr
    call printf

    xor ecx, ecx
    call exit
main endp
```

```
main proc
    call _CRT_INIT
    push rbp
    mov rbp, rsp
    sub rsp, 32

    push rax

    call VMProtectBegin
    mov rax, 0deadcafeh
    inc rax
    inc rax
    inc rax
    inc rax
    inc rax
    inc rax
    mov  rax, 0deadbeefh
    inc rax
    inc rax
    inc rax
    inc rax
    inc rax
    mov  rax, 0dead1337h
    inc rax
    inc rax
    inc rax
    inc rax
    inc rax
    call VMProtectEnd

    pop rax
    lea rcx, fmtStr
    call printf

    xor ecx, ecx
    call exit
main endp
```

Sample code for testing

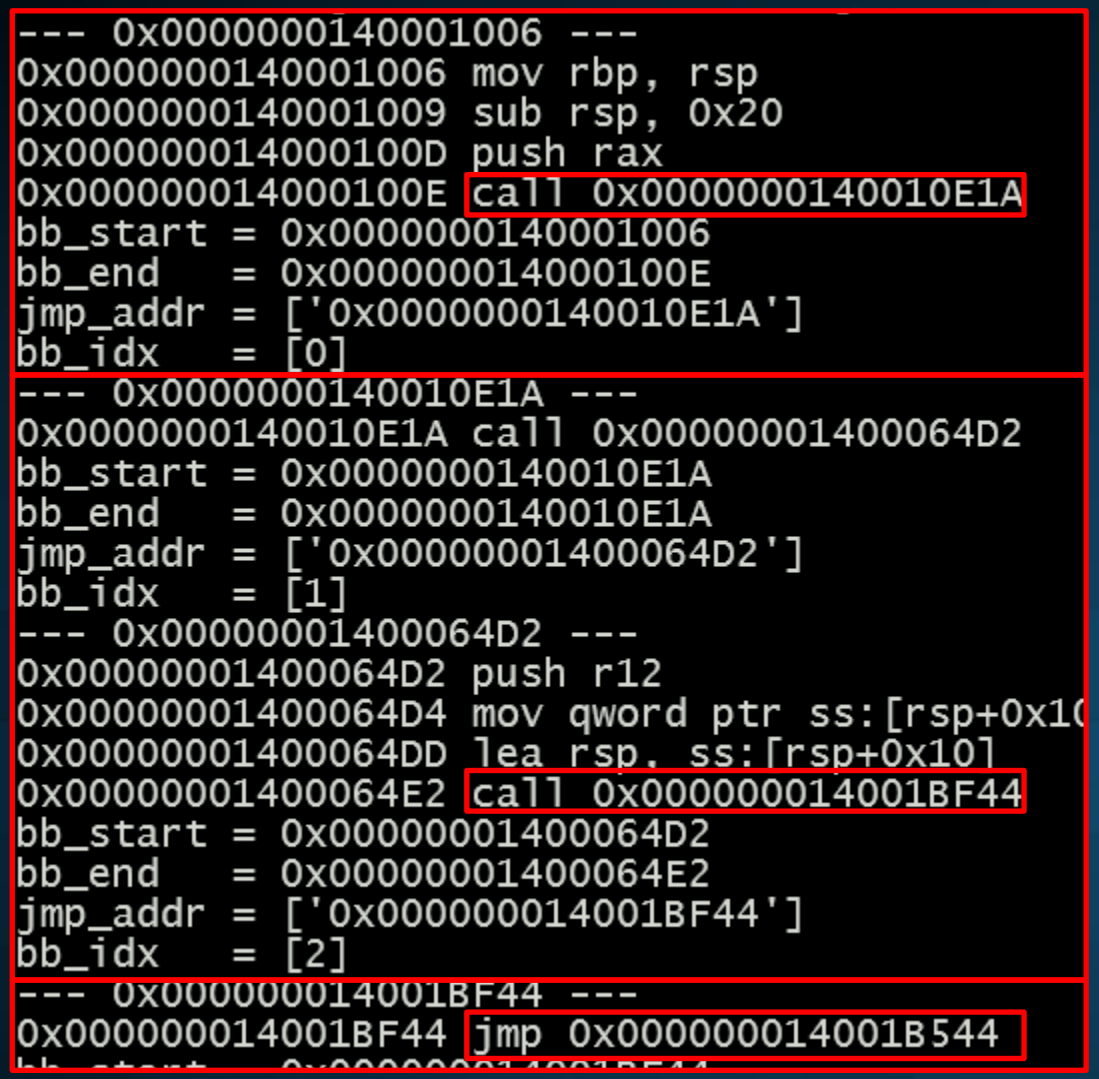
Control Flow Graphs to the rescue ….

```
$ head asm2_single_instr.vmp3-c0-i1-20240609-135638.log
0x0000000140001006 mov rbp, rsp
0x0000000140001009 sub rsp, 0x20
0x000000014000100D push rax
0x000000014000100E call 0x0000000140006580
0x0000000140006580 pushfq
0x0000000140006581 add qword ptr ss:[rsp+0x08], 0xFFFFFFFFBFFFEFED
0x000000014000658A push rbp
0x000000014000658B mov rbp, 0xD78290B4F6B57699
0x0000000140006595 push r9
0x0000000140006597 mov qword ptr ss:[rsp+0x18], 0x60228E5B
```

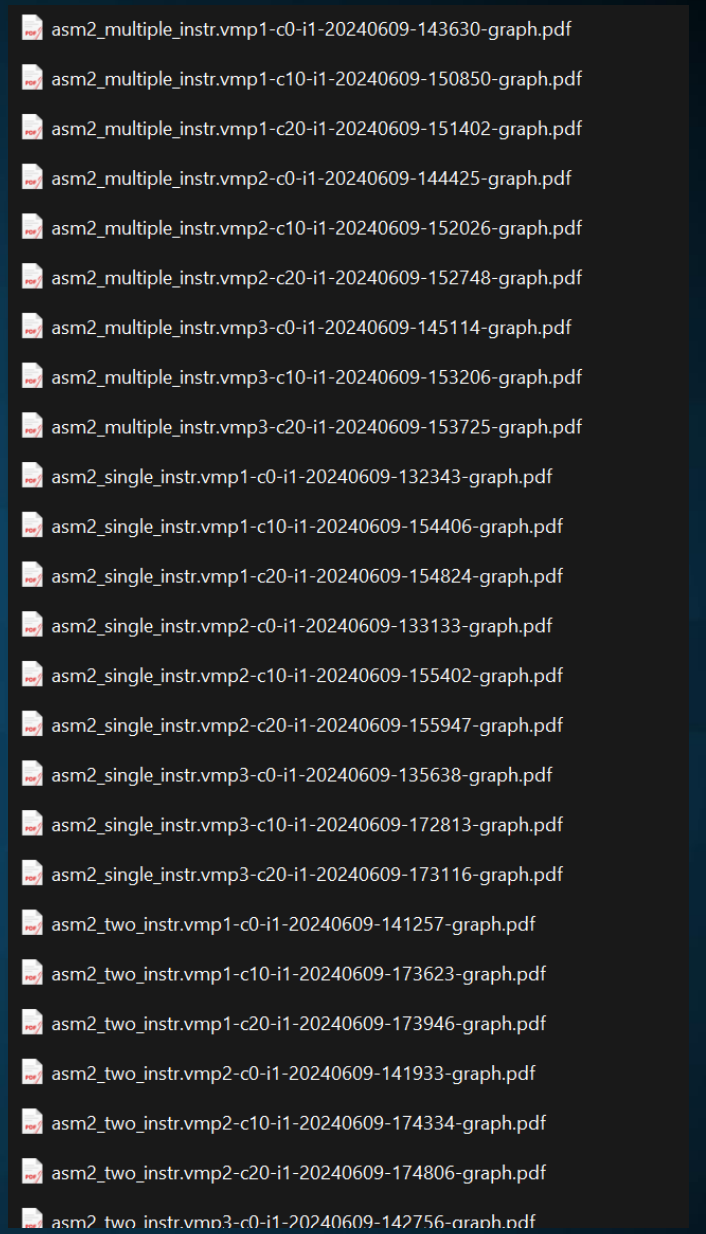
Code trace

```python
import sys
import graphviz
import argparse
from pathlib import Path

cfg_instr = [ 'call','ret','ja','jae','jb','jbe','jc','jcxz','je','jg','jge','jl','jle','jmp','jna','jnae','jnb','jnbe','jnc','jne','jng','j

parser=argparse.ArgumentParser(description="This script takes an X64dbg instructions trace log file and generates a corrosponding graph.")
parser.add_argument("tracelog_filename")
args=parser.parse_args()

trace_fname = args.tracelog_filename
graph_fname = f"{Path(trace_fname).stem}-graph"

print(f"[INFO] Loading trace from {trace_fname}")

with open(trace_fname) as file:
    lines = file.readlines()
```

Python Script builds CFG as PDF

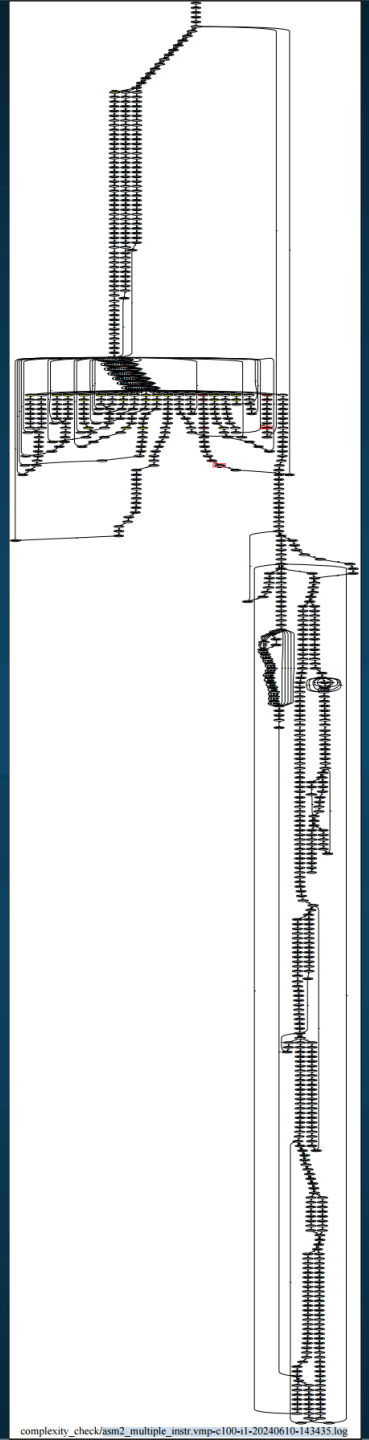


Code trace split into basic blocks and than use Graphviz to build a Code Flow Graph (CFG)



CFG in PDF Format

asm2_multiple_instr.vmp-c100-i1-20240610-143435.log



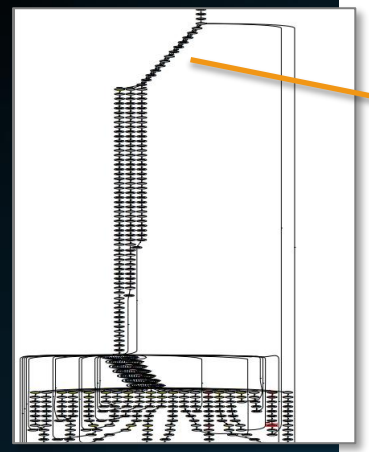
Initialization and central VM logic (next slide)

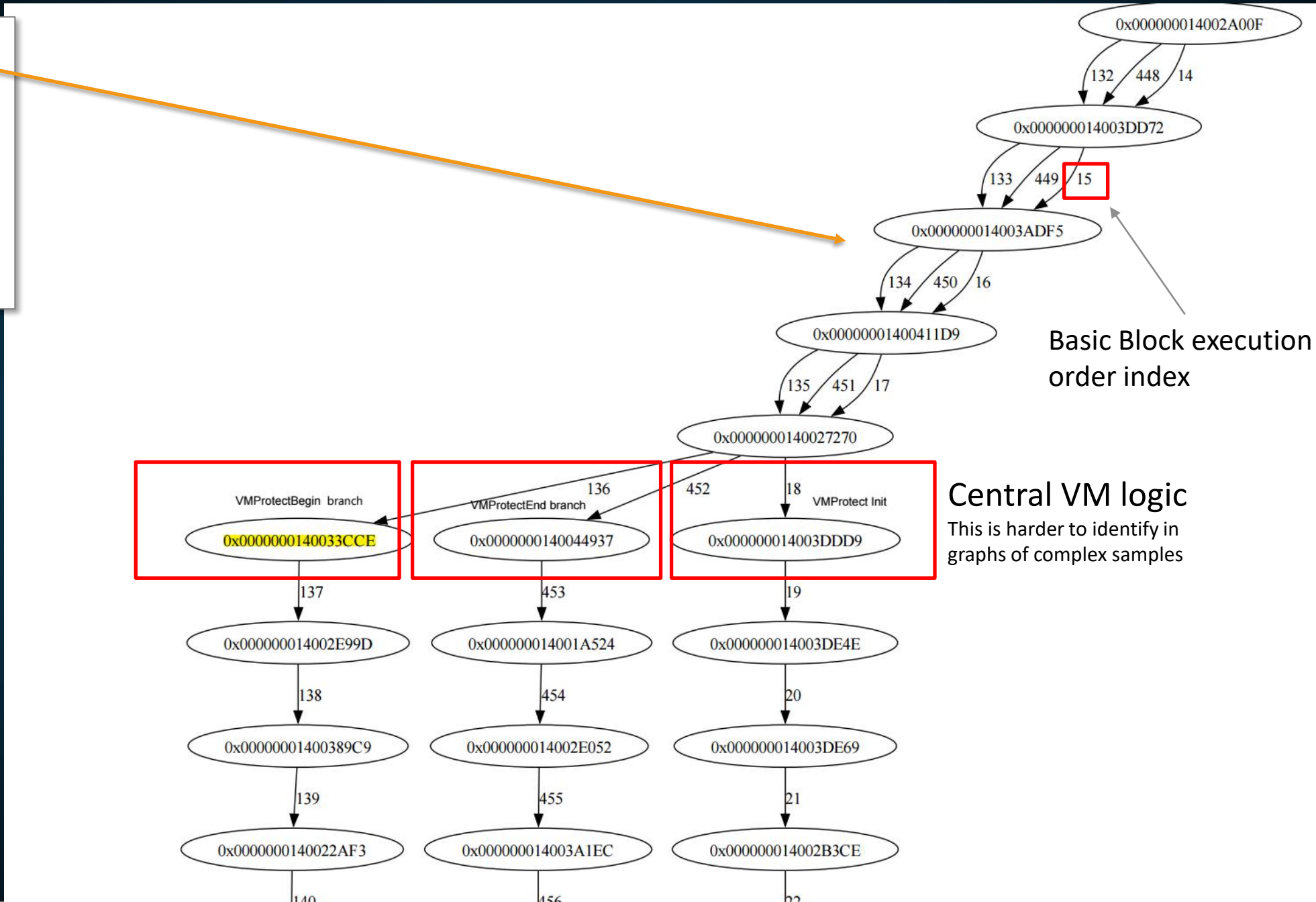
Central switch

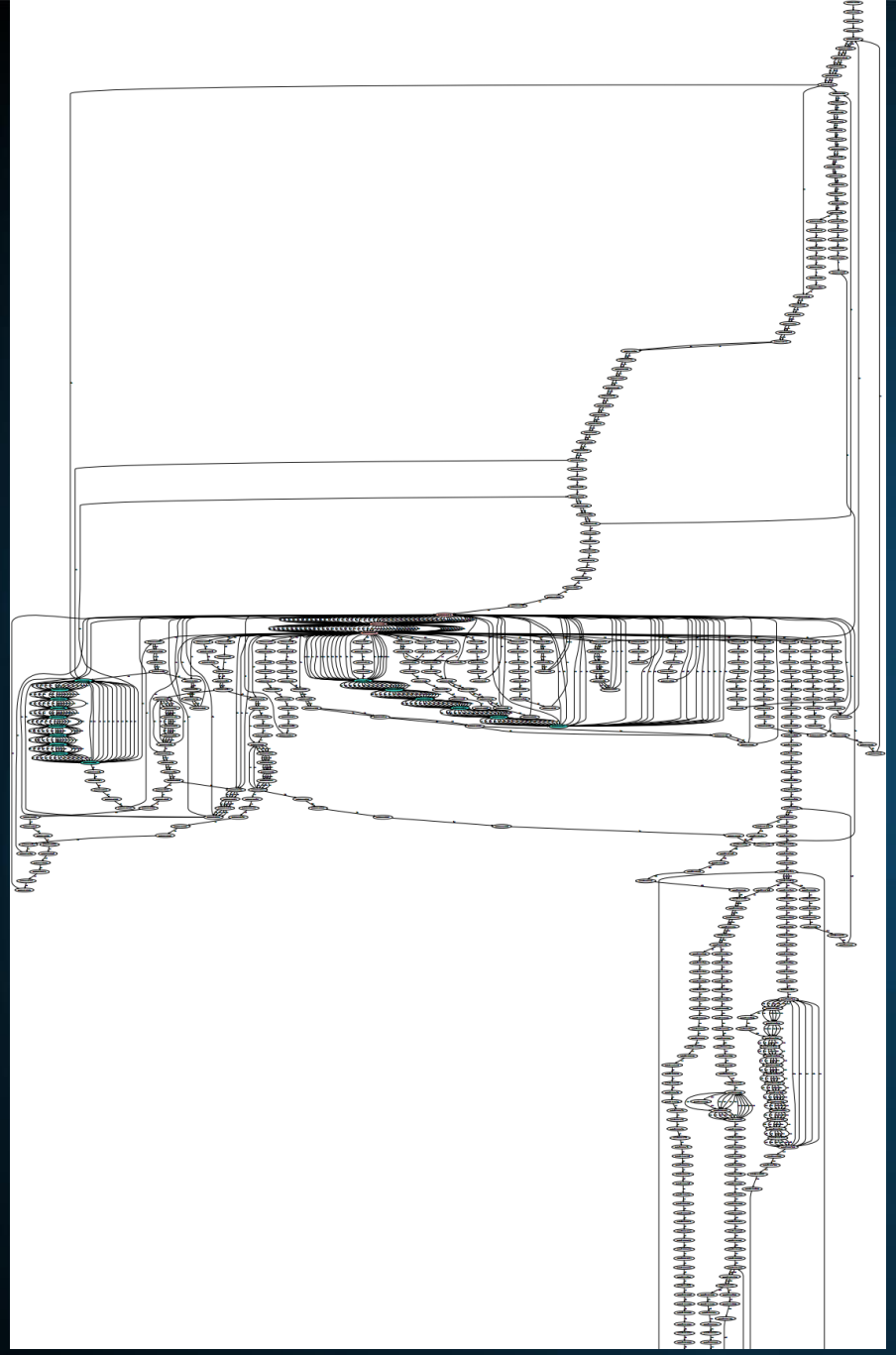
Unprotected Code

Execution order

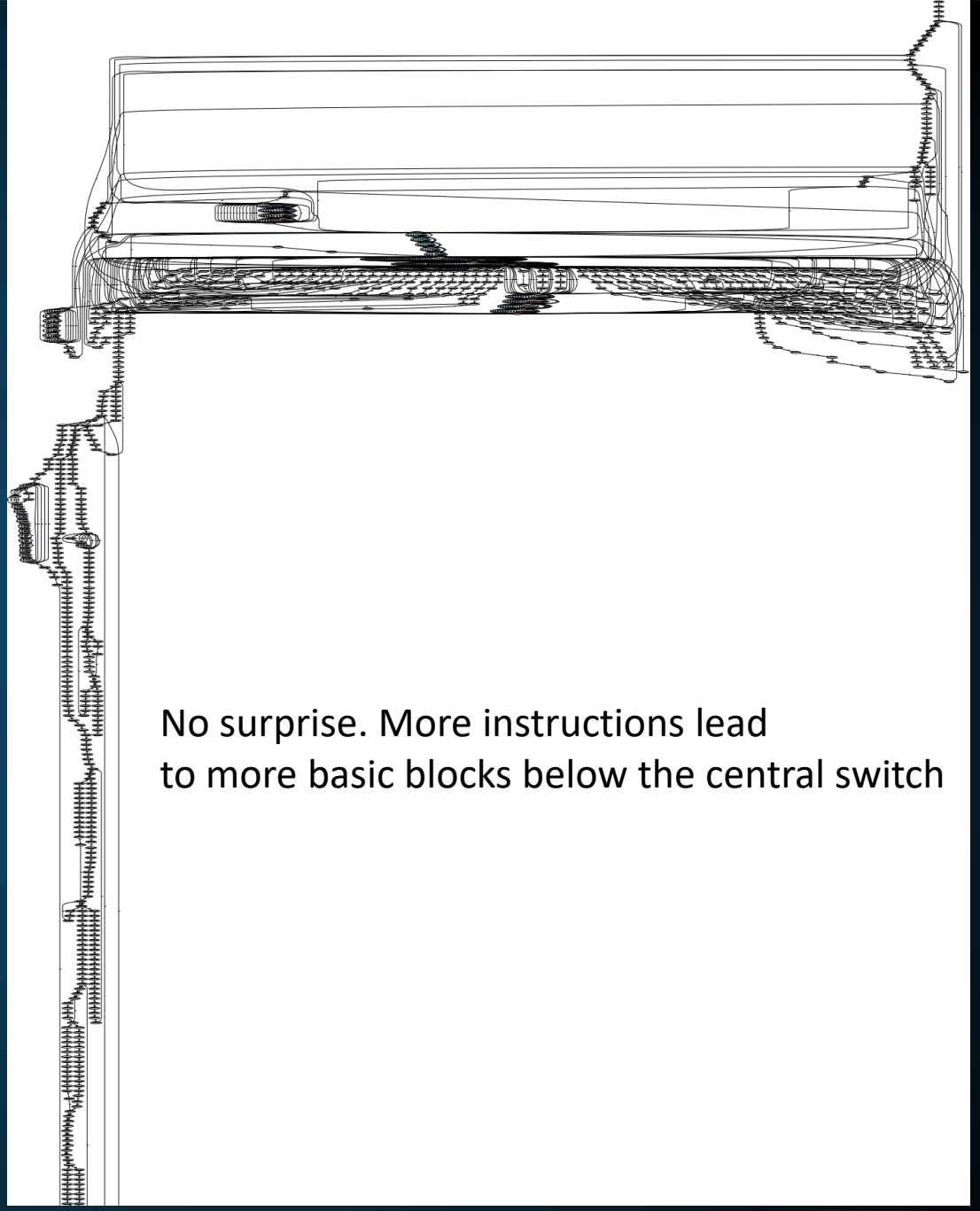
asm2_multiple_instr.vmp-c100-i1-20240610-143435.log







asm2_single_instr-c1-i1.vmp-20240610-143947.log

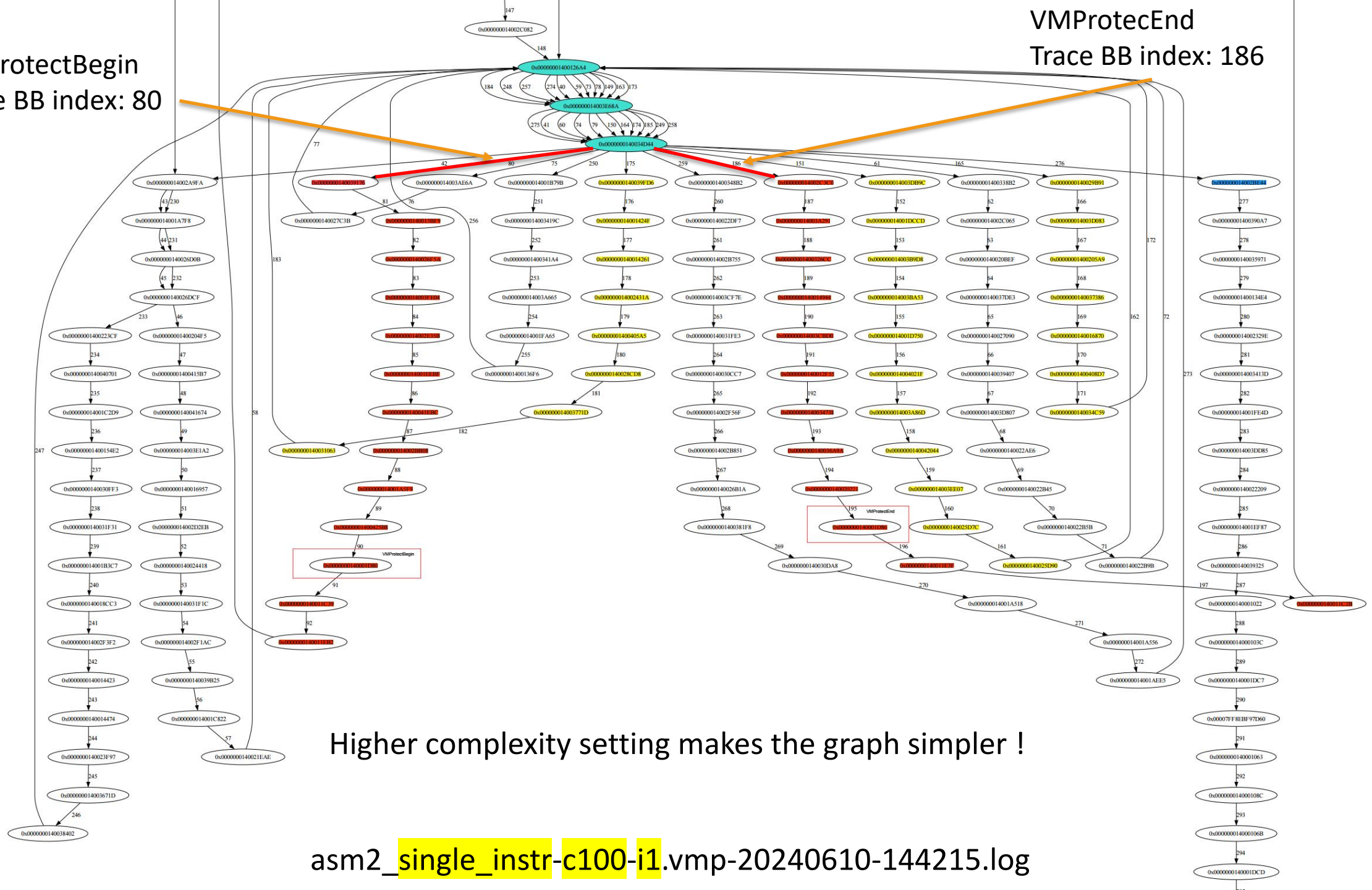


No surprise. More instructions lead
to more basic blocks below the central switch

asm2_multiple_instr.vmp-c1-i1-20240610-142851.log

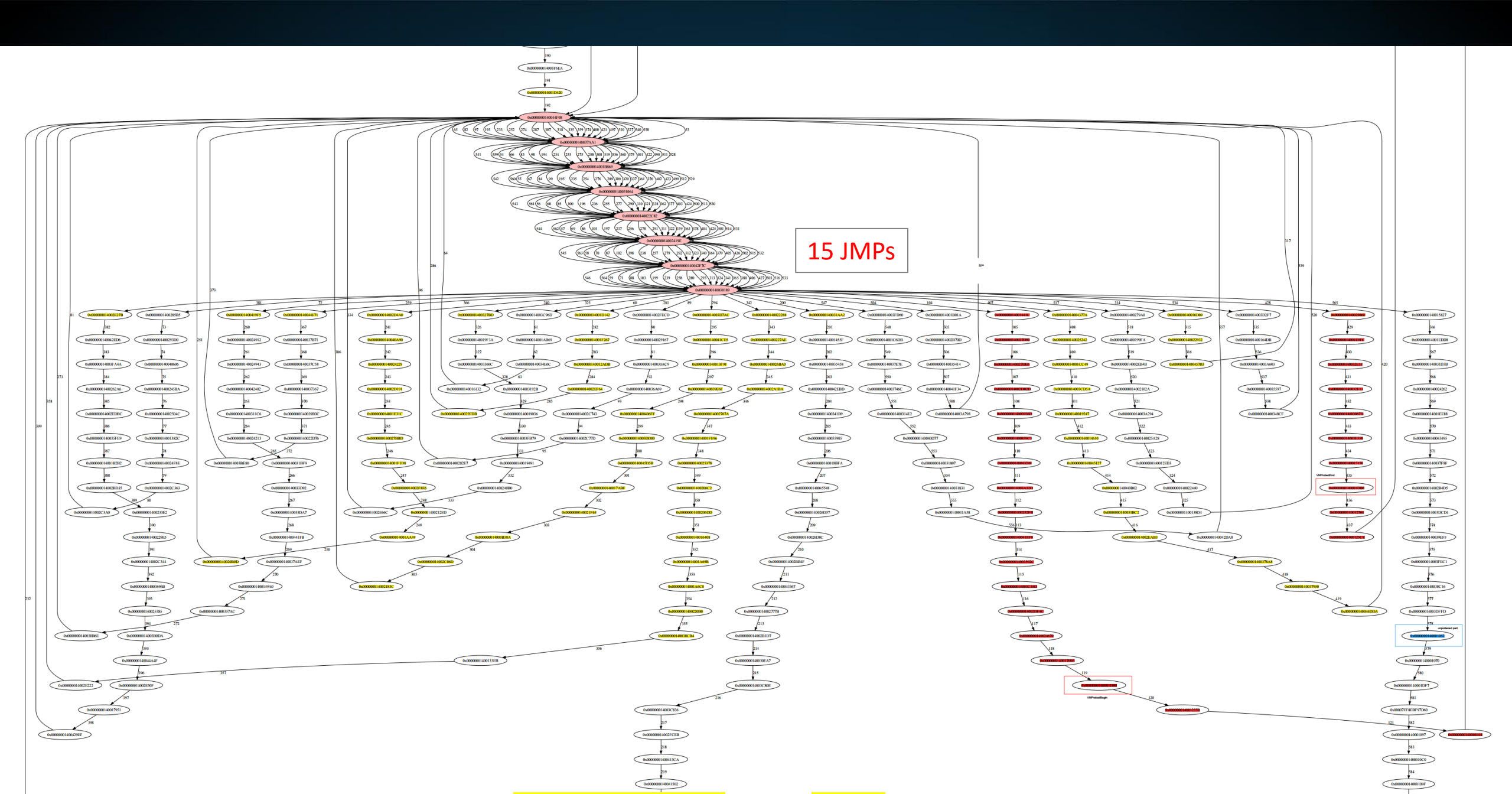
VMProtectBegin
Trace BB index: 80

VMProtecEnd
Trace BB index: 186



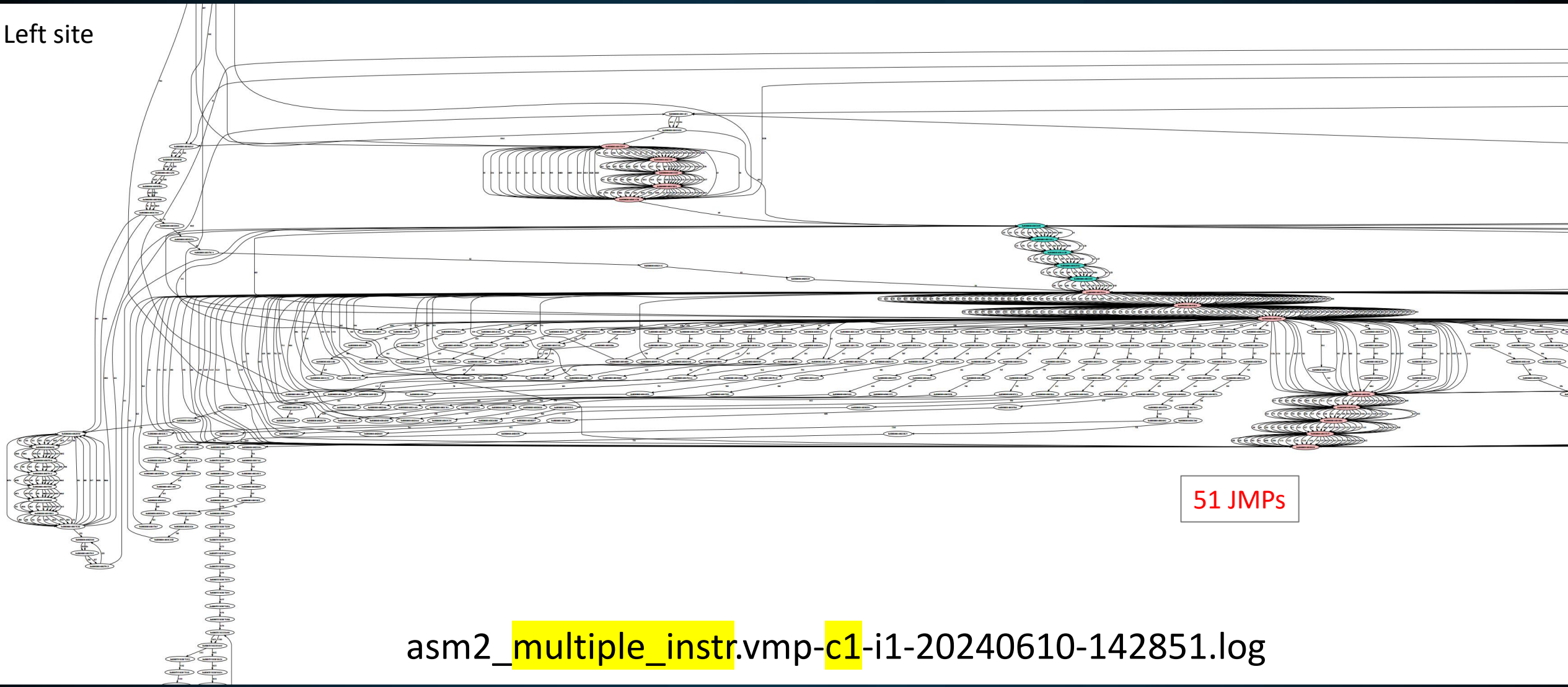
Higher complexity setting makes the graph simpler !

asm2_single_instr-c100-i1.vmp-20240610-144215.log



asm2_multiple_instr.vmp-c100-i1-20240610-143435.log

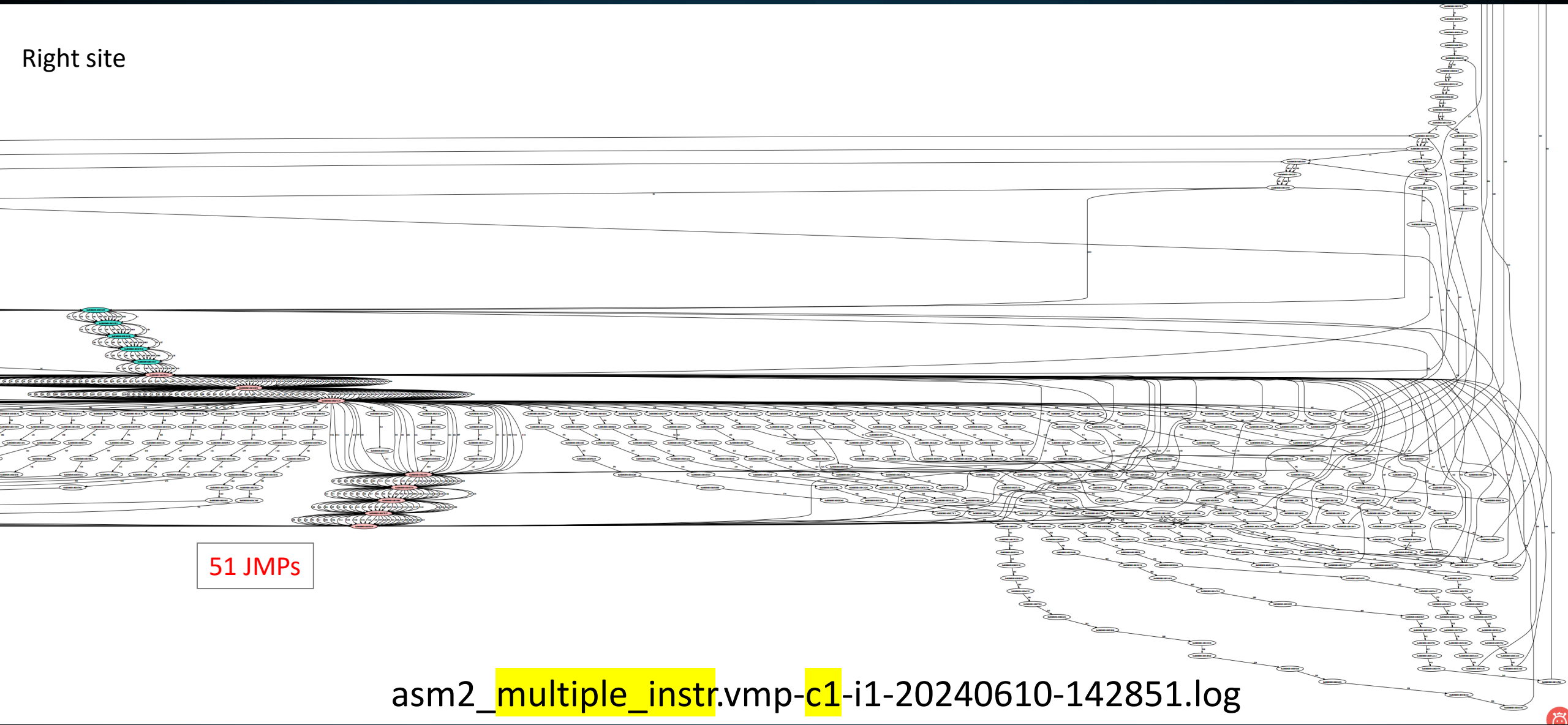
Left site



51 JMPs

asm2_multiple_instr.vmp-c1-i1-20240610-142851.log

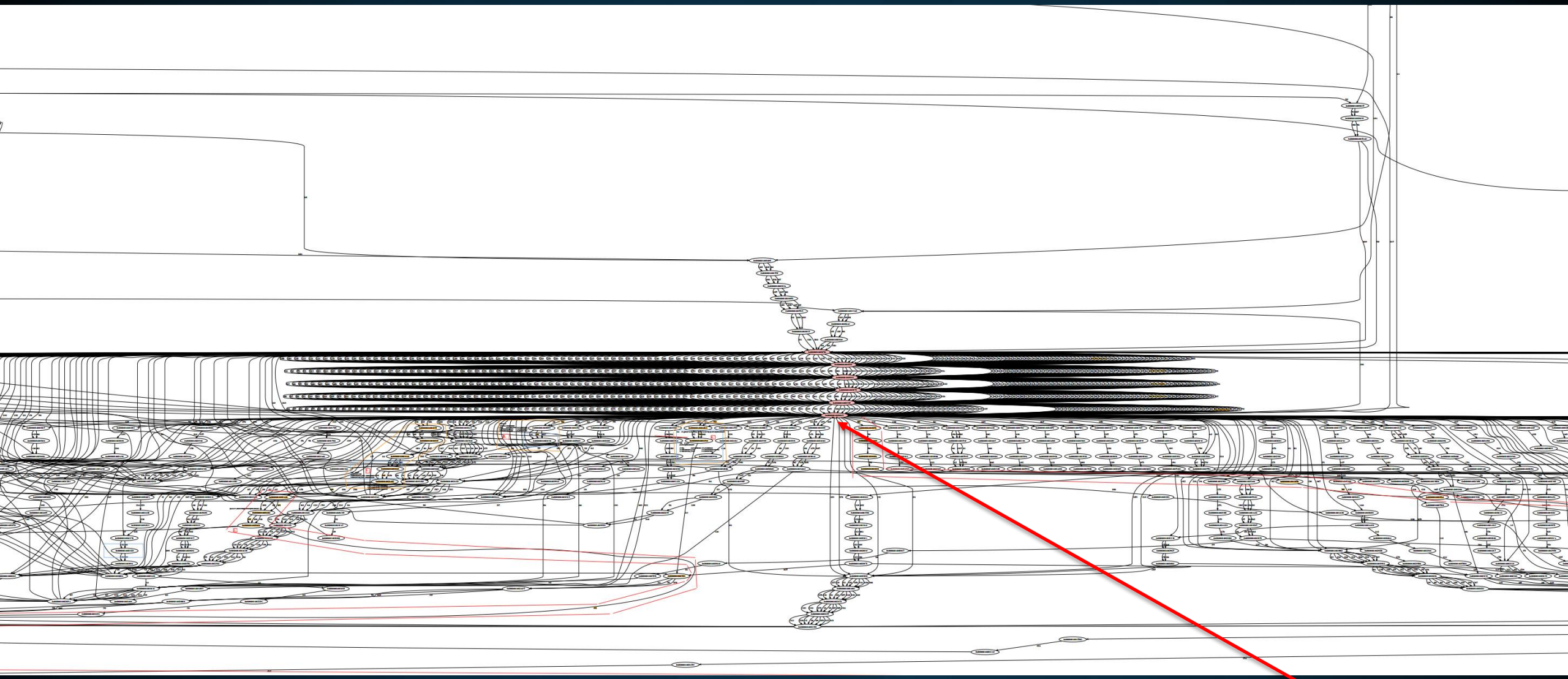
Right site



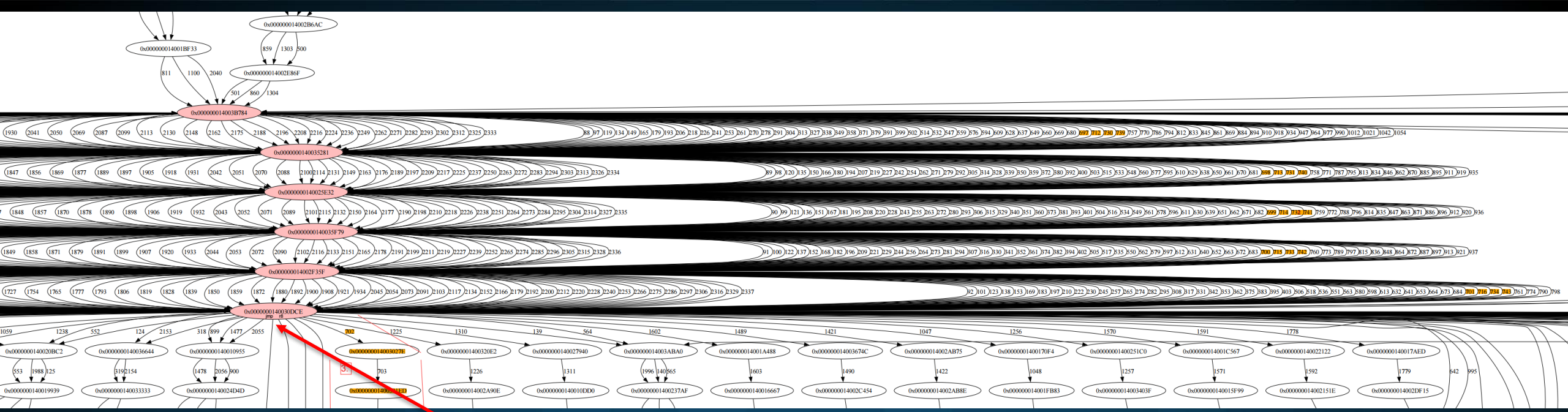
51 JMPs

asm2_multiple_instr.vmp-c1-i1-20240610-142851.log

VMProtect Internals



asm2_5incs_3mov.vmp1-c0-i1.exe

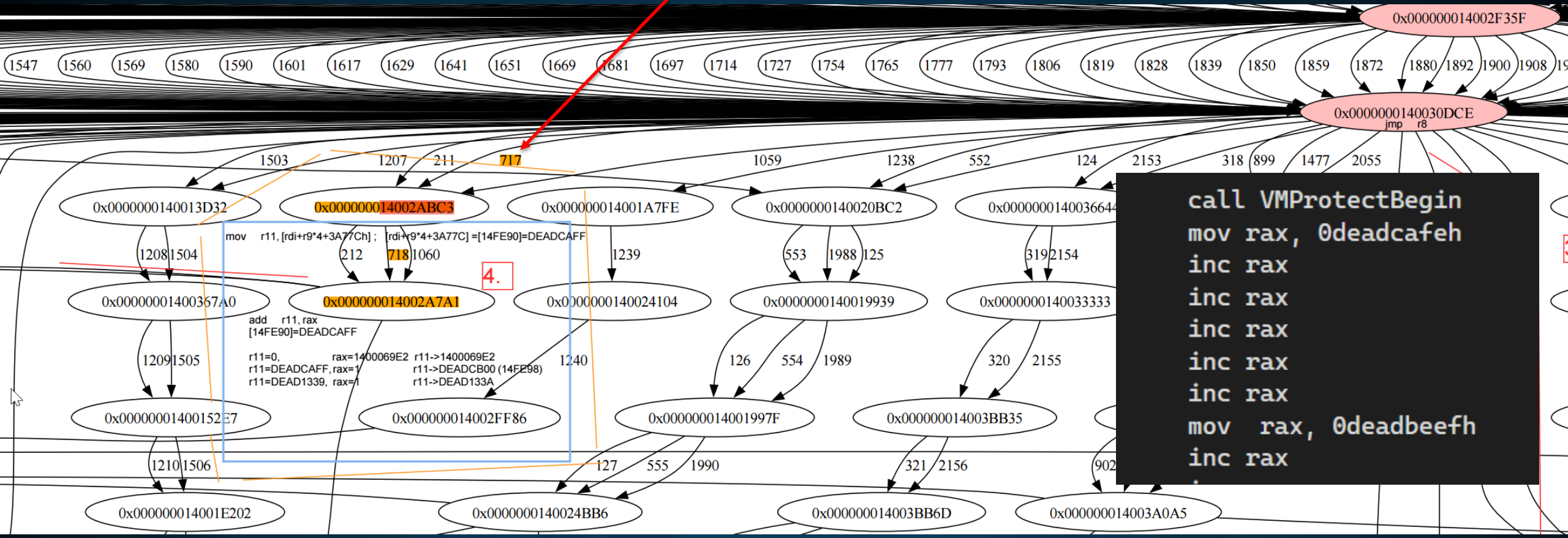


Next virtual Instruction Block
e.g. VMHandler “inc <register>”

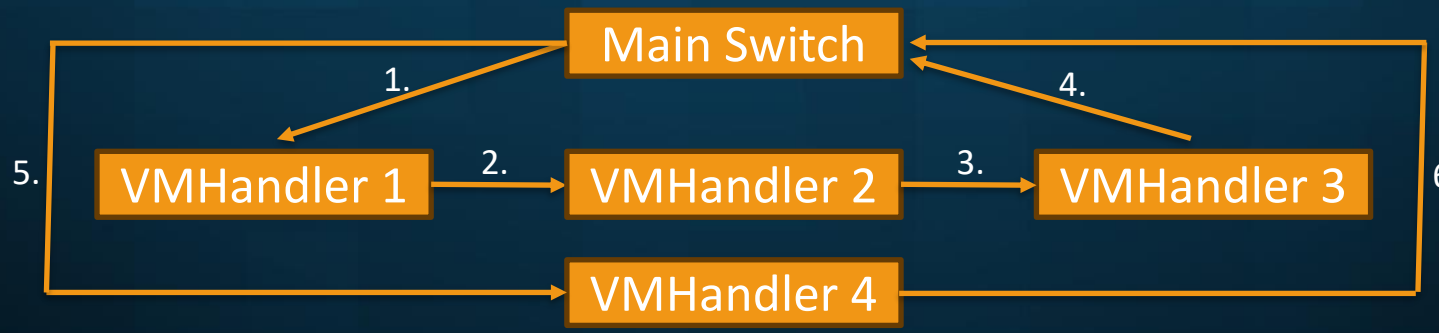
In this case it is R8, but this is random, could also be another register

How are VMHandlers executed and connected ?

We are starting here on the next slide at block 717




```
call VMProtectBegin
mov rax, 0deadcafeh
inc rax
inc rax
inc rax
inc rax
inc rax
mov  rax, 0deadbeefh
inc rax
```



Simplified Logic

Calculation of the next vInstr Block – Block 14002ABC3 (717)

The code below is distributed over multiple lines of code, plus a lot of dead code and other instruction in-between

```
sub     rbx, 4              ; next offset from PE

mov     edx, [rbx+r9*8-74B78h] ; rbx=1400106B9 r9=E96F -> 1400106B9  -> 8F5A8928
                            ;        rbx=1400068CF r9=E96F -> 1400068CF  -> 11F3C063
                            ;        rbx=14000678B r9=E96F -> 14000678B  -> 11F3F9BA

xor     edx, ebp            ; ebp=0x15834,     rdx -> 0x8f5bd11c
                            ; ebp=0x19eaf3830, rdx -> 0x8f5cf853
                            ; ebp=0x19eaf01e9, rdx -> 0x8f5cf853
neg     r10w
movzx   esi, r9w
neg     edx
xor     edx, 0F8185C2Fh
sub     edx, 88BB4029h
movsx   r11d, cx
neg     edx                 ; edx -> FFFECD5E     final EDX value
                            ; edx -> FFFFE4A7
                            ; edx -> FFFFE4A7
```

ByteCodePtr can be different e.g. *sub rbx,1*

Encrypted offset to next vInstr (E966F * 8 = 74B78 -> EDX ptr to offset) -> VMHandlerTableEntry

XOR EDX (VMHandlerTableEntry) with Rolling key in EBP

Do some math to calculate final value for EDX (offset to next vInstr handler)

```
push    rbp                 ; rsp=14FBD0  rsp->14FBC8
xor     [rsp+r10*2-18h+var_1FEC6], edx ; rsp=14FBC8, r10=FF6F*2=1FEDE, [rsp+r10*2-1FEDE]=[14FBC8]=00015834    XOR [RBP] with EDX
                            ;                rsp=14FBC8, r10=FF6F*2=1FEDE, [rsp+r10*2-1FEDE]=[14FBC8]=9EAF3830
                            ;                rsp=14FBC8, r10=FF6F*2=1FEDE, [rsp+r10*2-1FEDE]=[14FBC8]=9EAF01E9
                            ;                rsp = 14FBC8 -> FFFF956A
                            ;                rsp = 14FBC8 -> 16150DC97
                            ;                rsp = 14FBC8 -> 16150E54E
pop     rbp                 ; rbp -> FFFF956A    [ move RSP to new RBP ]   new rolling key
                            ; rbp -> 16150DC97
                            ; rbp -> 16150E54E
```

New rolling Key in EBP

```
add     r8, rdx             ; r8=14002ABC3 rdx=FFFFFFFFFFFECD5E r8->140017921
                            ; r8=14002ABC3 rdx=FFFFFFFFFFFFE4A7 r8->14002906A
                            ; r8=14002ABC3 rdx=FFFFFFFFFFFFE4A7 r8->14002906A
```

Next vInstr handler

This vInstr handler start

… next slide

Calculation of the next vInstr Handler – Block 14002906A (720)

```
sub     rbx, 4          ; next 32bit vInstr offset from PE
mov     edx, [rbx]      ; rbx=14001075B -> A8C858C0   encrypted offset to next vInstr block
                        ; rbx=1400069BC -> A8C908F6   hardcoded in PE
                        ; rbx=1400068CB -> 564EC3B3
                        ; rbx=140006787 -> 87B4FB6A
                        ; rbx=1400050CF -> 57347DD2
```

```
xor     edx, ebp        ; edx=A8C858C0 ebp=BFFE05AD edx -> 17365D6D   XOR offset with rolling key in ebp
                        ; edx=A8C908F6 ebp=BFFF559B edx -> 17365D6D
                        ; edx=564EC3B3 ebp=6150DC97 edx -> 371E1F24
                        ; edx=87B4FB6A ebp=6150E54E edx -> E6E41E24
                        ; edx=57347DD2 ebp=400220BF edx -> 17365D6D
```

```
lea     edx, [rcx+rdx-35BA0A4Bh] ; rcx=78534C82 rdx=17365D6D -> 59CF9FA4
                                 ; rcx=78534C82 rdx=17365D6D -> 59CF9FA4
                                 ; rcx=78534C82 rdx=371E1F24 -> 79B7615B
                                 ; rcx=78534C82 rdx=E6E41E24 -> 1297D605B
                                 ; rcx=78534C82 rdx=17365D6D -> 59CF9FA4
bswap   edx
not     edx
mov     esi, 21B1E6A2h
xor     edx, 0A49EA93Bh ; edx=FFFE999D  final value (offset from start of this block to next vInstr block)
                        ; edx=FFFE999D
                        ; edx=0000E1BD
                        ; edx=00012BED
                        ; edx=FFFE999D
```

Add hardcoded value in RCX with RDX

Hardcoded value loaded earlier in ECX

```
push    rbp             ; rsp=14FBD0 -> 14FBC8 (=1BFFF559B)
```

```
xor     [rsp+r11-20h+var_1168D419], edx ; rsp=14FBC8 r11=1168D439 [rsp+r11*1-1168D439] =  14FBC8 -> 140009C30   XOR [RBP], EDX
                                        ; rsp=14FBC8 r11=1168D439 [rsp+r11*1-1168D439] =  14FBC8 -> 14001CC06
                                        ; rsp=14FBC8 r11=1168D439 [rsp+r11*1-1168D439] =  14FBC8 -> 161503D2A
                                        ; rsp=14FBC8 r11=1168D439 [rsp+r11*1-1168D439] =  14FBC8 -> 16151CEA3
                                        ; rsp=14FBC8 r11=1168D439 [rsp+r11*1-1168D439] =  14FBC8 -> 1BFFCB922
```

New rolling key

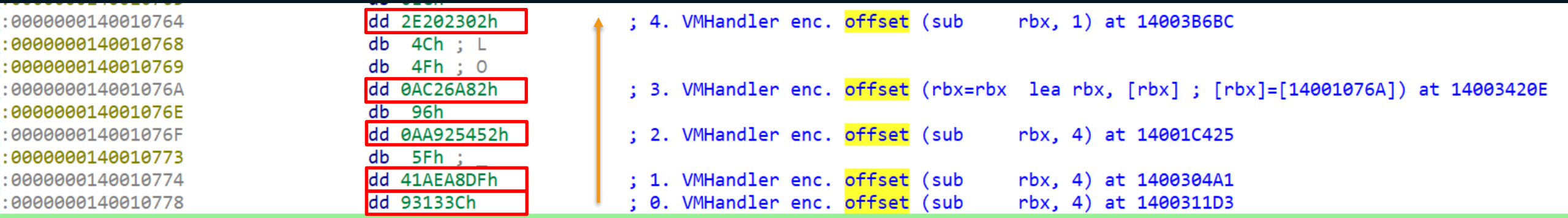
```
pop     rbp             ; 14001CC06
```

```
add     r8, rdx         ; r8=14002906A, rdx=FFFFFFFFFFFE999D r8->140012A07
                        ; r8=14002906A, rdx=FFFFFFFFFFFE999D r8->140012A07
                        ; r8=14002906A, rdx=000000000000E1BD r8->140037227
```

Next vInstr Handler

This vInstr Handler start

Remember the VMHandler table offset (*sub rbx,1) ?*



The VMHandler table is not linear.

```
sub     rbx, 4
mov     edx, [rbx+r9*8-74B78h] ;
```

CISCO TALOS

Remember the rolling key ? What about R10 ?

```
push    rbp                 ; rsp=14FBD0  rsp->14FBC8
xor     [rsp+r10*2-18h+var_1FEC6], edx ; rsp=14FBC8, r10=FF6F*2=1FEDE, [rsp+r10*2-1FEDE]=[14FBC8]=00015834    XOR [RBP] with EDX
                        ;               rsp=14FBC8, r10=FF6F*2=1FEDE, [rsp+r10*2-1FEDE]=[14FBC8]=9EAF3830
                        ;               rsp=14FBC8, r10=FF6F*2=1FEDE, [rsp+r10*2-1FEDE]=[14FBC8]=9EAF01E9
                        ;               rsp = 14FBC8 -> FFFF956A
                        ;               rsp = 14FBC8 -> 16150DC97
                        ;               rsp = 14FBC8 -> 16150E54E
pop     rbp             ; rbp -> FFFF956A    [ move RSP to new RBP ]   new rolling key ?
                        ; rbp -> 16150DC97
                        ; rbp -> 16150E54E
```

New rolling Key in EBP

… it's all just obfuscation:

```
mov     eax, 88807D21h  ; --- new block 14002ABC3
lea     rcx, ds:0FFFFFFFFDC26EF2Dh[rax*2] ; [rax*2-23D910D3]=[ED27 E96F]
neg     cx              ; rcx -> 00000000ED271691
movzx   r10d, cl
neg     r10w            ; r10 -> FF6F
xor     [rsp+r10*2-18h+var_1FEC6], edx ; rsp=14FBC8, r10=FF6F*2=1FEDE, [rsp+r10*2-1FEDE]=[14FBC8]=00015834
                        ;               rsp=14FBC8, r10=FF6F*2=1FEDE, [rsp+r10*2-1FEDE]=[14FBC8]=9EAF3830
                        ;               rsp=14FBC8, r10=FF6F*2=1FEDE, [rsp+r10*2-1FEDE]=[14FBC8]=9EAF01E9
```

Let’s proof the assumption and get all connections of all VMHandlers…

```
dex@Win11Dev UCRT64 /c/Users/dex/Desktop/vmp/python
$ python parse_bp.py
bp 0x0000000014003D568; SetBreakpointLog 0000000014003D568, "addr=0000000014003D568,rbx={rbx},rdx={rdx},res={rbx+rdx}"
SetBreakpointCommand 0x0000000014003D568, go
bp 0x0000000014003D56F; SetBreakpointLog 0000000014003D56F, "addr=0000000014003D56F,rbx={rbx},rsi={rsi},res={rbx+rsi}"
SetBreakpointCommand 0x0000000014003D56F, go
bp 0x0000000014001AF15; SetBreakpointLog 0000000014001AF15, "addr=0000000014001AF15,r8={r8},rcx={rcx},res={r8+rcx}"
SetBreakpointCommand 0x0000000014001AF15, go
```

The python script extracts all ‘**add <reg1>, <reg2**>’ statements and sets **breakpoints** on them to get their values at runtime.

regex1 = re.compile(r'(0x[01234567890ABCDEF]{16}) (ad[cd]) (r.{1,2}), (r.{1,2})’)

```
add     r8, rdx         ; r8=14002906A, rdx=FFFFFFFFFFFE999D r8->140012A07
                        ; r8=14002906A, rdx=FFFFFFFFFFFE999D r8->140012A07
                        ; r8=14002906A, rdx=000000000000E1BD r8->140037227
```

Next vInstr Handler

This vInstr Handler start

… after feeding it into X64dbg:

Execution Order

```
addr=0000000140003D568,rbx=4001077C,rdx=0,res=4001077C
addr=0000000140003D56F,rbx=4001077C,rsi=100000000,res=14001077C
addr=0000000140001AF15,r8=1400311D3,rcx=FFFFFFFFFFFF2CE,res=1400304A1
addr=00000001400338DF,rdx=0,r8=1400304A1,res=1400304A1
addr=0000000140030500,r8=1400304A1,r11=FFFFFFFFFFEBF84,res=14001C425
addr=0000000140001C4A9,r8=14001C425,r9=17DE9,res=14003420E
addr=0000000140001C510,r10=70,rsp=14FBD0,res=14FC40
addr=000000140003428E,r8=14003420E,rsi=FFFFFFFFFFE0A49,res=140014C57
addr=000000140003708A,rcx=0,rsp=14FBD0,res=14FBD0
addr=0000000140031BFC,r8=140014C57,r10=12E10,res=140027A67
addr=0000000140037041,r8=140027A67,rcx=1603,res=14002906A
addr=0000000140026A90,r8=14002906A,rdx=FFFFFFFFFFE999D,res=140012A07
addr=0000000140017672,r8=140012A07,r9=C540,res=14001EF47
addr=0000000140021377,rax=110,rsp=14FBD0,res=14FCE0
addr=0000000140002B5D0,r8=14001EF47,r9=19FF1,res=140038F38
addr=0000000140038FA8,r8=140038F38,r10=1A35,res=14003A96D
addr=0000000140003A9AB,r9=D087EEFD,rax=D0870003,res=1A10EEF00
addr=0000000140002F9C3,r8=14003A96D,rsi=FFFFFFFFFFFD70C4,res=140011A31
addr=0000000140003C4AF,r8=140011A31,r11=CF42,res=14001E973
addr=0000000140026EF5,r9w=AD8F,r9w=AD8F,res=15B1E
addr=0000000140026F6A,r8=14001E973,r11=11B2E,res=1400304A1
addr=0000000140030500,r8=1400304A1,r11=DF3,res=140031294
addr=0000000140010DB6,r8=140031294,r11=FFFFFFFFFFFEB191,res=14001C425
```

False Positive of the regex

Proofs our assumption it is always R8 (in this case)

… and cleaned up (including an anomaly check).

```
$ python parse_bp_out.py
0000000014001AF15: 1400311D3 - 1400304A1  <---------------------------- 14001AEC4 Start Block
0000000140030500: 1400304A1 - 14001C425
0000000014001C4A9: 14001C425 - 14003420E
0000000014003428E: 14003420E - 140014C57
0000000140031BFC: 140014C57 - 140027A67
0000000140037041: 140027A67 - 14002906A
                                                                          Initialization
0000000014002948B: 140038ED7 - 14002ECDD
0000000014002EE21: 14002ECDD - 140030A15
0000000140015AF6: 140030A15 - 140031C62
0000000140031C98: 140031C62 - 140031493
0000000014001AF15: 1400311D3 - 14001E973  <---------------------------- 14001AEC4 Start Block
0000000140026F6A: 14001E973 - 1400304A1
0000000140030500: 1400304A1 - 14001C425
0000000014001C4A9: 14001C425 - 14003420E
                                                                          VMProtectStart
0000000140014F41: 14002441D - 140023A16
0000000140011169: 140023A16 - 140015B02
0000000140015B6F: 140015B02 - 14002AD95
0000000014001AF15: 1400311D3 - 140038F38  <---------------------------- 14001AEC4 Start Block
0000000140038FA8: 140038F38 - 14003A96D
0000000014002F9C3: 14003A96D - 14001DC41
0000000140021A1D: 14001DC41 - 140011A31
                                                                          VMProtectEnd
```

Breakpoint address

Start of VMHandler

Next VMHandler

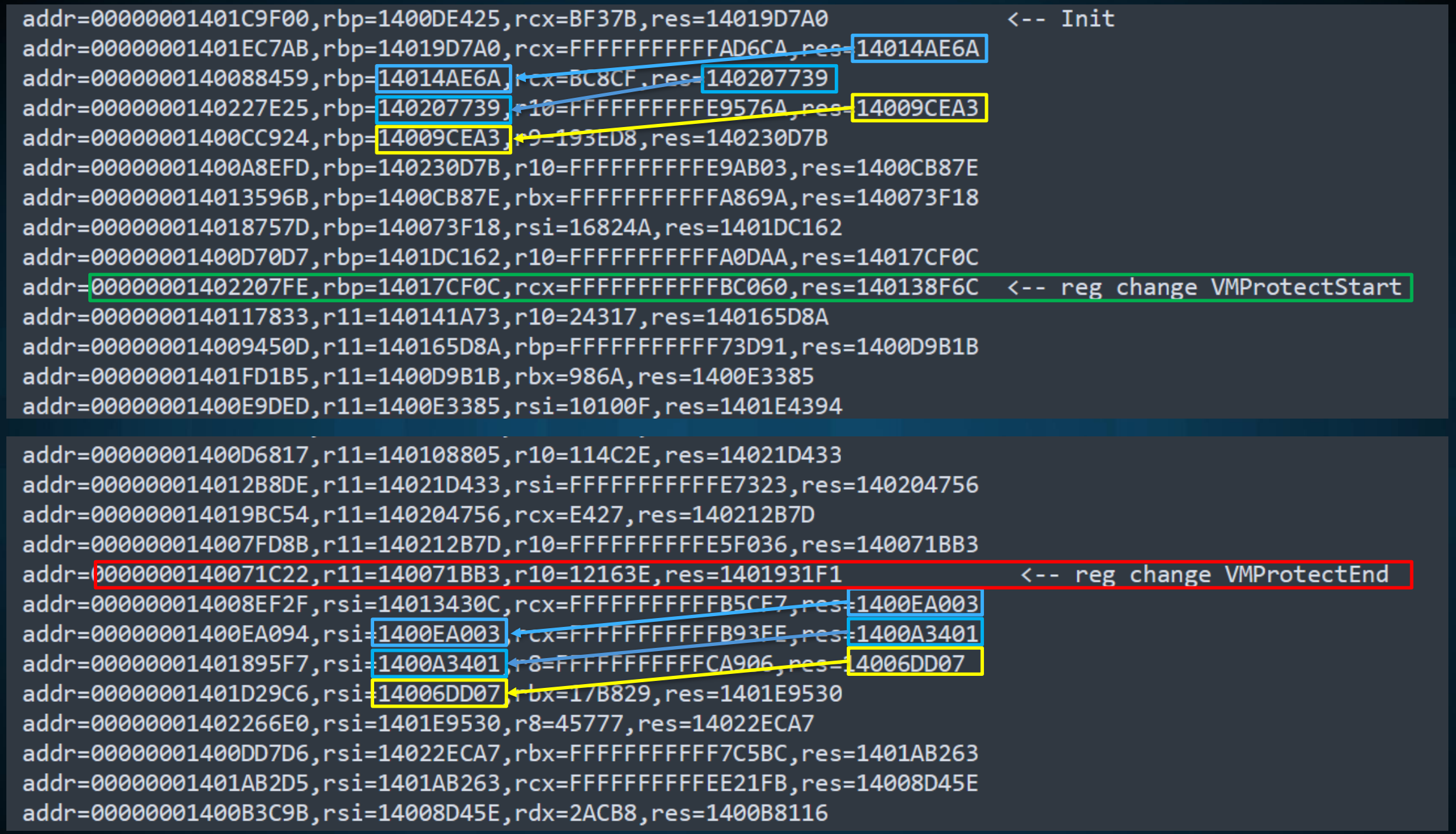
asm2_5incs_3mov.vmp1-c0-i1-20240620-105921

Does it make a difference if we change the complexity rate ? No.

```
$ python parse_bp_out.py
00000000140033957: 140031F48 - 140013CA7  <-- 140033957 Init Block (Anomaly: 1)
000000014003C78C: 140013CA7 - 140013CA7
000000014003C78C: 140013CA7 - 140013CA7
000000014003C78C: 140013CA7 - 140036452
000000014003ED68: 140036452 - 140034A5B
0000000140019D90: 140034A5B - 14003C445
000000014002AECD: 14003C445 - 14003999C
0000000140016EF9: 14003999C - 140024FD7
0000000140146F3: 140024FD7 - 140024FD7
0000000140146F3: 140024FD7 - 140024FD7
0000000140146F3: 140024FD7 - 140024FD7
0000000140146F3: 140024FD7 - 140024FD7
0000000140146F3: 140024FD7 - 140024FD7
0000000140146F3: 140024FD7 - 140024FD7
0000000140146F3: 140024FD7 - 140024FD7
0000000140146F3: 140024FD7 - 140032378
0000000140323FD: 140032378 - 140041975
00000000140033957: 140031F48 - 140013CA7  <-- 140033957 VMProtectStart Block (Anomaly: 2)
000000014003C78C: 140013CA7 - 140013CA7
```

```
0000000140146F3: 140024FD7 - 140024FD7
0000000140146F3: 140024FD7 - 1400298B1
00000000140033957: 140031F48 - 140013CA7  <-- 140033957 VMProtectEnd Block (Anomaly: 3)
000000014003C78C: 140013CA7 - 140013CA7
000000014003C78C: 140013CA7 - 140013CA7
000000014003C78C: 140013CA7 - 140036452
```

asm2_5incs_3mov.vmp1-c100-i1-20240625-110446

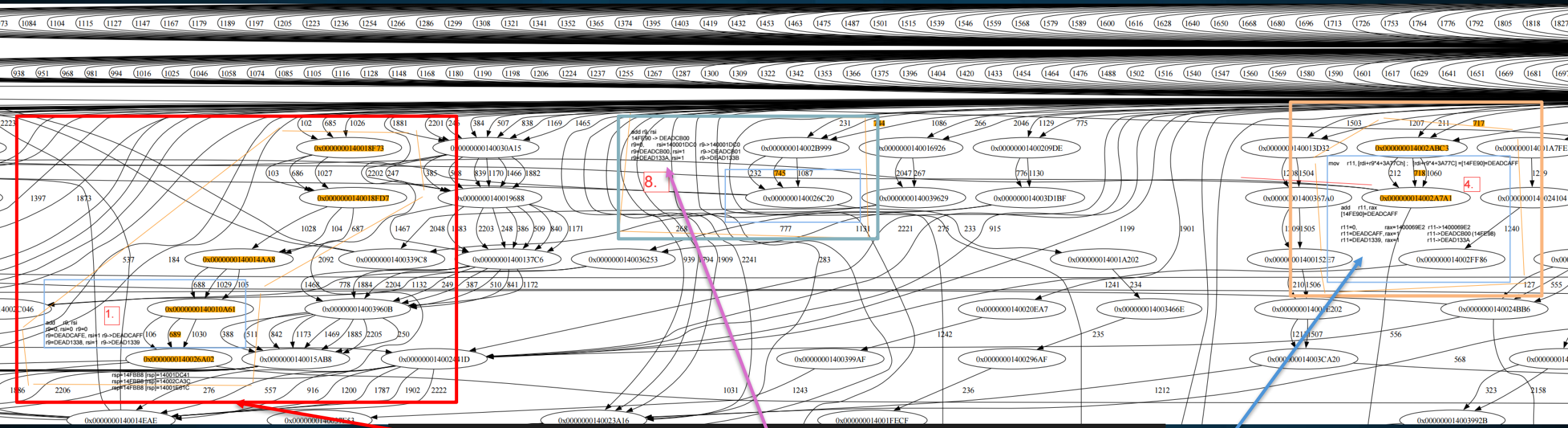


asm2_5incs_3mov.vmp1-c100-i10-20240625-110446

VMHandler

Translation of “INC EAX”

1st : There is not just one “INC <Register>” VMHandler and they all look different!



```
call VMProtectBegin
mov rax, 0deadcafeh
inc rax                 ; 0xdeadcafe -> 0xdeadcaff
inc rax                 ; 0xdeadcaff -> 0xdeadcb00
inc rax                 ; 0xdeadcb00 -> 0xdeadcb01
inc rax                 ; 0xdeadcb01 -> 0xdeadcb02
inc rax                 ; 0xdeadcb02 -> 0xdeadcb03
```

“INC <Register>” handler (likely add <p1,p2>) – VMHandler Nr. 4 in Graph

```
mov     r11, [rdi+r9*4+3A77Ch] ;  e.g  [rdi+r9*4+3A77C] =[14FE90]=DEADCAFF    Parameter 1
mov     rax, [rdi+r10*4-3FDACh] ; e.g. [rdi+r10*4-3FDAC]=[14FE98]=1           Parameter 2
```

Get Parameter 1 and 2

```
add     r11, rax          ; r11=0,        rax=1400069E2  r11->1400069E2
                          ; r11=DEADCAFF, rax=1          r11->DEADCB00
                          ; r11=DEAD1339, rax=1          r11->DEAD133A
```

“inc eax”
(add p1,p2)

```
mov     [rdi+r9*4+3A77Ch], r11 ; r11=1400069E2 rdi=14FEB0 r9=FFFFFFFFFFFF1621 -> 14FEB0
                               ;       r11=DEADCB00  rdi=14FE98 r9=FFFFFFFFFFFF1621 -> 14FE98
                               ;       r11=DEAD133A  rdi=14FE98 r9=FFFFFFFFFFFF1621 -> 14FE98
```

Store result

```
mov rax, 0deadca
inc rax
inc rax
inc rax
inc rax
inc rax
```

? Next slide…

Next “INC” – VMHandler – Nr. 8 in Graph

```
mov     r9, [rdi+rax*2-462Ch] ; di=14FEA0 rax=2316  [rdi+rax*2-462c]=14FEA0 -> 0
                              ; rdi=14FE90 rax=2316 [rdi+rax*2-462c]=14FE90 -> DEADCB00 (moved from 14F
                              ; rdi=14FE90 rax=2316 [rdi+rax*2-462c]=14FE90 -> DEAD133A
```

```
add     r9, rsi           ; add r9, rsi
                          ; r9=0,        rsi=140001DC0  r9->140001DC0
                          ; r9=DEADCB00, rsi=1          r9->DEADCB01
                          ; r9=DEAD133A, rsi=1          r9->DEAD133B
```

```
mov     [rax+rdi-232Eh], r9 ; rax=232E rdi=14FEA8 [rax+rdi-232E]=14FEA8->140001DC0
                            ;     rax=232E rdi=14FE98 [rax+rdi-232E]=14FE98->1
                            ;     rax=232E rdi=14FE98 [rax+rdi-232E]=14FE98->1
```

Other “INC” - VMHandler

```
adc     rdx, rcx          ; rdx=DEADCB02
```

Different “INC” VMHandler can have different instructions, other registers, other dead code, etc

Stack is changed between VMHandlers

First mov of DEADCB00 from 14FE98 to 14FC00

```
mov     rsi, [rdi+r11*2-22D1A870h] ; rdi=14FE58 r11=1168D438  [rdi+r11*2-22D1A870] = 14FE58 -> 53AC6E
                                   ;            rdi=14FE60 r11=1168D438  [rdi+r11*2-22D1A870] = 14FE60 -> 0
                                   ;            rdi=14FE98 r11=1168D438  [rdi+r11*2-22D1A870] = 14FE98 -> DEADCB00

mov     [rdx+r11-1F3F29F5h], rsi ; rsi=56AC6E   rdx=14FC70 r11=1F3F29F5 [rdx+r11-1F3F29F5] = 14FC70
                                 ;            rsi=0      rdx=14FC80 r11=1F3F29F5 [rdx+r11-1F3F29F5] = 14FC80
                                 ;            rsi=DEADCB00 rdx=14FC00 r11=1F3F29F5 [rdx+r11-1F3F29F5] = 14FC00
```

Second mov of DEADCB00 from 14FC00 to 14FE90

```
add     rdx, [rdx+rcx]   ; rdx=0, rcx=14FC00, [rdx+rcx*1]=[14FC00]=DEADCB00

mov     [rdi], rdx       ; [rdi]=[14FE90]=DEADCAFF  rdx=DEADCB00
```

Translation of MOV rax, 0xDEADCAFE Operation

Get and “Decode String” VMHandler (From Block 140019C53 – Block 140022639)

```
0000000140038BD8    mov     rdx, [rbx+rsi*4-2F650h] ; [rbx+rsi*4-2F650]=[14000692B]=F40000014379B84E
0000000140038BE0    xor     rdx, rbp        ; rbp=140030F62  rdx->F4000000037AB72C
0000000140038BE3    mov     eax, 0E49E8082h
0000000140038BE8    neg     rdx             ; rdx -> 0BFFFFFFFC8548D4
0000000140038BEB    bts     r9, 0Fh
0000000140038BF0    not     rdx             ; rdx -> F4000000037AB72B
0000000140038BF3    sbb     ecx, r9d
0000000140038BF6    rol     rdx, 6          ; rdx = 00000000DEADCAFD
00000001400268C9    inc     rdx             ; rdx -> 0xdeadcafe
000000014002264D    mov     [r9+rdi-55A1E187h], rdx ; [r9+rdi*1-55A1E187]=[14FEB0]  rdx=DEADCAFE
```

Decode DEADCAFE value

Stack organization VMHandler (From Block 140027A67 – Block 140025CD4)

```
000000014002F059    mov     rcx, [rdi]      ; [rdi]=[14FEB0]=DEADCAFE
000000014002011C    mov     [rdx+rax*8-39620h], rcx ; [rdx+rax*8-39620]=[14FC88] , rcx=DEADCAFE
```

Move DEADCAFE from 14FEB0 to 14FC88

Actual “MOV” VMHandler (From Block 14001E330 – Block 14002BA0C)

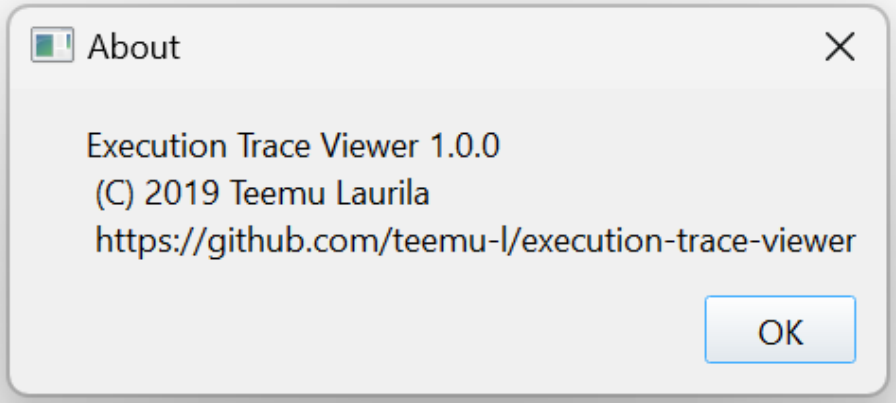
```
000000014002E248    xor     rax, [r9+rax*2] ; [r9+rax*2]=[14FC88]=DEADCAFE  rax=0
```

Write DEADCAFE to EAX

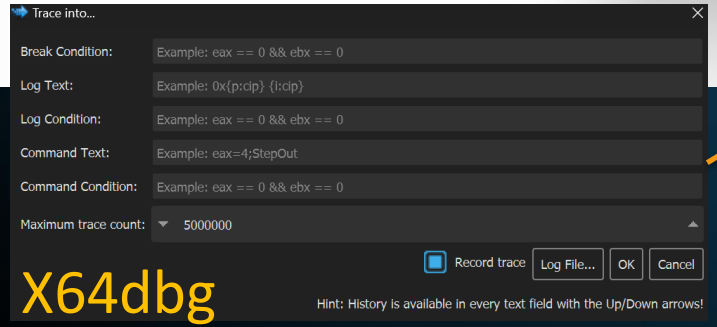
Helpful Tool – Execution Trace Viewer

Filtered ti | 1 | Page: 1 / 1 | regex=0xdeadcaff | Filter | Find: Registers

#	address	opcodes	disasm	registers
2872	0x1401da1ed	4813d3	adc rdx, rbx	rdx: 0xdeadcaff eflags
2876	0x1401da1f9	4c8bd2	mov r10, rdx	r10: 0xdeadcaff
2938	0x140224d15	498bac0270ffdd	mov rbp, qword ptr [r10 + rax - 0xa220090]	rbp: 0xdeadcaff
2939	0x140224d1d	488bdd	mov rbx, rbp	rbx: 0xdeadcaff



reg	hex	dec
rax	0x11a81e7c	296230524
rcx	0xffffffff7e8...	18446744071537485979
rdx	0xdeadcafe	3735931646
rbx	0x1	1
rsp	0x14fbf0	1375216
rbp	0x1f20361d2	8355275218
rsi	0x3fff	16383
rdi	0x14fe70	1375856
r8	0x140006994	5368736148
r9	0x19ea38551	6956483921
r10	0xfffffffffe23...	18446744073210304153
r11	0x1401da1cb	5370651083
r12	0x0	0
r13	0x0	0
r14	0x0	0
r15	0x0	0
rip	0x1401da1ed	5370651117
eflags	0x212	530
dr0	0x0	0
dr1	0x0	0
dr2	0x0	0
dr3	0x0	0
dr6	0x0	0
dr7	0x0	0
flags	C:0 P:0 Z:0 S...	



X64dbg

What’s next ?

- Blog post
- Tool release
- Collaboration ?
- VMP 4.0 … I am a bit scared ☺

Q&A

 blog.talosintelligence.com

 @talossecurity

TALOSINTELLIGENCE.COM

thank you!

blog.talosintelligence.com

@talossecurity

TALOSINTELLIGENCE.COM

Stay Connected and Up To Date

Spreading security news, updates, and other information to the public.

White papers, articles & other information
talosintelligence.com

ThreatSource Newsletter
cs.co/TalosUpdate

Talos Blog
blog.talosintelligence.com

Social Media Posts
X: @talossecurity

Videos
cs.co/talostube

Beers with Talos & Talos Takes
talosintelligence.com/podcasts

Talos publicly shares security information through numerous channels to help make the internet safer for everyone.

CISCO TALOS

TALOSINTELLIGENCE.COM