Static and Generic Deobfuscation and Devirtualization with LLVM

YUSUF "NACI" İŞLEK

bio

- Yusuf "naci" İşlek

- Researching obfuscation/deobfuscation

- https://github.com/NaC-L

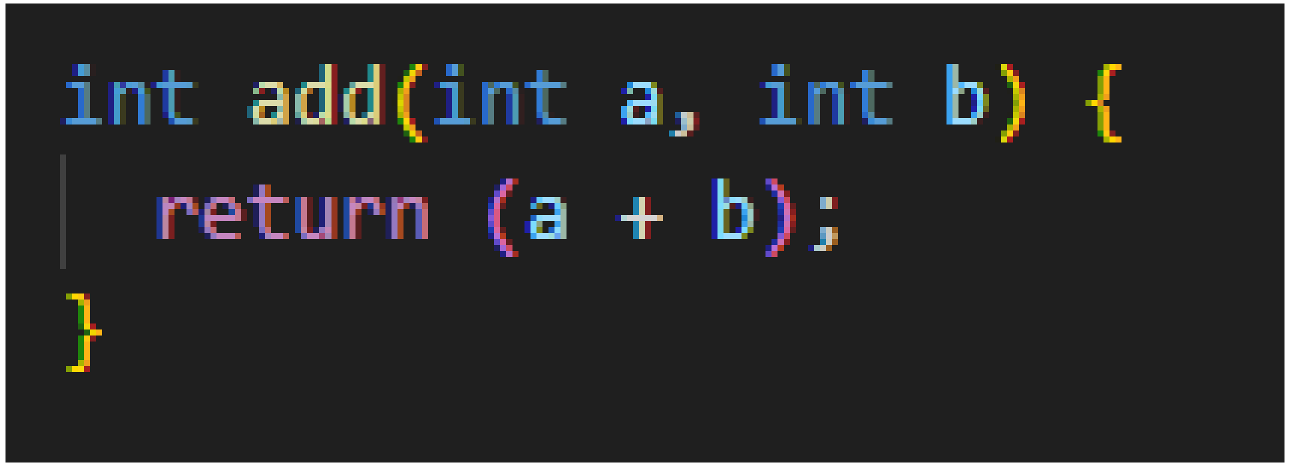
The goal

- Understanding what obfuscation is

- Reducing effort for deobfuscating protected software

What is (vm based) obfuscation and why is it an issue

A basic add function:



```c
int add(int a, int b) {
    return (a + b);
}
```

What is (vm based) obfuscation and why is it an issue

Toy VM example:

https://godbolt.org/z/Yq3abMcao

Traditional ways to deal with VM based protection

Static analysis of vm structure:

- Trying to understand each opcode and analysing the VM bytecode
- Time-consuming
- Becomes obsolete when VM structure is changed
- Need to anaylse the bytecode correctly and being able to understand what it does

Traditional ways to deal with VM based protection

Dynamic analysis of vm structure:

- A function might have multiple, misdirecting behaviour
- VM protection might have Anti-emulation, Anti-VM, Anti-Debug
- VM content stays as a black box

What Mergen does different

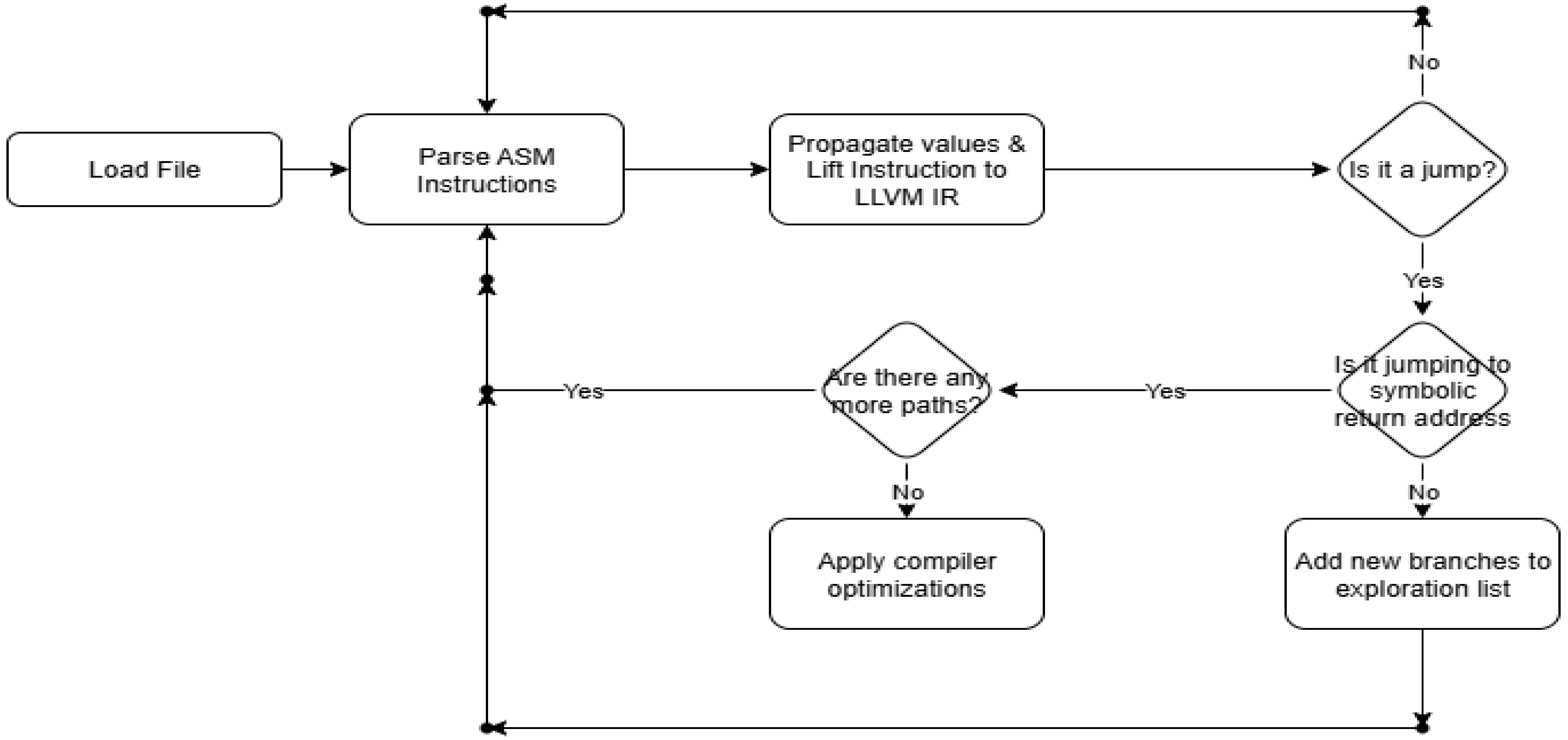
What Mergen does different?

- Aims to be a generic solution

- Doesn’t execute any code, explores paths symbolicly

- Uses a technique called "dynamic lifting" to use compiler optimizations on obfuscation

Workflow chart



Load File
Parse ASM Instructions
Propagate values & Lift Instruction to LLVM IR
Is it a jump?
No
Yes
Is it jumping to symbolic return address
Yes
No
Are there any more paths?
Yes
No
Apply compiler optimizations
Add new branches to exploration list

What is "lifting"?

- Taking a lower level language and "lifting" into a higher level language.
- Instead of fetch/decode/execute, we do fetch/decode/transform
- In "dynamic lifting" we propagate the values

Why Lifting to LLVM IR is better?

- LLVM is a collection of compiler toolchains

- LLVM IR(Intermediate Representation) is between programming languages and assembly

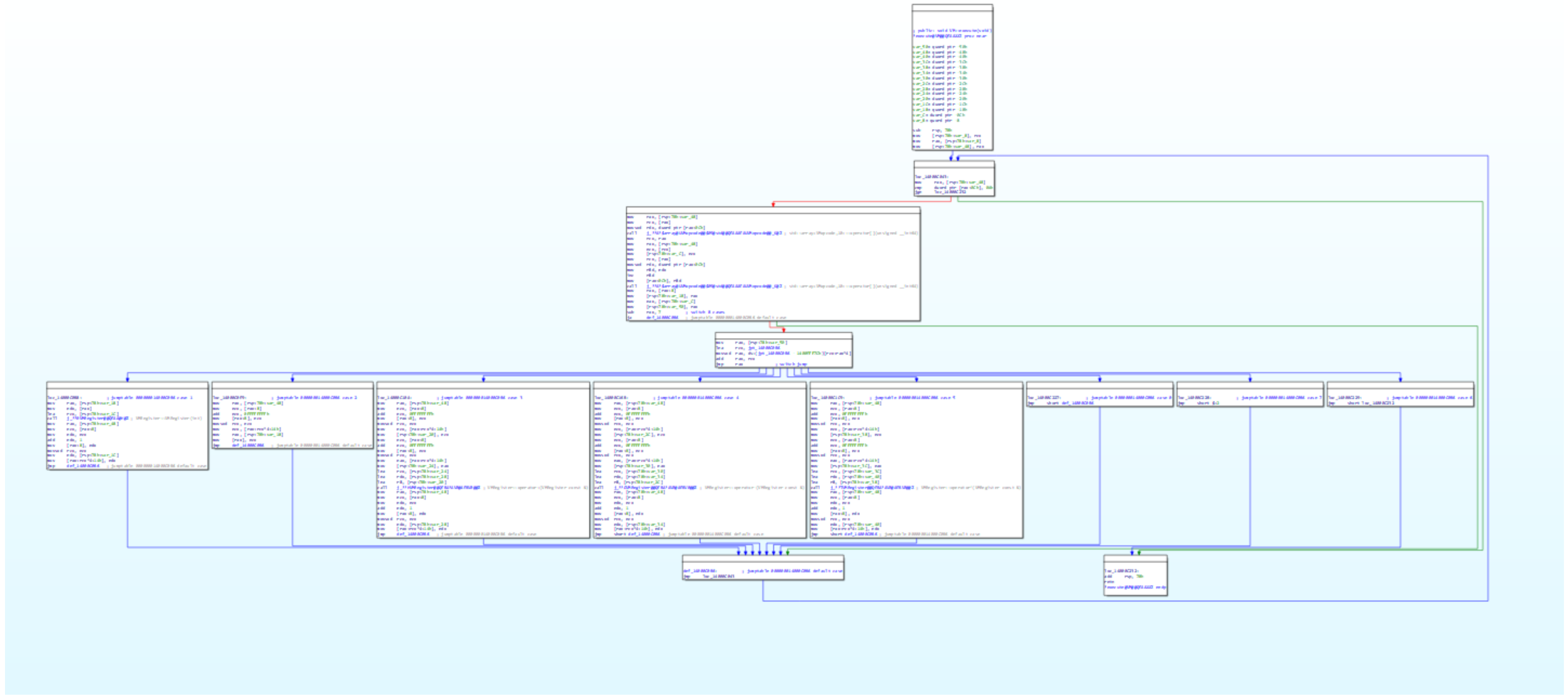
- LLVM has built-in optimizations

- Single Static Assignment(SSA) format

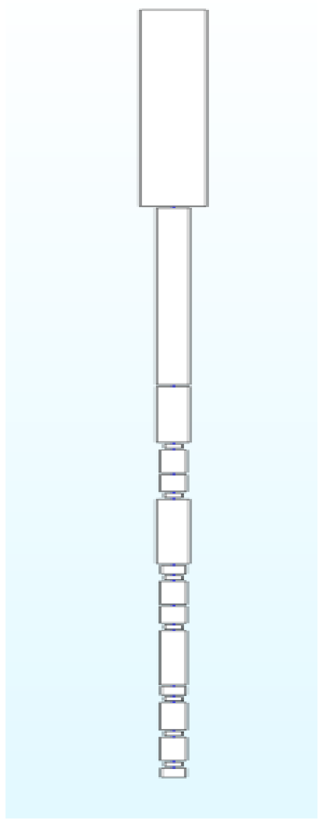
- Being able to re-compile

Target: Toy VM

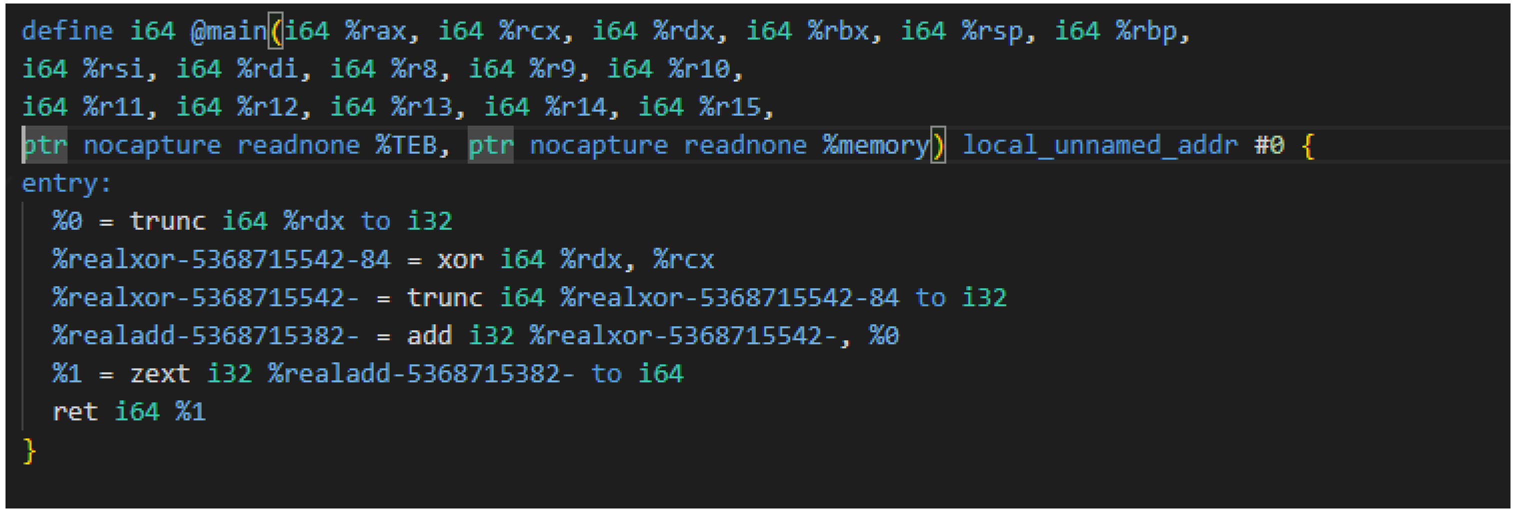
https://godbolt.org/z/Ke33Ph5so



Unrolling the CFG of Toy VM

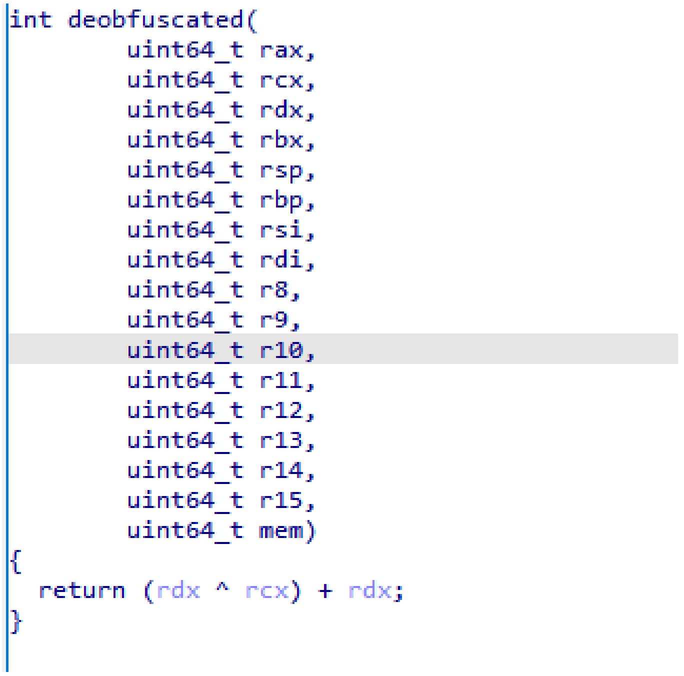


Results, unrolled & optimized



```
define i64 @main(i64 %rax, i64 %rcx, i64 %rdx, i64 %rbx, i64 %rsp, i64 %rbp,
i64 %rsi, i64 %rdi, i64 %r8, i64 %r9, i64 %r10,
i64 %r11, i64 %r12, i64 %r13, i64 %r14, i64 %r15,
ptr nocapture readnone %TEB, ptr nocapture readnone %memory) local_unnamed_addr #0 {
entry:
  %0 = trunc i64 %rdx to i32
  %realxor-5368715542-84 = xor i64 %rdx, %rcx
  %realxor-5368715542- = trunc i64 %realxor-5368715542-84 to i32
  %realadd-5368715382- = add i32 %realxor-5368715542-, %0
  %1 = zext i32 %realadd-5368715382- to i64
  ret i64 %1
}
```

Results, compiled & decompiled



```
int deobfuscated(
        uint64_t rax,
        uint64_t rcx,
        uint64_t rdx,
        uint64_t rbx,
        uint64_t rsp,
        uint64_t rbp,
        uint64_t rsi,
        uint64_t rdi,
        uint64_t r8,
        uint64_t r9,
        uint64_t r10,
        uint64_t r11,
        uint64_t r12,
        uint64_t r13,
        uint64_t r14,
        uint64_t r15,
        uint64_t mem)
{
  return (rdx ^ rcx) + rdx;
}
```

Target: Complex math function

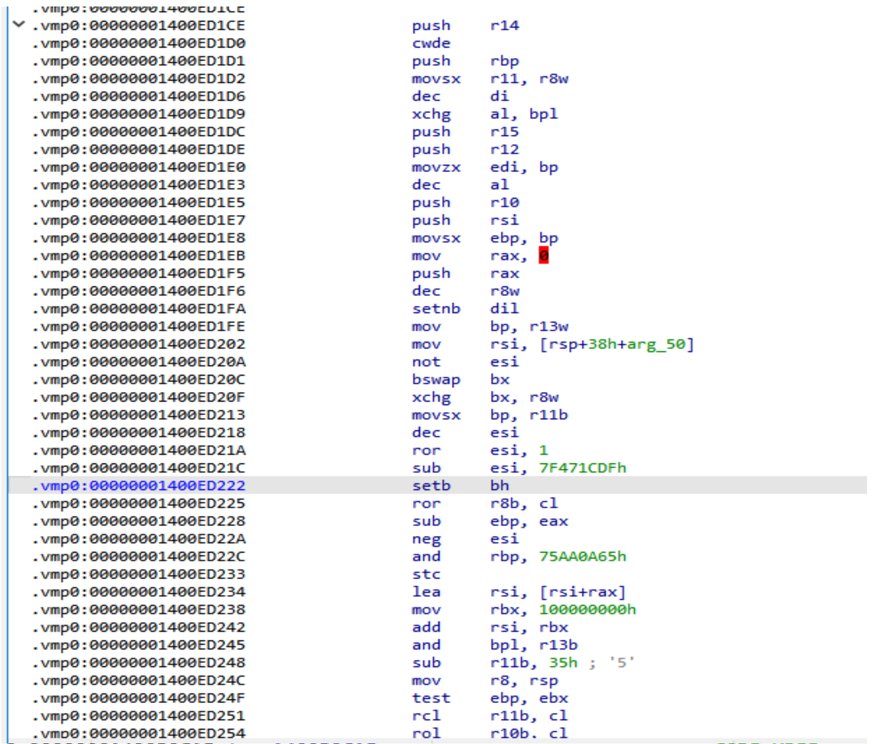
https://godbolt.org/z/TvPnWqzvo

Obfuscated with VMProtect 3.8

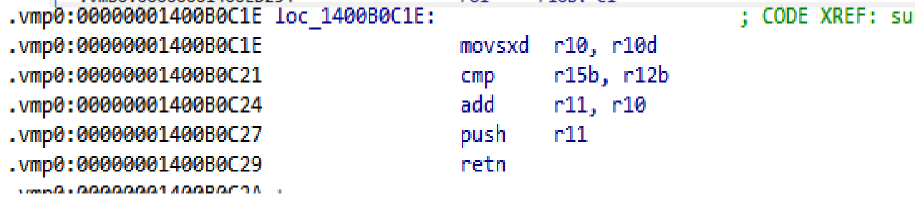
There are 37.077 blocks just like this =>

https://nac-l.github.io/assets/img/vmp38_def_branch.svg

https://nac-l.github.io/2025/01/25/lifting_0.html



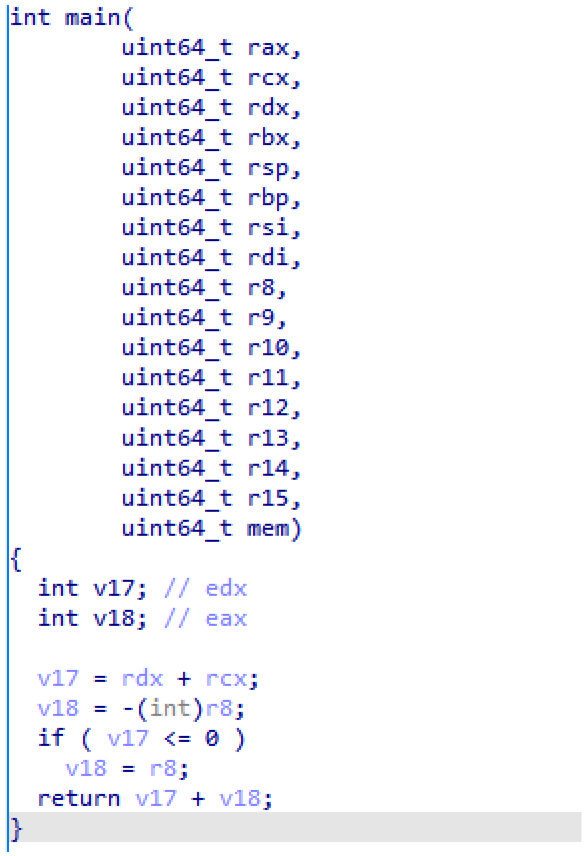
```
.vmp0:00000001400ED1CE
.vmp0:00000001400ED1CE                  push    r14
.vmp0:00000001400ED1D0                  cwde
.vmp0:00000001400ED1D1                  push    rbp
.vmp0:00000001400ED1D2                  movsx   r11, r8w
.vmp0:00000001400ED1D6                  dec     di
.vmp0:00000001400ED1D9                  xchg    al, bpl
.vmp0:00000001400ED1DC                  push    r15
.vmp0:00000001400ED1DE                  push    r12
.vmp0:00000001400ED1E0                  movzx   edi, bp
.vmp0:00000001400ED1E3                  dec     al
.vmp0:00000001400ED1E5                  push    r10
.vmp0:00000001400ED1E7                  push    rsi
.vmp0:00000001400ED1E8                  movsx   ebp, bp
.vmp0:00000001400ED1EB                  mov     rax, 0
.vmp0:00000001400ED1F5                  push    rax
.vmp0:00000001400ED1F6                  dec     r8w
.vmp0:00000001400ED1FA                  setnb   dil
.vmp0:00000001400ED1FE                  mov     bp, r13w
.vmp0:00000001400ED202                  mov     rsi, [rsp+38h+arg_50]
.vmp0:00000001400ED20A                  not     esi
.vmp0:00000001400ED20C                  bswap   bx
.vmp0:00000001400ED20F                  xchg    bx, r8w
.vmp0:00000001400ED213                  movsx   bp, r11b
.vmp0:00000001400ED218                  dec     esi
.vmp0:00000001400ED21A                  ror     esi, 1
.vmp0:00000001400ED21C                  sub     esi, 7F471CDFh
.vmp0:00000001400ED222                  setb    bh
.vmp0:00000001400ED225                  ror     r8b, cl
.vmp0:00000001400ED228                  sub     ebp, eax
.vmp0:00000001400ED22A                  neg     esi
.vmp0:00000001400ED22C                  and     rbp, 75AA0A65h
.vmp0:00000001400ED233                  stc
.vmp0:00000001400ED234                  lea     rsi, [rsi+rax]
.vmp0:00000001400ED238                  mov     rbx, 100000000h
.vmp0:00000001400ED242                  add     rsi, rbx
.vmp0:00000001400ED245                  and     bpl, r13b
.vmp0:00000001400ED248                  sub     r11b, 35h ; '5'
.vmp0:00000001400ED24C                  mov     r8, rsp
.vmp0:00000001400ED24F                  test    ebp, ebx
.vmp0:00000001400ED251                  rcl     r11b, cl
.vmp0:00000001400ED254                  rol     r10b, cl
```



```
.vmp0:00000001400B0C1E loc_1400B0C1E:                       ; CODE XREF: su
.vmp0:00000001400B0C1E                  movsxd  r10, r10d
.vmp0:00000001400B0C21                  cmp     r15b, r12b
.vmp0:00000001400B0C24                  add     r11, r10
.vmp0:00000001400B0C27                  push    r11
.vmp0:00000001400B0C29                  retn
```

Results.

https://godbolt.org/z/qMxP7e55a



```
int main(
        uint64_t rax,
        uint64_t rcx,
        uint64_t rdx,
        uint64_t rbx,
        uint64_t rsp,
        uint64_t rbp,
        uint64_t rsi,
        uint64_t rdi,
        uint64_t r8,
        uint64_t r9,
        uint64_t r10,
        uint64_t r11,
        uint64_t r12,
        uint64_t r13,
        uint64_t r14,
        uint64_t r15,
        uint64_t mem)
{
  int v17; // edx
  int v18; // eax

  v17 = rdx + rcx;
  v18 = -(int)r8;
  if ( v17 <= 0 )
    v18 = r8;
  return v17 + v18;
}
```

How fast it is?

310.327 total instruction

Mergen – 2.8s (exploration) + 2.5s (optimization) = 5.3s

Triton – 29.2s (exploration) + 32.12s (optimization) = 61.3s

Alternate usage ideas

- Optimizing software without source code
- Recompiling existing software into other platforms
- Inserting binary instrumentation for testing

Technical challenges

- Complex, unbounded loops (due to symbolic execution)
- Needs a bigger scope of context than other approaches
- Complex Mixed Boolen Arithmethics

Contributions

- Demonstrated a public, static, and generic methodology for deobfuscating and devirtualizing x86_64 binaries.

- Provided new insights into the inner workings of commercial software protectors.